Database Management System

DATABASES: INTRODUCTION

I. (1.1) Applied overview of Databases:—

The foundational reasoning "why we need database?" is to maintain a data store. So, our primary task is to build a mechanism in order to store data.

In order to get this, let us consider an example of a e-commerce website say "Amazon". Amazon stores humongous amount of data which includes information like:

→ Customer details, product details, purchases/returns, Inventory information, Shipment details, customer reviews, customer services etc.

This is just a small subset of data in Amazon and this data is being used by the amazon recommended system inorder to predict the user choices and providing suggestions.

In order to design a database :

(i) We need to understand the big picture :- We need to know how the pieces of data are related. For the previous example of amazon database, we should know how customers are related to other business logics like login, passwords, phone numbers etc. Note that we represent databases pictorially using E-R diagrams. E-R diagrams provides us this big picture.

(ii) Tables to store the data :- Book-keeping and tables are the traditional ways to store the data in pre-computers. Since, tables are something which are being used from generations and is easy to understand. Therefore, database managers decided to store data in tabular form. Once we have the big picture, we need to find/know "how many tables will store this data in?" and "what should each table contain?" and that too occupying minimal storage in harddisk. The mathematical model which talks about how to store data in tables is called Relational model

(iii) Storage of files in disk :- In order to store all data in computer we need to store it in disks. So, we need to store data in the disk such that we can access it optimally whenever required.

(iv) Search / Retrieve fast :- The retrieving of data stored or searching of data should be fast and therefore, the database is stored in the form of B-Trees and B+-Trees

(v) Multi-system access to your data :- These are situations when data is accessed by multiple systems / programs ~~at a time~~ simultaneously. So database manager should ensure that there is no flaw in the data during this concurrent access of data. This management system is termed as Transaction and Concurrency control system, whose job is to manage the concurrent access to the data.

NOTE :- Language which is used to work on the database data is "SQL". SQL is a ~~go~~ query language based on relational model and is used to add, delete, modify, search, retrieve data from the database

II (2.2) Files vs DBMS :-

File System	DBMS
Data is stored in disk in the format of comma seperated files.	Data is being stored by database management system internally using indexing.
Querying :- Accessing any information from file system is a time consuming process.	Querying :- database also stores info. in the form of file system but it is built based on special datastructure called Indexing ∴ Time required to access any information is comp. less. (Objective of indexing is to make search

- Redundancy:-

File systems contain / have redundant data and therefore consumes more disk space.

- Consistency:-

It is difficult to have consistency of data in file system because interlinking of data is readily available in file system

- Data independence:-

- Security and access control:

- Abstraction and ease of data access:-

- Redundancy:-

It reduces redundancy, internally it avoids users to store the redundant data

- Consistency:-

Database management system is designed such that it takes care of the consistency of data internally because of interlinking of data.

- Data independence:-

DBMS provides a simple interface (framework) to query/search without thinking about how the data is stored. It hides the low level details from the engineers using DBMS.

- Security and Access control:-

A DBMS can have multiple users, and each user have these access permissions, which were managed by DBMS internally.

- Abstraction and ease of data access:-

It is the most important feature of DBMS. We need not have to care about how database works internally. We only asks questions using SQL.

So, it abstracts away all the complexity of DS, algo, file storage.

NOTE: A filesystem can do whatever ~~DBA~~ database does but for that we need to write 1000's of lines of code. In place of that we can simply install DB management system which does everything what file system do, plus huge other advantages.

III (1.3) Tables and Keys:-

Terminology:

1) Relation:- Relation is nothing but a table. Table is a set of rows and columns.
for example

CUSTOMER TABLE:

	CUST_ID	CUST_NAME	CUST_ADDR	CUST_PIN	
Column 1 / field 1 / attribute	1	abc	202,...	940861	→ row 1 / tuple 1
	2	def	202,....	898910	→ row 2 / tuple 2
	3	~~def~~	88, neo...	204102	⋮
	4	xyz	civil lines..	509601	→ row 4

Column 4 → (CUST_PIN)

tuples / records

Fig: 1

The given customer table have 4 rows and 4 columns.

2) Attributes - Attributes are nothing but Columns in relational data base. These are sometimes also called as fields.

3) Tuples:- Rows are called as tuples in relational database. These are also known as records. In a relation there should not be two same tuples (tuple uniqueness constraint)

4) Instance:- The example given for customer table is an instance. That is, the whole table with some data filled in is called instance. "It is a set of tuples/rows along with the table structure."

5) Key:- It is the minimum number/set of attributes/columns to uniquely identify a row/tuple.
Consider the customer table, CUST_ID can uniquely identify a row. Therefore, CUST_ID can become the key of the relation.
- If we consider (CUST_ID, CUST_NAME) → It can uniquely identify a row, but it is not a key because it is not minimum set of attributes. So, key has to be a minimum set of attribute to uniquely identify a row.
- If we consider (CUST_NAME, CUST_ADDR) → It can uniquely identify a row and it is a key because neither CUST_NAME nor CUST_ADDR can uniquely identify a row seperately.

6) Simple key:- key with only one attribute / column.
Ex- CUST_ID is the simple key for above relation.

7) Compound key:- Key with multiple attributes / columns.
Ex- (CUST_NAME, CUST_ADDR) is a compound key.

8) Candidate key :- Set of all unique keys is called as candidate keys.

Ex - { CUST_ID , { CUST_NAME, CUST_ADDR } } are key candidates.
(CUST_ID → key 1; {CUST_NAME, CUST_ADDR} → key 2)

Both key 1 and key 2 can uniquely identify a row/tuple.

9) Primary key :- One of the candidate key that the DB-designer chooses to maintain uniqueness is called primary key.

Ex- Either CUST_ID or {CUST_NAME, CUST_ADDR} could be choosen as primary key.

But there are certain rules while choosing a primary key.
If CUST_ID is choosen as primary key than

- → It cannot be NULL for any tuple/row
- → There should be at most one primary key per table
- → Should be unique for each row/tuple.

These constraints / conditions are called as Entity integrity.
Integrity constraints are properties to be satisfied while inserting, deleting or ~~modify~~ modifying the data in a relation/table.

NOTE- In a table/relation once a primary key has been choosen, it cannot have NULL values but, the remaining candidate key can have NULL values.

10) Alternate / secondary key: These are candidate keys that are not primary keys. That is they are unique to each row but the DB designer do not choosen it as primary, therefore, they can have NULL values now.

11) Relational Schema: Table/relation structure + Integrity Constraints

12) Super key: Candidate key ∪ other attributes.

Example {{CUST_ID}, {CUST_NAME}} → it is a super key

{{CUST_NAME, CUST_ADDR}, {CUST_PIN}} → it is also a Super key

13) Foreign key: It is an attribute or group of attributes in a relational database that provides a link between data in two tables.

Consider the below example :-

CUSTOMER RELATION

CUST_id	CUST_name	CUST_Adds	CUST_Pin
1	abc	202...	940861
2	def	202...	898910
3			
4			

(Referenced)

PURCHASES RELATION

foreign key

Trans_id	CUST_id	Pr_id	Time
12345	1	12	5:00AM
45687	2	11	11:30
25678	5	14	18:00PM

refers to

X NOT VALID

(Referencing)

Fig: 2

Every CUST_id in PURCHASES RELATION has to be present in the CUSTOMER RELATION. The tuple (25678, 5, 14, 18:00) is not valid because CUST_id = 5 is not there in "CUSTOMER".

Hence, CUST_id is the foreign key, as it is a sort of pointer

where each value of CUST_id in PURCHASES points to a value of CUST_id in CUSTOMER Relation, logically.

NOTE:- A foreign key may or may not be the primary key in referencing relation but it should be a primary key in refered relation, and hence, it removes redundancy and avoids inconsistencies

14) Self-referential foreign keys:- A foreign key is said to be self referential, if an attribute or a group of attributes in one table that uniquely identifies a row of the same table. Consider the following example -

EMPLOYEE Relation:-

refers to

Emp_id	Emp_name	Manger_id	DOJ
1	abc	3	06-06-96
2	⋮	⋮	⋮
3			

Fig: 3

Here, Emp_id is the primary key while Manager_id is a foreign key because it refers to the Emp_id (A manager is also a employee of the company). Therefore, Manager_id is a self-referential foreign key.

IV (1.4) Integrity Constraints :-

1) Entity Constraint: It is as follows:

⇒ Every relation must have a primary key

⇒ Every primary key has to be unique

⇒ A primary key should not be NULL.

→ These 3 together are called entity constraints.

2) Domain Integrity Constraints: It gives/tells the Columns/attribute's valid values.

Ex- For the Customer relation in fig: 1, Range of CUST_ID is → {1, 2 . . . . 10,000} and Domain is NUMERIC or INTEGER value.

3) User-defined Integrity Constraints: These constraints are business specific.

Consider the below example of Amazon's PURCHASES relation.

Trans_id	P_pname	C_id	Time	# items
1206 - - 7	Nike shoes	...	...	4

Fig: 1.

Amazon has set constraint over number of items (# items) you can buy of a product (say Nike Shoes). It doesn't allow you to buy more than 4 pair of Nike shoes at a time. Such type of constraints are called user-defined Integrity Constraint. which are Business specific.

4) Referential Integrity Constraint :-

As we can see in fig: 2, CUST_ID in PURCHASES is the foreign key which is refering to CUST_ID of CUSTOMER relation. (which is the referenced relation). Referential integrity constraints corresponds to this foreign key concept and ~~there~~ hence provides several constraints for inserting, deleting and modifying content in the referenced relation, (CUSTOMER relation in our set example) or referencing relation.

* Changes to Referenced relation :-

(i) On Inserting :- On inserting a ~~relat~~ row in a referenced relation, no changes will be made in the referencing relation

(ii) On Deleting :- On deleting a row from the referenced relation, changes will be made in the referencing relation based on the following set condition.

(a) On delete no action :- On deleting 1st row from ~~refere~~ CUSTOMER, no changes are made in PURCHASES. Though referential integrity is broken but we ignore that.

(b) On delete Cascade:- The moment we delete 1st row from CUSTOMER, on delete Cascade will make the cascaded changes in PURCHASES. that is, it will delete all row from PURCHASES refering to the deleted CUST_ID.

In our example on deleting tuple of CUST_ID = 1 from CUSTOMER's will also delete 1st row from the PURCHASES due to cascade delete.

NOTE: It is a dangerous option to go for, therefore we avoid using it unless we know what we are doing. As this can result in loss of data.

On delete no action is also dangerous as it voilates the referential Integrity.

(c) On delete Set NULL:- This is the safest option among all the three available options. It says, on deleting 1st row from CUSTOMER's, ~~Set NULL in the CUST_ID~~ CUST_ID is set to NULL for that particular referencing tuple.

NOTE:- There may be a condition where foreign key is a part of the primary key in referencing relation. In such cases On delete Set NULL is not a feasible option to go for.

(iii) On Modifying / editing :- On modifying / editing a tuple is in referenced relation, Changes will be made in referencing relation based on following set condition.

(a) On modify no action :- If a row / tuple is modified in CUSTOMER, no action will be taken in the referencing relation. It is not widely used.

(b) On modify Cascade :- It is the safest option in case of modification of data in referenced relation. It will cascade the changes to the referencing relation.

(c) On modify set NULL :- It is not widely used. It will set the referencing attribute in the particular tuple if any changes were made in referenced relation.

NOTE- On modify cascade is a time consuming operation because editing the changes in one relation will / may also result editing the changes in other relation. Hence, it is time consuming.

NOTE- A relation cannot have two same tuples. every tuple should be unique (they should differ by atleast one attribute ie. primary key)

ex- R_1

A	B
a_1	b_1
a_1	b_1
a_2	b_1

same ← { } X → This is not possible as (1) and (2) are same

Changes to the Refencing Relation:-

(i) **On inseting :-** On inseting any tuple/row in refeencing relation PURCHASES, first it will check CUSTOMER's relation if the inseted CUST_ID exists in CUSTOMER or not. In fig:2, the entry made in PURCHASES with CUST_ID = 5 is not valid because 5 doesnot exists in CUSTOMER's. and hence, it will through an error.

Therefore, on inseting check for any violation.

(ii) **On deleting :-** On deleting a row from refeencing relation will make/brings no change in the refeenced relation

(iii) **On modifying/editing :** We need to check for any violations regarding foreign key constraint in refeenced relation. If any, DBMS will raise an error.

Example- What are the rows getting deleted on deleting (2,4) from R

Let R be a relation given as follows and DB ~~manag~~ manager set the constraint as ON DELETE CASCADE.

foreign key

S.No.	A_1	A_2
1	2	4
2	3	4
3	4	3
4	5	2
5	7	2
6	9	5
7	6	4

$\Rightarrow$ Since, the constraint is ON DELETE CASCADE.

Therefore, on deleting (2,4)

* Initially ~~tuple~~ 4th and 5th tuple will get delete as they are refering to '2' so (5,2) and (7,2) will get delete.
* Deletion of (5,2) and (7,2) will lead to the multilevel cascading, so all tuples refering to 5 and 7 will also get deleted

Therefore, total number of tuples deleted from R, on deleting (2,4) are 3, and they are :- (5,2), (7,2) and (9,5)

This is the reason why On delit cascade is a dangerous option as it leads to the multilevel cascading and hence, there is loss of data.

V (1.5) Solved Problems :-

(Q1) $R(A_1, A_2, A_3, \cdots A_n)$ (This represents a relation R having n attributes)

(a) Number of super keys possible, if candidate keys are $\{A_1\}$?

$\Longrightarrow$ Since, Super keys are $\Rightarrow$ Candidate keys $\cup$ Other attributes

$A_1 \cup \phi$ ← it is candidate key given in question.

$A_1 \cup A_2$

$A_1 \cup A_3$

$\vdots$

$A_1 \cup A_2 A_3$

$\vdots$

$A_1 \cup A_2, A_3 \cdots A_n$

So A_1, can be merged with remaining $(n-1)$ items in relation R. So this is nothing but the concept of power set. for a set of n elements, no. of subset we can create using n elements are 2^n. Similarly, here for $(n-1)$ elements (bcoz A_1 is already taken) we can have subset $= 2^{n-1}$

When each of these 2^{n-1} subsets get combined with A_1 it will become a super key.

Hence, # super keys given candidate key as $\{A_1\}$ are 2^{n-1}

(b) For given relation R, what is the number of super keys if, candidate keys are $\{A_1, A_2\}$

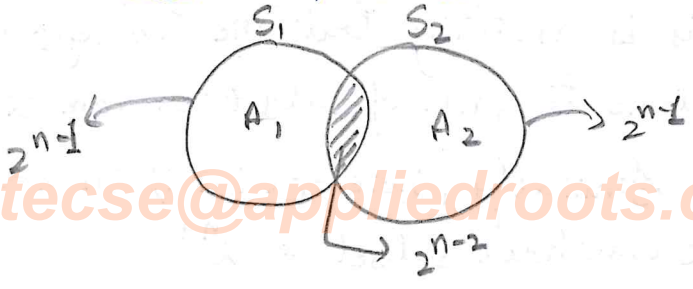
$\Rightarrow$

Candidate Keys $\cup$ Other attributes

$A_1 \cup \phi$
A_2
A_2, A_3
$A_2, A_3 \cdots A_{n-1}$
} 2^{n-1} super keys

$A_2 \cup \phi$
A_1
A_1, A_3
$A_1, A_3, \cdots A_{n-1}$
} 2^{n-1} super keys.

As we can see, we have counted set $\{A_1, A_2\}$ twice as it appears in both A_1 and A_2, Similarly there are many other subsets which we have counted twice.
(Note set $\{A_1, A_2\} \Leftrightarrow \{A_2, A_1\}$)
So Acc. to the principle of inclusion and exclusion from set theory we need to remove the intersection part.



only

So, $|S_1 \cup S_2| = |S_1| + |S_2| - |S_1 \cap S_2|$ (acc. to the inclusion, exclusion Principle)

$|S_1 \cap S_2|$ → these are those superkeys which are counted twice and contains A_1, A_2 together.

Therefore, # of superkeys $= 2^{n-1} + 2^{n-1} - 2^{n-2}$

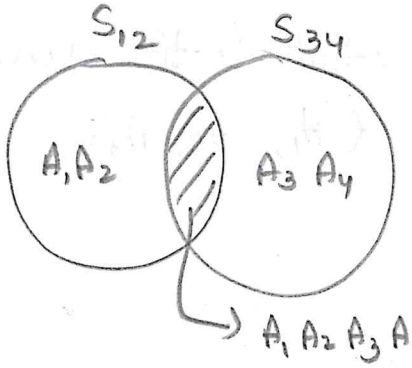
$= 2 * 2^{n-1} - 2^{n-2}$

$= 2^n - 2^{n-2}$

(c) For given relation R, what is the number of super keys if candidate keys are $\{\{A_1, A_2\}, \{A_3, A_4\}\}$

$\Rightarrow$ With $\{A_1, A_2\}$ as candidate key, number of subsets will be (S_{12}) $= 2^{n-2}$

With $\{A_3, A_4\}$ as candidate key, number of subsets will be (S_{34}) $= 2^{n-2}$



$A_1 A_2 A_3 A_4$ are the elements in this intersection. So no. of subsets $= 2^{n-4}$

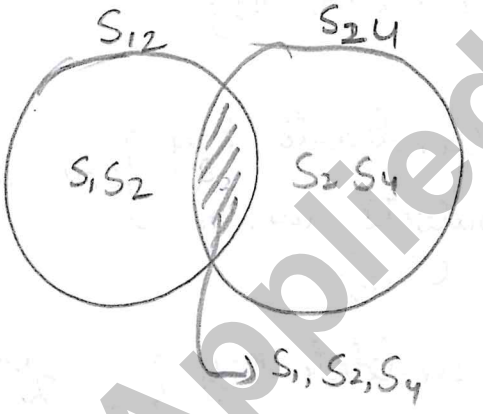
Therefore, no. of superkeys $= 2^{n-2} + 2^{n-2} - 2^{n-4}$

(d) For the given Relation R, what is the number of superkeys if, candidate keys are $\{\{A_1, A_2\}, \{A_2, A_4\}\}$

$\Rightarrow$ For $\{A_1, A_2\}$ as candidate key, no. of subset will be (S_{12}) $\} = 2^{n-2}$

For $\{A_2 A_4\}$ as candidate key, no. of subset will be (S_{24}) $\} = 2^{n-2}$

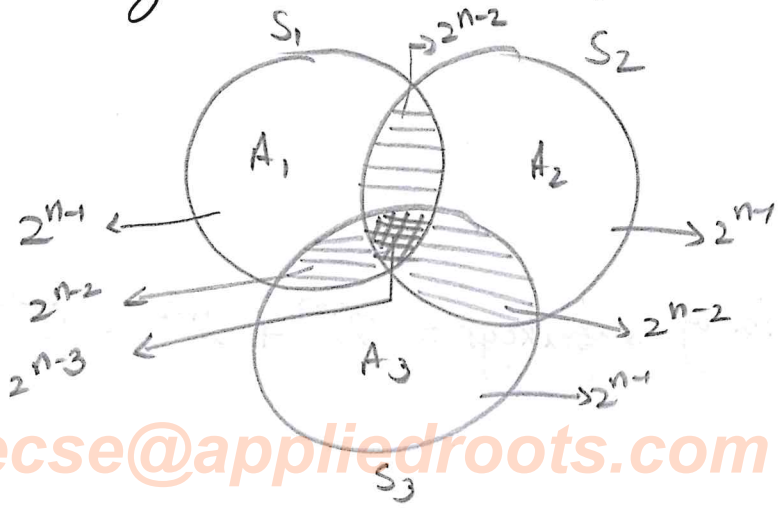
For $\{A_1, A_2, A_4\}$ no. of subsets will be $= 2^{n-3}$



Therefore, no. of superkeys $= 2^{n-2} + 2^{n-2} - 2^{n-3}$

(e) for given relation R, what is the number of superkeys if, candidate keys are $\{A_1, A_2, A_3\}$

$\Rightarrow$



According to the principle of inclusion and exclusion

$$|S_1 \cup S_2 \cup S_3| = |S_1| + |S_2| + |S_3| - |S_1 \cap S_2| - |S_1 \cap S_3| - |S_2 \cap S_3| + |S_1 \cap S_2 \cap S_3|$$

Therefore, no. of Super keys $= 2^{n-1} + 2^{n-1} + 2^{n-1} - 2^{n-2} - 2^{n-2} - 2^{n-2} + 2^{n-3}$

$$= 3 \cdot 2^{n-1} - 3 \cdot 2^{n-2} + 2^{n-3}$$

(Q2) Given a relation R with n attributes, how many Superkeys can be present?

$\Rightarrow$ Given, $R(A_1, A_2, \ldots A_n)$

Since, no information is given regarding candidate keys. Therefore any attribute could be the key

if, A_1 is key $\rightarrow$ # super keys $= 2^{n-1}$

if, A_2 is key $\rightarrow$ # super keys $= 2^{n-1}$

⋮ ⋮

So, total 2^n subsets are possible except NULL set ($\{\}$) can be a super key. (ϕ could not be the super key)

So, # superkey will be $= 2^n - 1$ superkes

↳ because $\{\}$ cannot be a super key.

ENTITY RELATIONSHIP (ER) MODELS & DIAGRAMS

I (2.1) Introduction to ER Diagrams:-

Purpose / objective of ER diagram or model is to get a high level picture of the database. It gives the logical view of the database.

What is an Entity?

Entity is an object that can be uniquely identified.
For ex: In fig: 2, Customer and Purchases are two entities as these are the physical or logical concept.
In an E-R diagram, an entity is represented as a simple rectangular box.

ex → CUSTOMER

→ Attributes:- Attributes are properties / features of an entity. It is typically represented using ellipse.
For example-
On considering fig: 1, we can represent in the form of ER diagram as:

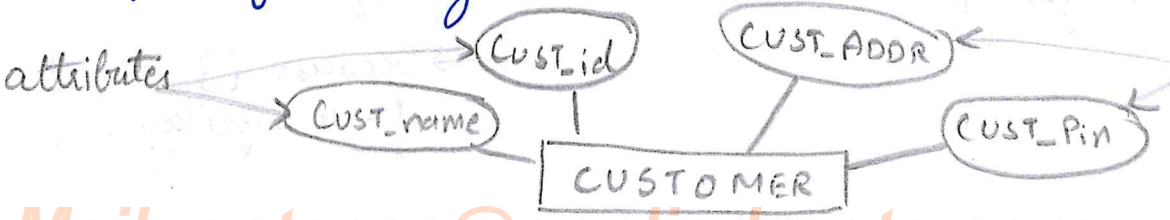


Fig: 5

NOTE: A relationship can also have attributes, those attributes are called declarative attributes.

NOTE: The attributes of an entity are nothing but column names / attribute names in relational model. An each row in the relation / table represents an entity. For the customer example in fig:1, each row represents a customer.

→ Key Attributes:- Key attributes are the one which uniquely identifies a row. It is often represented as an ellipse and is underlined.

For example: In fig: 6, CUST_ID is the key attribute.

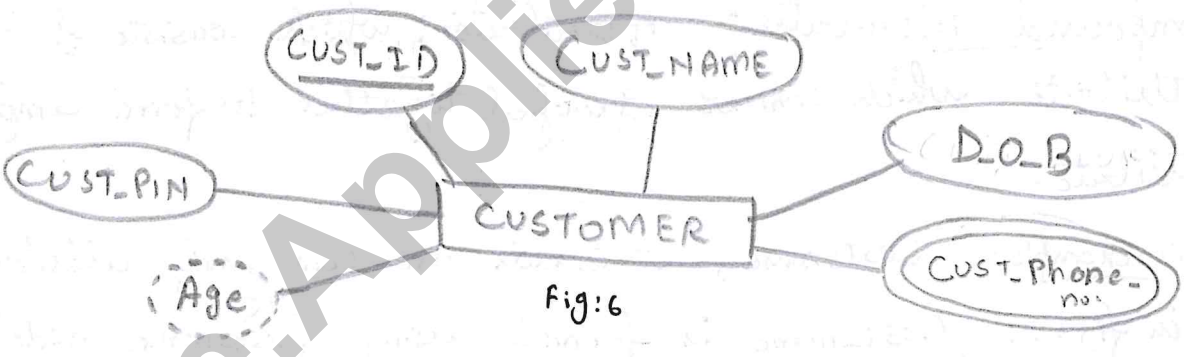


Fig:6

→ Multi-valued Attributes:- Attributes which are having multiple values associated with them are multi-valued attributes. These are typically represented by two concurrent ellipse's.

For example In fig:6, CUST_Phoneno. is a multivalued attribute. A customer can have multiple contact numbers.

→ Derived Attributes :- An attribute whose value is derived / calculated for other attributes are called derived attributes. These are represented by dotted ellipse.

For example: In fig:6, age is an derived attribute, which is calculated / derived from D_O_B (date of birth of a customer)

NOTE :- Derived attributes are not typically stored within the database, physically; insted it can be derived by using an algorithm.

→ Compound Attributes :- Attributes, which consists of other attribute and can be grouped together to form compound attribute.

Foreexample :- CUST_name, CUST_adde are compound attributes in fig:7. CUST_name is formed using first_name, middle_name, and last_name. So all of these constitutes to form the CUST_name.

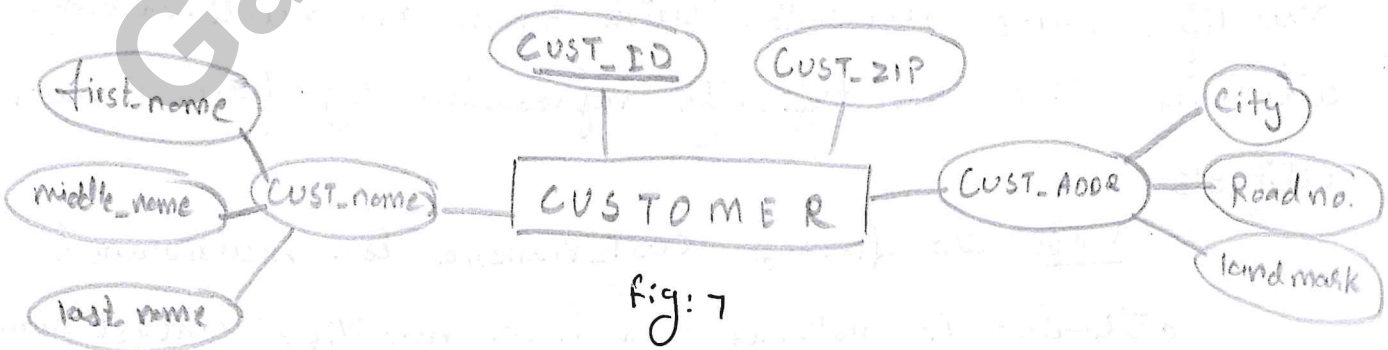


fig: 7

Similarly, CUST_ADDR is a compound attribute as it is formed of City, landmark and Road no.

→ **Relationships** :- A relationship is a situation (in context of databases) which exists in between multiple entities. It is typically represented using a rombus / diamond.

For example:- In fig:8, "buys" is a relationship between customer and products. This could be read as "A customer is ~~buy~~ buys a product is." Therefore, relationship is a way to connect multiple entities.

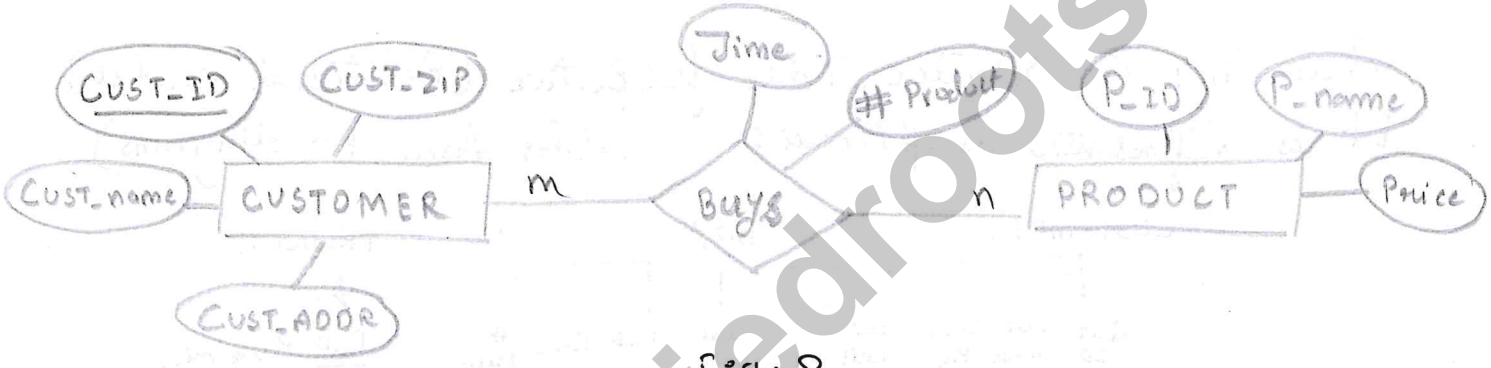


Fig: 8

When a customer buys a product, there is some time stamp associated with it. So, this attribute "time" will be the attribute of "buys" and not of CUSTOMER or PRODUCT because for us to have the time, we need to have both CUSTOMER and a PRODUCT. Similarly, # Product will be the attribute of relationship "buys".

According to set theoretic perspective, (NOTE: all of the database concept can be understood from the concept of sets and relation) it will be represented as follows:

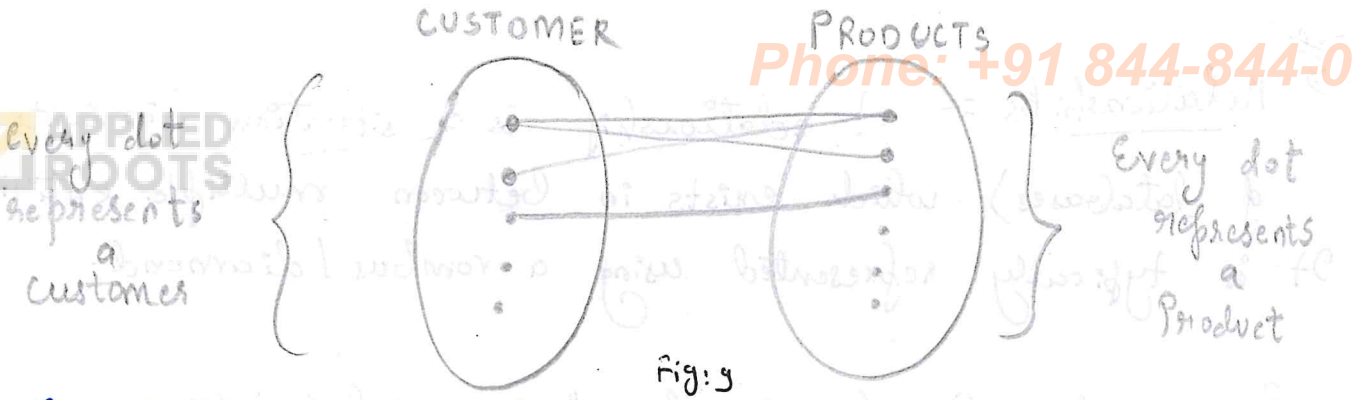


Fig: 9

Acc. to the relation depicted above, many customers can buy a product, many products could be bought by a single customer or many products could be bought by many customers.

According to relation / Table perspective (Relational model) it is represented as follows: (Tables from E-R diagrams)

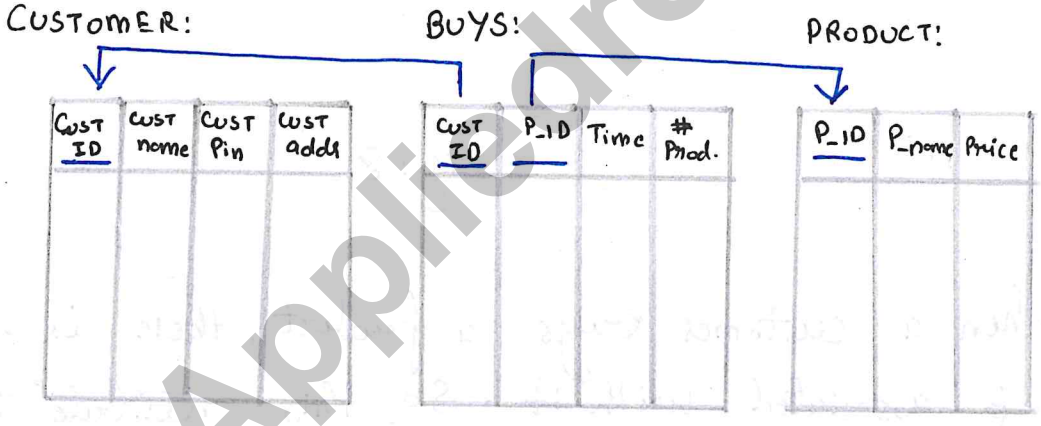


Fig: 10

Since, the given relation is many-to-many in between CUSTOMER and PRODUCTS, therefore, we need a separate table/relation for "buys", Otherwise it would be possible to store all information using two relations.

(This concept is further explained in ~~the~~ detail)

Note, In BUYS relation, CUSTID and P_ID are the foreign key to CUSTOMER and PRODUCTS.

→ Total and partial participation :-

This participation is about entities participating in a relationship. Consider the below E-R diagram of 'Employee' and a 'Team' related to each other using relationship "manages"

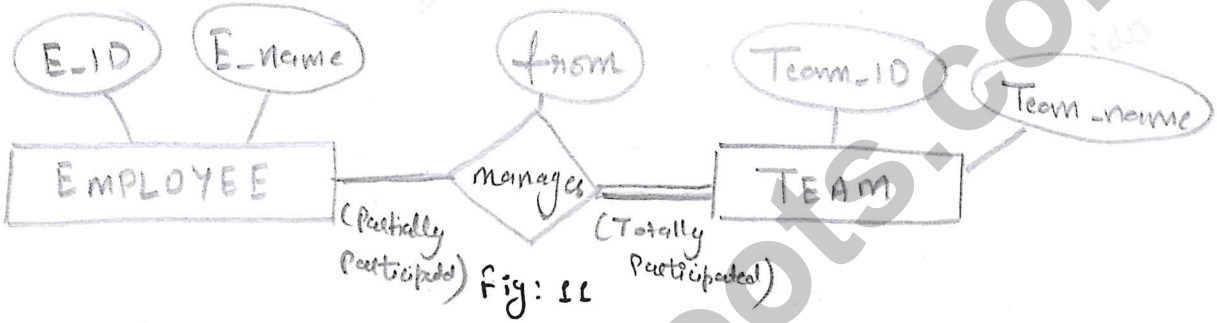


Fig: 11

For the given E-R diagram in Fig: 11, the diagram could be explained as " Every team needs to have a manages and a Employee/ manager manages a team from some give date "

So, here total participation could be understood as — For every 'Team' there should be a manager OR Every member of Entity 'Team' should participate in the relationship with Entity 'Employee'. Hence, 'Team' is said to be totally participating with the "manages" relationship.

Note: Total participation is represented using two parallel lines (as we can see in figure fig: 11)

While, Entity Employee doesnot participated totally with relationship manages because every Employee is need not suppose to be a manager.

Since, there are employee who do not manages any team, therefore, entity employee is having Partial participation with the relationship "manages". It is represented using a single line (it is set default).

Based on set theoretic concept it could be represented as:

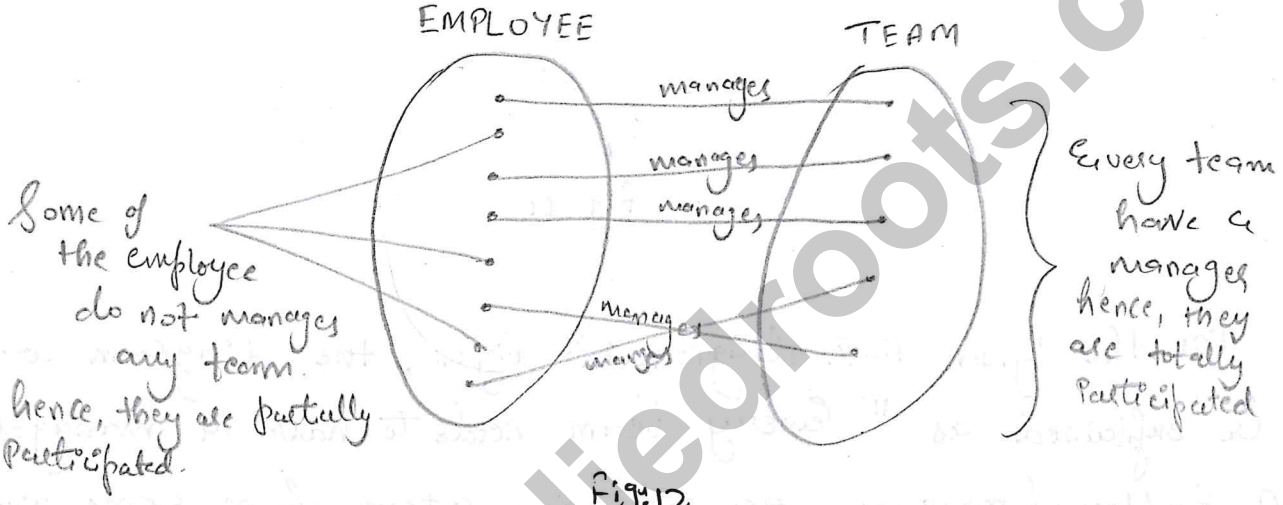


Fig: 12

Note

NOTE:- We can think of total and partial participation as a form of User-defined Integrity constraints. (As it depends as per business requirement)

II (2.2) Cardinality of Relationships and Constructing

Minimal Tables / Relations :—

The 4 possible cardinalities of Relationship are:-

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

One-to-One Relationship :- (1:1)

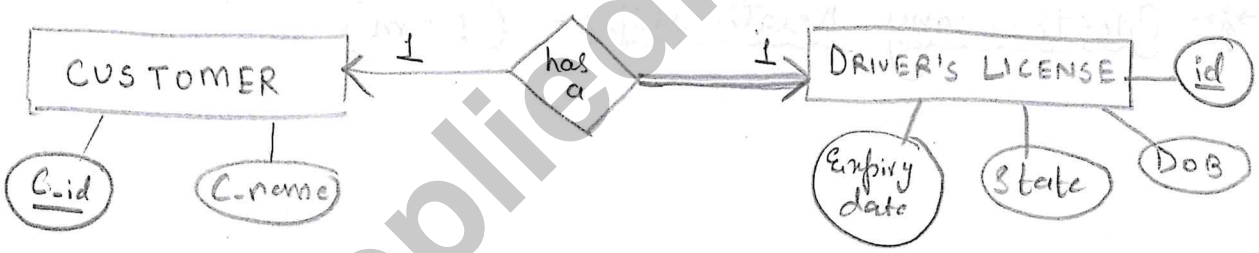


Fig: 13

One-to-One relationship is represented using an 'arrow' (→, ←)
In Fig:13, as we can see both customer and driver's license is having an arrow which means "The entity "customer" is participated in the relation "has a" in a one-to-one fashion" and it could be read as:

"Each customer has exactly one driver's license and every driver's license is associated with exactly one customer"

Fig: 14, depicts the set theoretic perspective of E-R diagram given in Fig: 13.

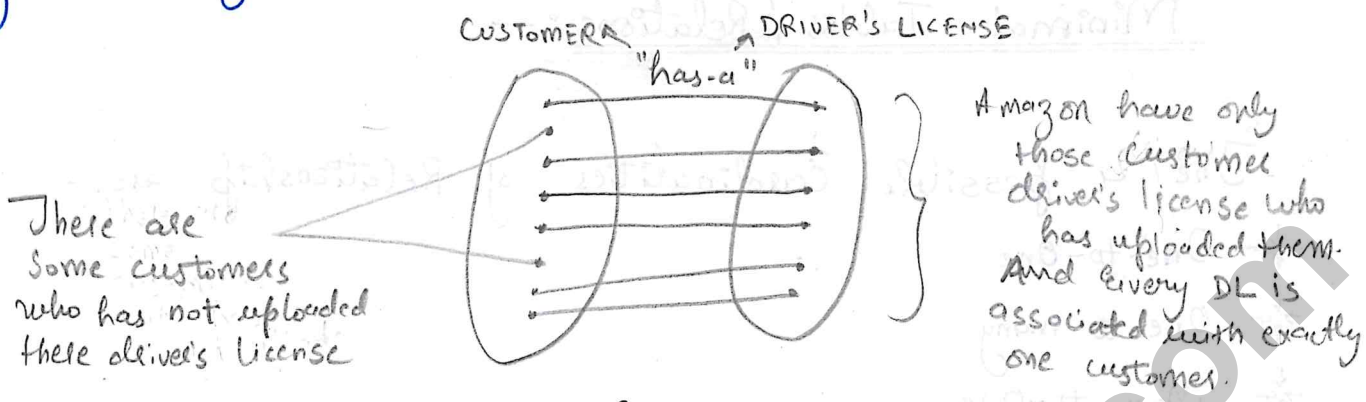


Fig: 14

As, we can see in Fig: 14, DL is totally participated with "has-a" while customers are partially participated with relationship "has-a".

One-to-many Relationship:- (1:m)

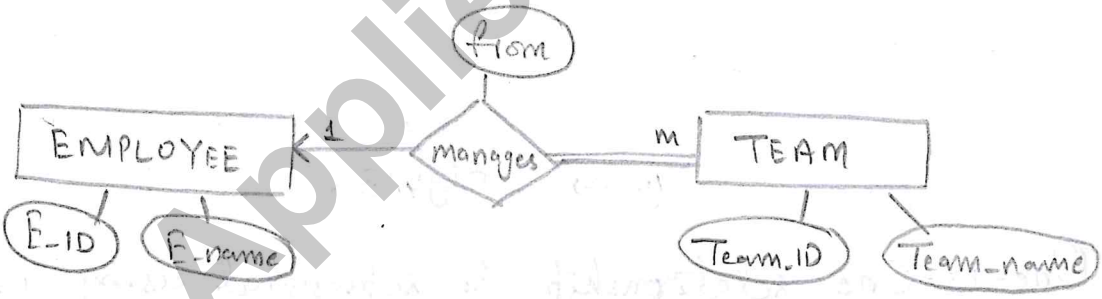
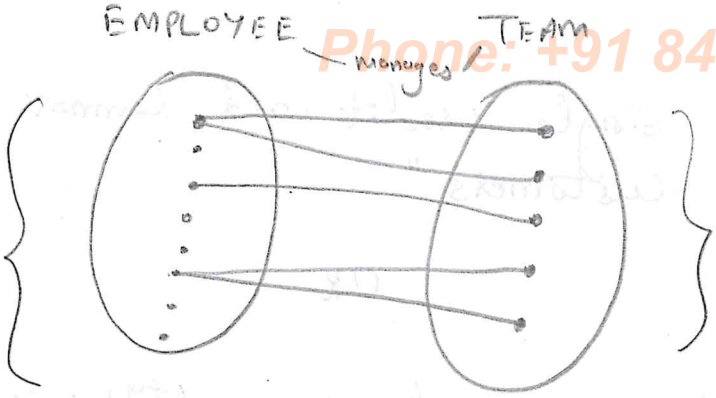


Fig: 15

Single or double line by default depicts the many part of the relationship. This relation is one-to-many because "there are some employees who manages more than one team while there is only one manager to manage a team"

Fig: 16 depicts the set theoretic perspective of E-R diagram shown in fig: 15.



A employee can manage more than 1 team and there are some employees who do not manages any team

Every team is managed by exactly one employee

Fig:16

NOTE:- Any of these 4 cardinalities of relationship can have both side partial, both side total, one side partial and one side total participation. They can have any permutation or combination of possibilities of a E-R diagram.

For ex- One-to-one can have any of the 3 possible condition of participation in an E-R diagram.

Many-to-one Relationship:- (m:1)

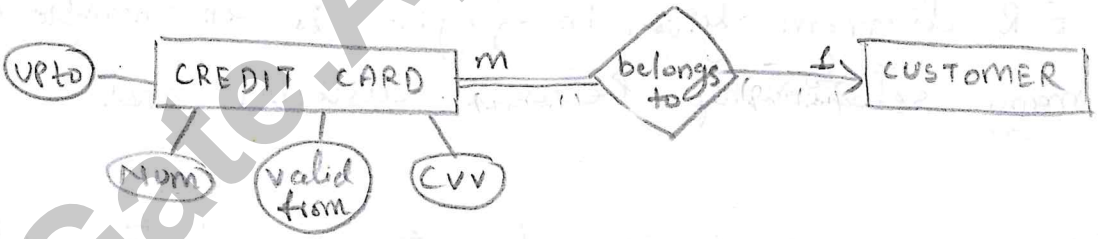


Fig:17

It is related to one-to-many relationship (the difference is due to the perspective)

" Any (multiple) number of Credit cards can belongs to a customer (and their might be some customer who do not have any Credit card but every Credit card we have in our system has to be associated with an employee (total Participation))

while a single credit card cannot belongs to the multiple customers"

OR

"A customer can have multiple credit cards but a credit card cannot have multiple customers"

Fig:18, defict its set theoretic perspective :-

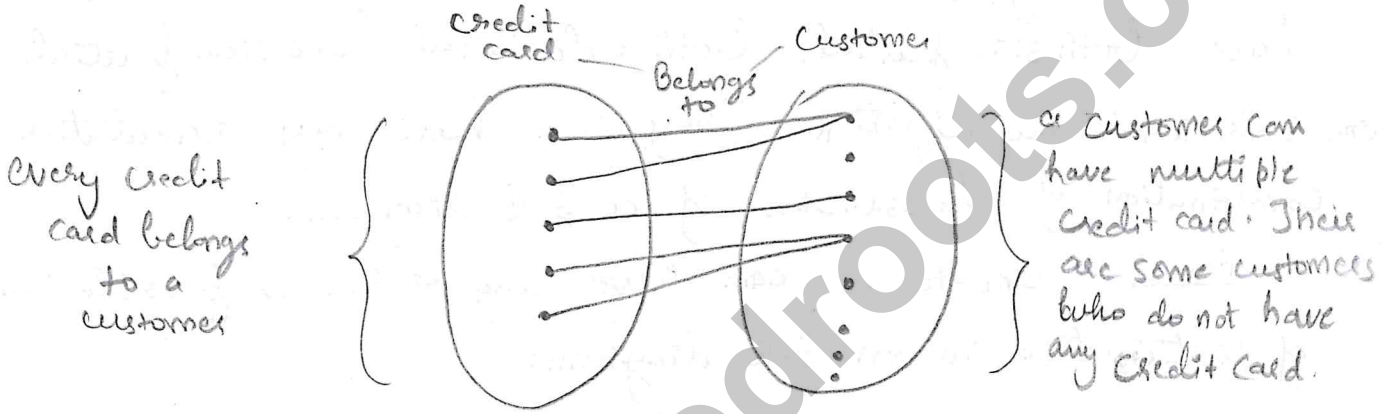


fig:18

Many-to-Many Relationship :- (m:n)

The E-R diagram given in fig:8 is an example of many-to-many relationship between customer and Product.

"A customer can buy many/multiple products and a product could be bought by many/multiple customers".

fig:19 defict its set theoretic perspective :-

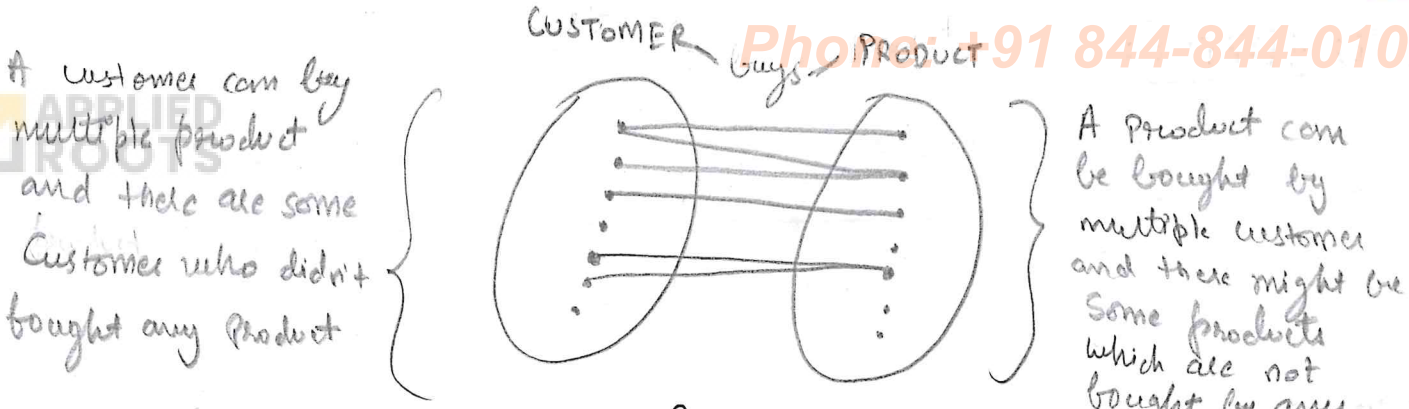


Fig: 19

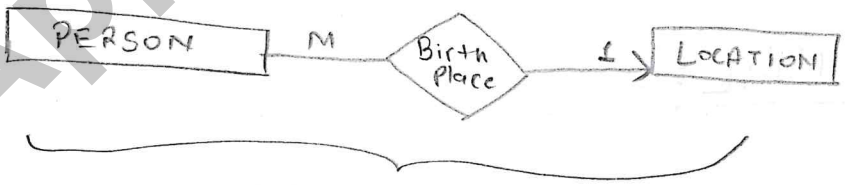
→ Alternative Conventions :-

What we have used till now to represent the cardinality of a relationship is :-

→ ⇒ To defict 'one'

—— ⇒ To defict 'many'

for example: For many-to-one :-

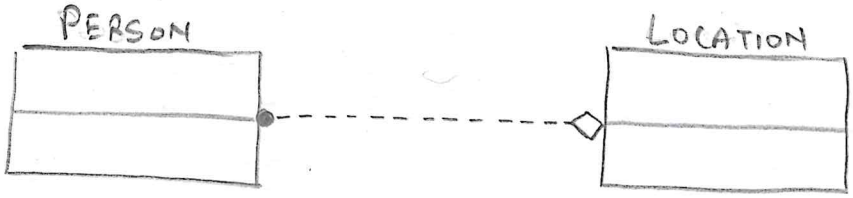


This is one of the way to represent cardinality.

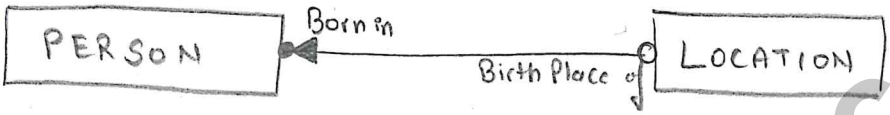
Other popular alternative conventions are:-

(i) Chen :

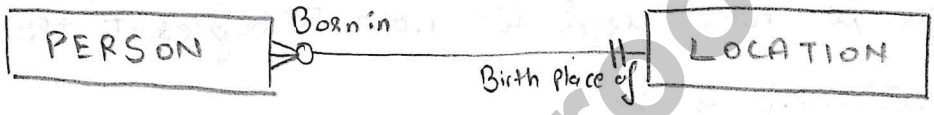
(ii) IDEFIX:



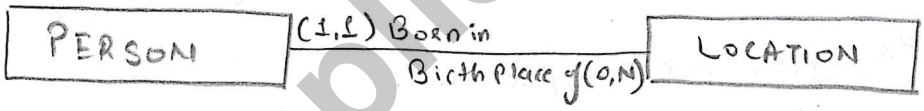
(iii) Bachman:



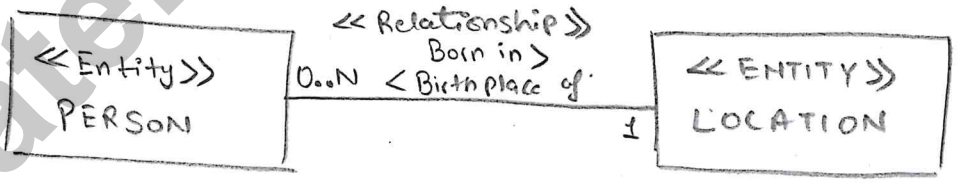
(iv) Martin / IE / Crow's Foot:



(v) Min-Max / ISO:



(vi) UML:



NOTE:- All of the above alternative Conventions are the example for many-to-one relationship.

(8) What is the minimum # relations for every possible combination of cardinality and participation of between ~~two~~ entities?

Imp

→ Constructing tables from E-R Diagrams :-

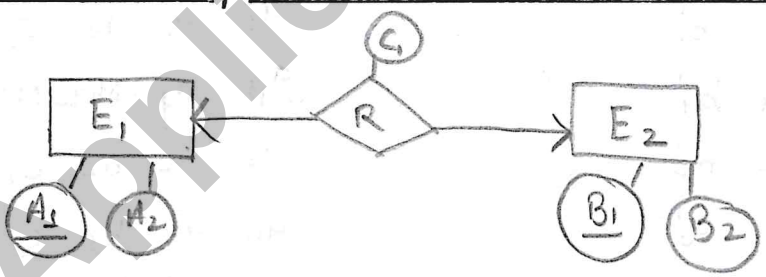
Given an E-R diagram we need to look for cardinality, participation, keys etc in order to construct tables from that. Along with this there is some set of rules defined for constructing a table / relation. These are as follows:

[1] We need to minimize the number of table / relations.

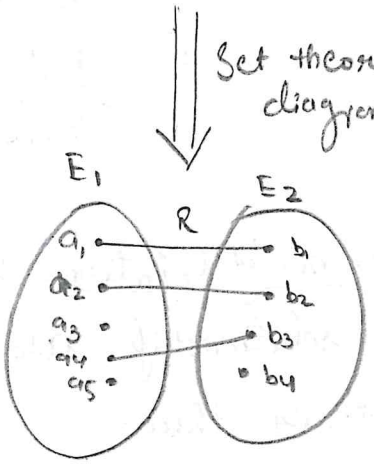
Imp [2] Every relation / table should have a primary key.

Imp [3] Each cell in table / relation must have only one value. (Cells can have NULL value except the primary key cells)

(i) One-to-one relationship + Partial-Partial participation :-



Set theoretic diagram



Check for 1 Relation:

A_1	A_2	B_1	B_2	C_1
a_1	a_1'	b_1		
a_2	a_2'	b_2		
a_3	a_3'	NULL		
a_4	a_4'			
a_5	a_5'			

⇒ Check for 1 relation is possible or not:

A_1	A_2	B_1	B_2	C_1
a_1	a_1'	b_1	b_1'	c_1
a_3	a_3'	NULL	NULL	NULL
NULL	NULL	b_4	b_4'	NULL

Cannot become key ← (NULL, NULL, b_4, b_4', NULL) ×

As we can see, all of them contain NULL values hence neither A_1 nor B_1 could be the primary key. Hence, it is not possible to set them into 1 relation. (It is voilating the 2nd rule [2])

⇒ Check for 2 relation is possible or not:

Table1 - refers Table2 -

B_1	B_2
b_1	b_1'
b_4	b_4'
b_2	b_2'
b_3	b_3'

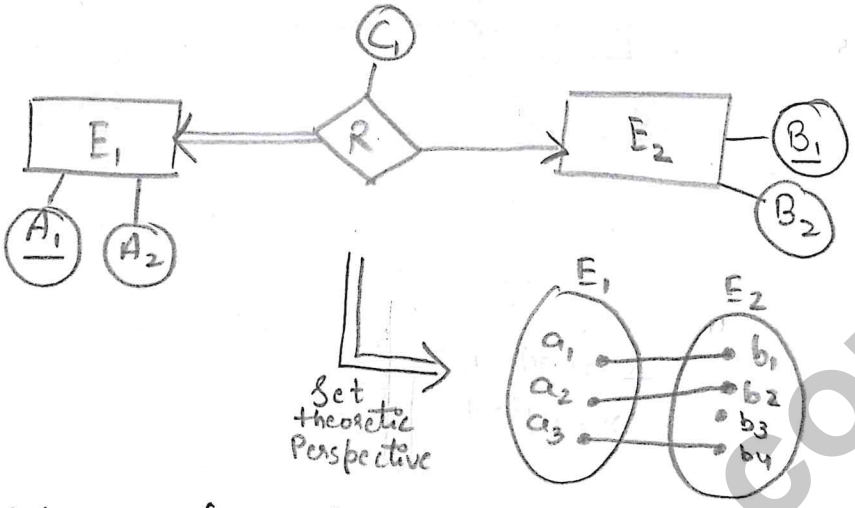
Key values are not NULL

A_1	A_2	B_1	C_1
a_1	a_1'	b_1	c_1
a_3	a_3'	NULL	NULL
a_2	a_2'	b_2	c_2
a_4	a_4'	b_3	c_3
a_5	a_5'	NULL	NULL

Key values are not NULL

Hence, minimum no. of relations required for one-to-one, both side partial relationship are two. And maximum number of relation is ovviously three.

(ii) One-to-One relationship + atleast one side total Participation



⇒ Check for 1 relation is possible or not:

A_1	A_2	B_1	B_2	C_1
a_1	a_1'	b_1	b_1'	c_1
a_2	a_2'	b_2	b_2'	c_2
a_3	a_3'	b_4	b_4'	c_4
NULL	NULL	b_3	b_3'	c_3

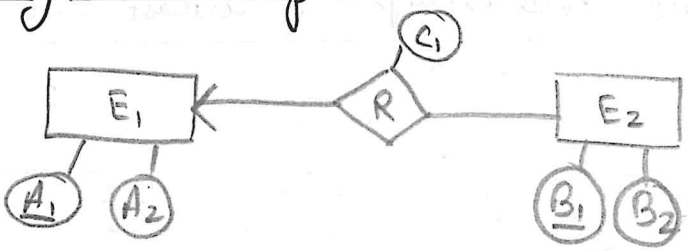
Cannot become Key

B_1 can become Key

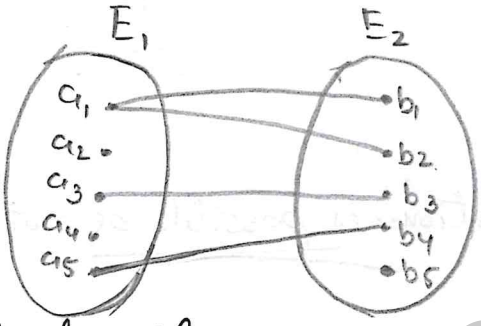
As, we can see there is no NULL value for attribute B_1, therefore, it B_1 can become the primary of this table.
Hence, minimum no. of relation for given condition is 1.

Note:- If the case given is one-to-one + Both side total then definetly one relation is required to represent it, and the key for that relation could be either A_1 or B_1 as none of the attributes can have NULL values.

iii) One-to-Many Relationship + Partial-Partial Participation:-



⇓ Set theoretic Perspective



⇒ Check for 1 relation is possible or not:

A_1	A_2	B_1	B_2	C_1
a_1	a_1'	b_1	b_1'	C_1
a_1	a_1'	b_2	b_2'	C_2
a_2	a_2'	NULL	NULL	NULL
a_3	a_3'	b_3	b_3'	C_3
a_4	a_4'	NULL	NULL	NULL
a_5	a_5'	b_4	b_4'	C_4
NULL	NULL	b_5	b_5'	NULL

None of these could become primary key.

Since, all the attributes has voilated the [2] condition hence, 1 relation is not possible.

⇒ Check for 2 relations are possible or not :-

Table 1:

A_1	A_2
a_1	a_1'
a_2	a_2'
a_3	a_3'
a_4	a_4'
a_5	a_5'

Table 2: (refers to A_1)

B_1	B_2	C_1	A_1
b_1	b_1'	c_1	a_1
b_2	b_2'	c_2	a_1
b_3	b_3'	c_3	a_3
b_4	b_4'	c_4	a_5
b_5	b_5'	NULL	NULL

Two relations will work as there are no NULL values for A_1 and B_1 in relation 1 and 2 respectively.
Hence, 2 relations are required in minimum for the given condition.

Nb

NOTE: There are ~~many~~ many ways to split the table but all splits will not work. What would be the result if the above relations are not splitted like this. Let us consider the follow two relations:

Table 1: (refers to B_1 in Table 2)

A_1	A_2	B_1	C_1
a_1	a_1'	$\{b_1, b_2\}$	c_1
a_2	a_2'	NULL	NULL
a_3	a_3'	b_3	c_3
a_4	a_4'	NULL	NULL
a_5	a_5'	b_4	c_4

this is having 2 values

Table 2:

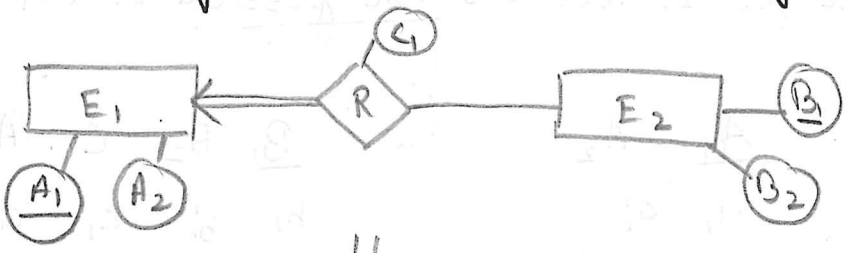
B_1	B_2
b_1	b_1'
b_2	b_2'
b_3	b_3'
b_4	b_4'
b_5	b_5'

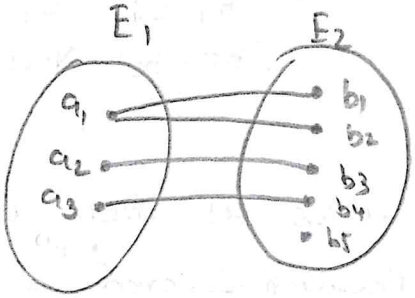
It violates the [3] rule and therefore such split of relations is not possible.

(iv) One-to-many relationship + One side total participation:

Case 1:

One-to-many + total participation on "one" side.





⇒ Check for 1 relation is possible or not:

A_1	A_2	B_1	B_2	C_1
a_1	a_1'	b_1	b_1'	C_1
a_1	a_1'	b_2	b_2'	C_2
a_2	a_2'	b_3	b_3'	C_3
a_3	a_3'	b_4	b_4'	b_4
NULL	NULL	b_5	b_5'	NULL

× ×

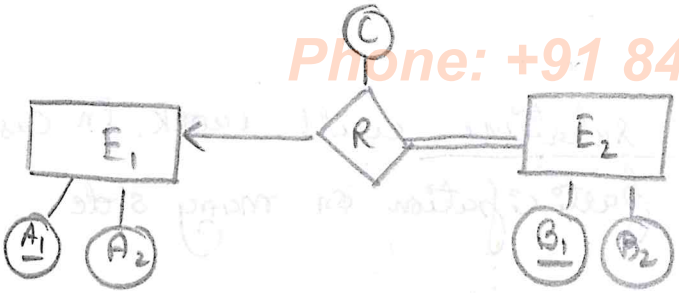
↓

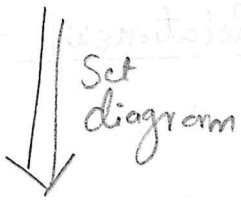
This could be the Primary key

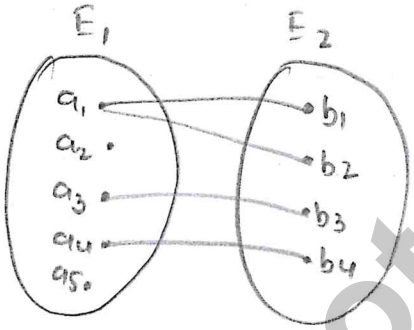
Hence, 1 relation is sufficient for one-to-many and total participation is on the "one" side.

Case 2:

One-to-many
+
total participation
on many sides







⇒ Check for 1 relation is possible or not:-

A_1	A_2	B_1	B_2	C_1
a_1	a_1'	b_1	b_1'	c_1
a_1	a_1'	b_2	b_2'	c_2
a_2	a_2'	NULL	NULL	NULL
a_3	a_3'	b_3	b_3'	c_3
a_5	a_5'	NULL	NULL	NULL

Neither A_1 nor B_1 cannot become the primary key.
Hence, 1 relation is not possible

⇒ Check for 2 relation is possible or not:-

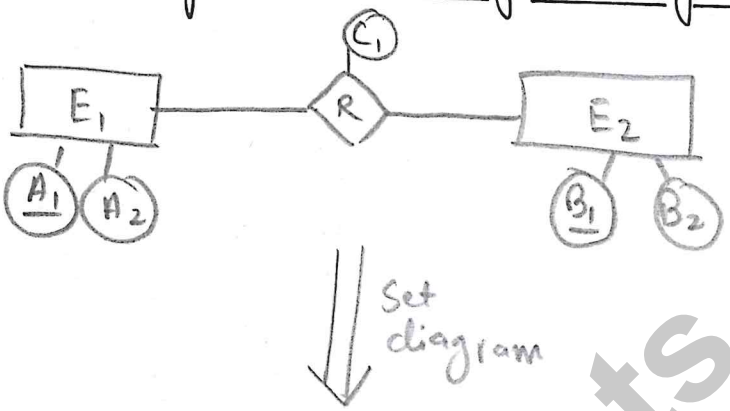
A_1	A_2
a_1	a_1'
a_2	a_2'
a_3	a_3'
a_4	a_4'
a_5	a_5'

B_1	B_2	C_1	A_1
b_1	b_1'	c_1	a_1
b_2	b_2'	c_2	a_1
b_3	b_3'	c_3	a_3
b_4	b_4'	c_4	a_4

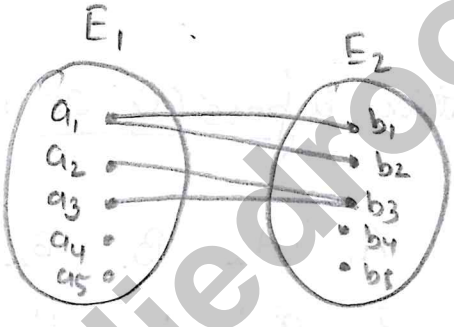
Hence, 2 relations will work in case of one-to-many and total participation on many side.

(v) Many-to-Many Relationship + Both side partial participation:-

Case 1:
Many-many
+
Both Partial.



Set diagram



We can clearly see that neither 1 relation nor 2 relation will work in this case as ~~each~~ a_1 is related to 2 values of E_2 and b_3 is related to two values of E_1 So, in which ever way we break this down in 2 table it will result in some invalid tuples. Hence, 2 relations will not work.

In this case we need 3 relations minimum, ~~two~~ relations to store E_1 and E_2 while third relation is to store relation R (along with its relationship attributes and primary keys of E_1 and E_2)

1)

A1	A2

2)

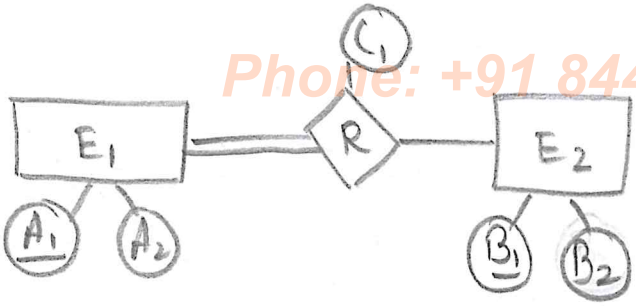
C1	A1	B1

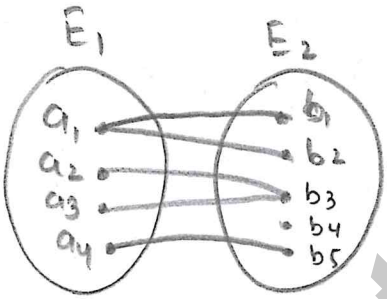
3)

B1	B2

Case 2:

many - many
+
one side
total
Participation





⇒ Check for 2 relation is possible or not :-

A_1	A_2	B_1	C_1
a_1	a_1'	$\{b_1, b_2\}$	c_1

X

B_1	B_2
b_1	b_1'
b_2	b_2'
b_3	b_3'
b_4	b_4'
b_5	b_5'

OR

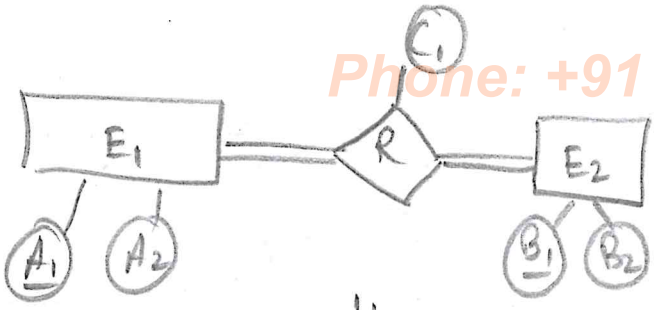
A_1	A_2
a_1	a_1'
a_2	a_2'
a_3	a_3'
a_4	a_4'

B_1	B_2	C_1	A_1
b_3	b_3'	c_3	$\{a_2, a_3\}$

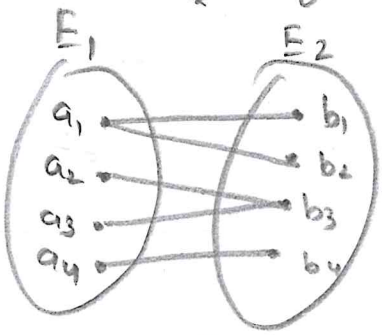
X

Hence, it is not possible to have 2 relations. and therefore we need 3 relations as minimum for the given case

Case 3:
Many-many
+
Both side
total
Participation.



Set diagram



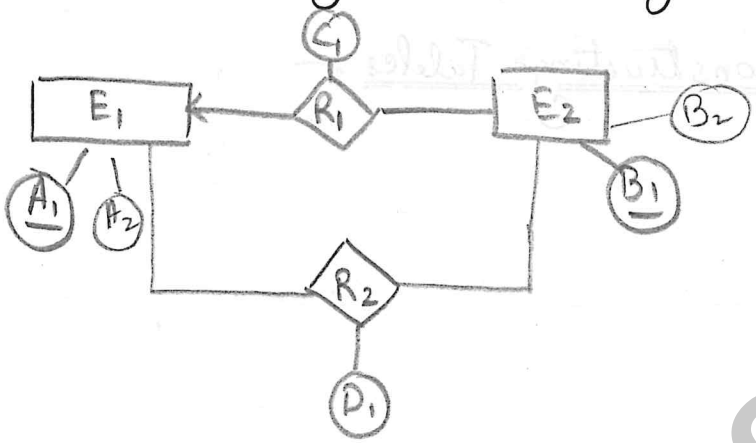
As we can see here 1 relation is possible in minimum case with $\{A_1, B_1\}$ as the primary key. But it will result in lot of redundancy of data.

A_1	A_2	B_1	B_2	C_1
a_1	a_1'	b_1	b_1'	c_1
a_1	a_2'	b_2	b_2'	c_2
a_2	a_2'	b_3	b_2'	c_2
a_3	a_3'	b_3	b_2'	c_3
a_4	a_4'	b_4	b_4'	c_1

NOTE:- For many-to-one, cases will remain same as one-to-many as there is only change in the perspective of taking / considering entities.

NOTE:- For one-to-many and many-to-one + total participation on both sides, only 1 relation is required in minimum case.

(Q 1) What is the minimum number of relations required for the below given E-R diagram?

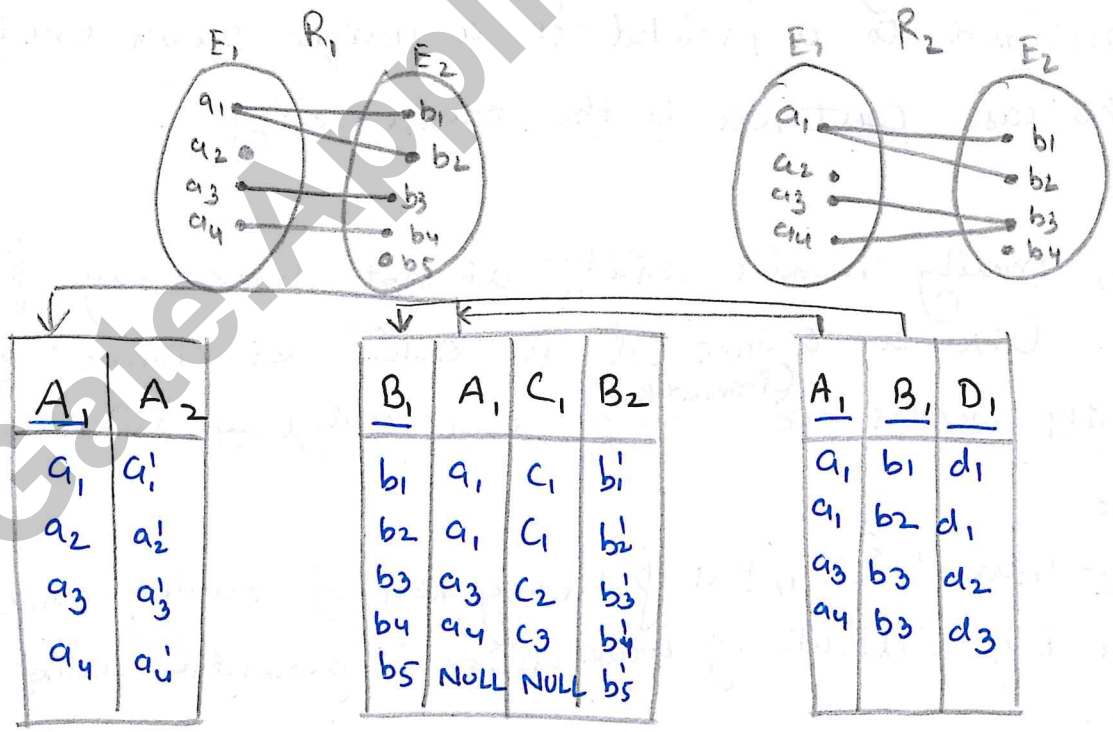


⟹ ~ We can trivially do it in 4 relations.

~ We cannot do it in 1 relation because there is 1:m and m:n relationship with R_1 and R_2 associated respectively.

Hence, we need to check for 2 and 3 relations

→ Check for 3 relations –



A_1	A_2
a_1	a_1'
a_2	a_2'
a_3	a_3'
a_4	a_4'

B_1	A_1	C_1	B_2
b_1	a_1	c_1	b_1'
b_2	a_1	c_1	b_2'
b_3	a_3	c_2	b_3'
b_4	a_4	c_3	b_4'
b_5	NULL	NULL	b_5'

A_1	B_1	D_1
a_1	b_1	d_1
a_1	b_2	d_1
a_3	b_3	d_2
a_4	b_3	d_3

Hence, minimum of 3 relations are required.

Note: We can not do it with 2 tables because these is m:n relation + Partial ∴ we need not able to get 2 tables.

III (2.3) Weak and Strong Entities, self-referential relationship and Constructing Tables :-

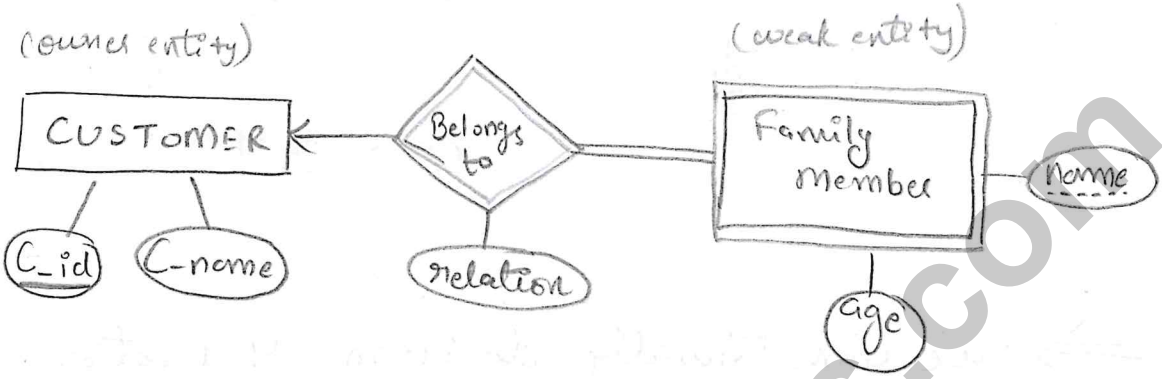


Fig: 20 Weak entity E-R diagram.

In the above E-R diagram, Family-member is a weak entity. It is represented using concurrent rectangles. Note that a weak entity do not have a primary key. A weak entity cannot exist in isolation, it needs to be assigned to a / related to a unique owner entity. In this case customer is the owner entity.

Here, Family member (entity) do not have any primary key. Given a owner id ie 'C_id' and 'name' of a family member we (Primary key) can uniquely determine a tuple.

Note 'name' is not a primary key of Family member, it is a key attribute of weak entity represented using dotted line.

Note: An owner entity should always be a strong entity and relationship between owner and weak entity is always one-to-many with total participation from weak entity.

Note: In Fig: 20, the concurrent diamond / rhombus represents a weak relation between weak entity and its owner strong entity. This relationship is also called identification relationship.

Since, it is a one-to-many relationship with total participation on many side, therefore, minimum number of relation/tables required are 2. One relation for weak entity and weak relationship attributes and other is for owner entity.

(Q2) What is the minimum no. of relations req. for the given E-R diagram?

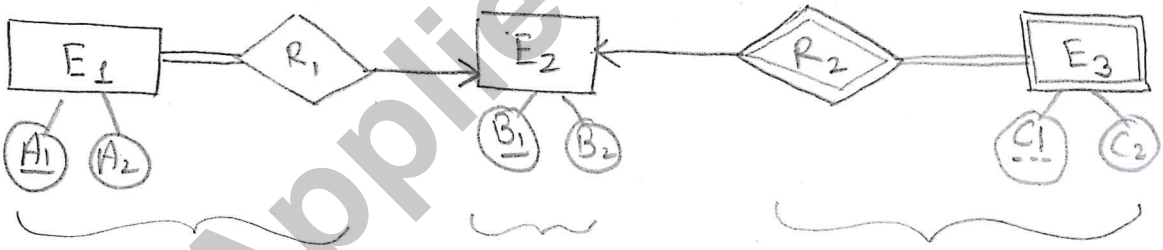


Fig: 21

⇒ As we can see:

R_1 → many-to-one with totale on many side

R_2 → one-to-many with total on many side.

So, we can combine E_1 and R_1 in one relation

E_2 is the second relation

R_2 and E_3 will form the 3rd relation

So, relations will be as follows:

A_1	A_2	B_1

B_1	B_2

B_1	C_1	C_2

So, minimum 3 relations are required for above E-R diagram.

Self-referential relationships:

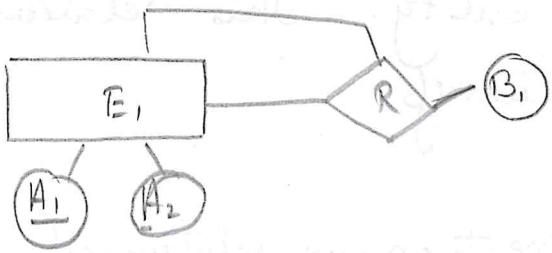
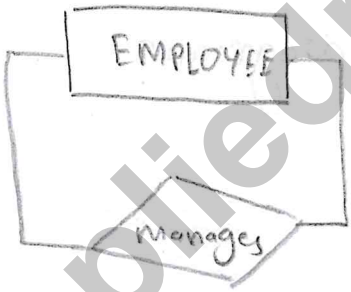


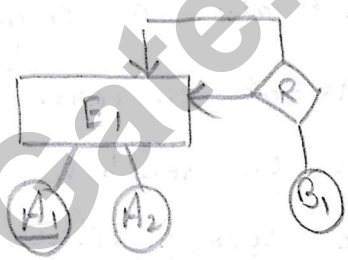
Fig: 21 Self-referential relationship

A relationship which refers to an entity and itself is called a self referential relationship.

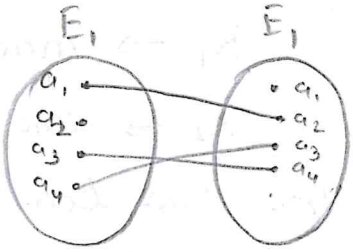
for example:- "An employee manages another employee" is a self-referential relationship.



Case 1: One-to-one + Partial-Partial participation.



Set diagram ⟹



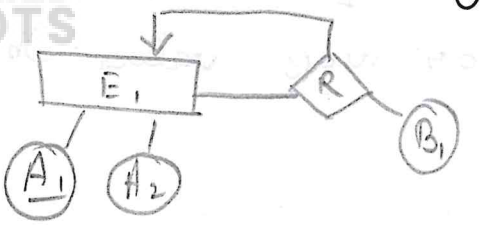
⇒ check for 1 relation possible or not:

A_1	A_2	A_1'	B_1
a_1	a_1'	a_2	b_1
a_2	a_2'	NULL	b_1
a_3	a_3'	a_4	b_2
a_4	a_4'	a_3	b_3

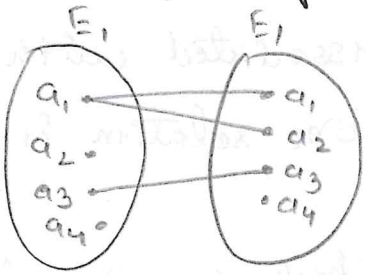
NO NULL ✓ Primary key ✗

Hence, 1 relation is enough for this case. with A_1 as primary key

Case :2: One-to-many + Partial-partial participation:



Set diagram ⟹



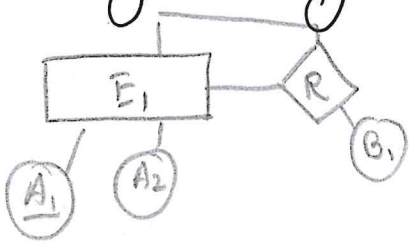
⟹ Check for 1 relation is possible or not:

A_1	A_2	A_1'	B_1
a_1	a_1'	a_1	b_1
a_1	a_2'	a_2	b_1
a_3	a_3'	a_3	b_3
NULL	a_4'	a_4	NULL

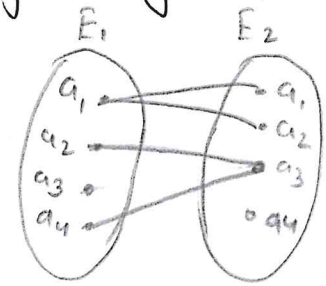
Here, we cannot make A_1 as our primary key because in set diagram we can see a_1 is related to more than one value of itself. Therefore, we will have A_1' as our primary key.

Hence, only 1 relation is required for given condition with A_1' as our primary key.

Case 3: Many-to-many + Partial-partial participation:



Set diagram ⟹



As we can see here both A_1 and A_1' are associated with two values or more values. Therefore, one relation is not possible.

⇒ Check for 2 relation is possible or not:

A_1	A_2
a_1	a_1'
a_2	a_1'
a_3	a_2'
a_4	a_3'

A_1	A_1'	B_1
a_1	a_2	b_1
a_1	a_2	b_1
a_2	a_3	b_2
a_4	a_3	b_3

Hence, minimum 2 relations are required.

NOTE: For self-referential relationships (any case) + total participation, always require 1 relation (minimum) (even in the case of many-to-many) but it have a lot of redundancy.

IV Solved Problems: (PYQs)

(2.4)

(Q1) In an Entity-Relationship (ER) model, suppose R is a many-to-one relationship from entity set E_1 to entity set E_2. Assume that E_1 and E_2 participate totally in R and that the cardinality of E_1 is greater than the cardinality of E_2. Which one of the following is true about R?

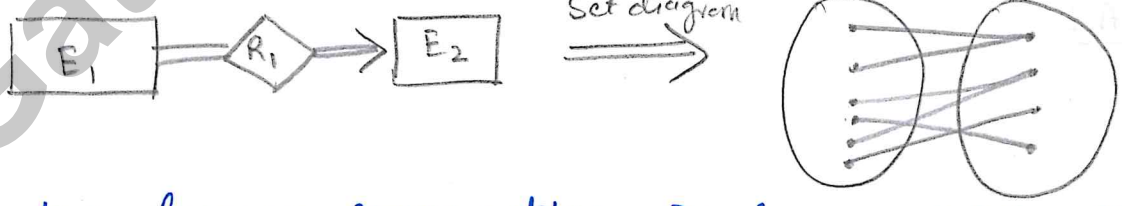
(A) Every entity in E_1 is associated with exactly one entity in E_2

(B) Some entity in E_1 is associated with more than one entity in E_2.

(C) Every entity in E_2 is associated with exactly one entity in E_1

(D) Every entity in E_2 is associated with at most one entity in E_1.

⇒ It is given that R_1 is many-to-one + total participation relationship from E_1 to E_2.



(A) It is true, because every entity in E_1 is associated with exactly one entity in E_2 (as we can see in set diagram)

While option (B), (C), and (D) are false because according (B) the relationship b/w E_1 and E_2 is many-to-many. According to (C) the relationship R_1 in b/w E_1 and E_2 is one-to-many and Option (D) is false as it is implying at most [illegible] with one-to-many b/w E_1 and E_2. Hence option (A) is True

(Q2) A E-R model of a database consists of entity types A and B. These are connected by a relationship R which does not have its own attribute. Under which of the following conditions, can the relational table for R be merged with that of A?

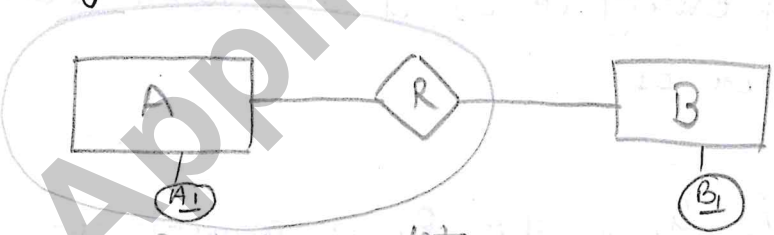
(A) Relation R is one-to-many and the participation of A in R is total.

(B) Relation R is one-to many and the participation of A in R is partial.

(C) Relation R is many-to-one and the participation of A in R is total.

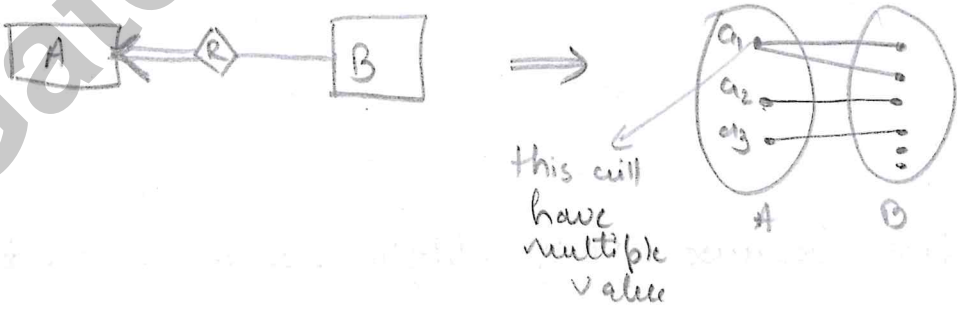
(D) Relation R is many-to-one and the participation of A in R is partial.

⇒



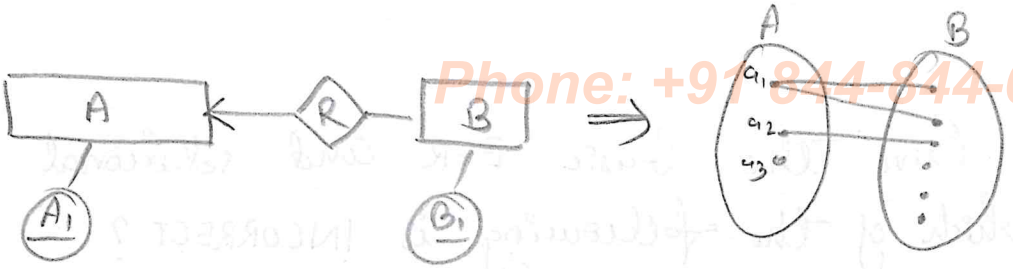
In which condition we can merge them?

Case A:



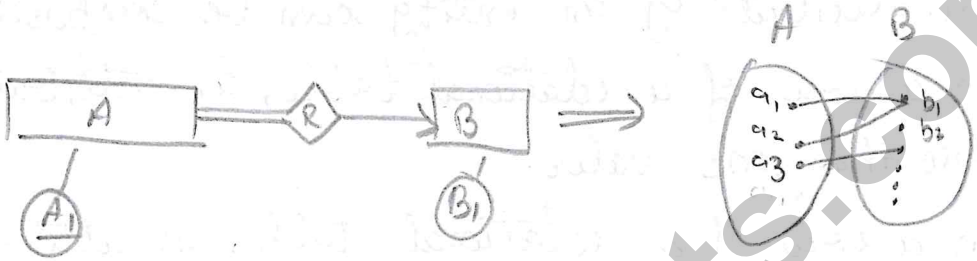
As we can see in entity A will have multiple values of 'a1'. Therefore, we cannot combine R with A in this condition. Hence, this is false.

Case B:



In case B also, there is the same problem. 'a1' will have multiple values. Hence, R could not be merge with A.

Case C:



Here, every value in A is associated with exactly 1 value in B. Therefore R could be merged A. Hence, this is true.

Case D:

Here, also ~~every~~ some value of A is associated with exactly one value in B while some are not even related. Hence, this is also true, But (C) is more appropriate.

Hence, answer is (C) and (D) both but (C) is more stronger option than (D) as it will not have any NULL values for R in relation. ~~Ans~~

(Q3) Given the basic E-R and relational models, which of the following is INCORRECT?

(A) An attribute of an entity can have more than one value

(B) An attribute of an entity can be composite.

(C) In a row of a relational table, an attribute can have more than one value.

(D) In a row of a relational table, an attribute can have exactly one value or a NULL.

⟹

(A) It is correct, it is talking about multivalued attributes, ie an attribute of a entity can have more than one value and these attributes are called multivalued attributes.

(B) It is correct, a entity can have composite/compound attributes. example 'name' is an attribute in 'customer' entity which is formed using 'first_name', 'last_name'.

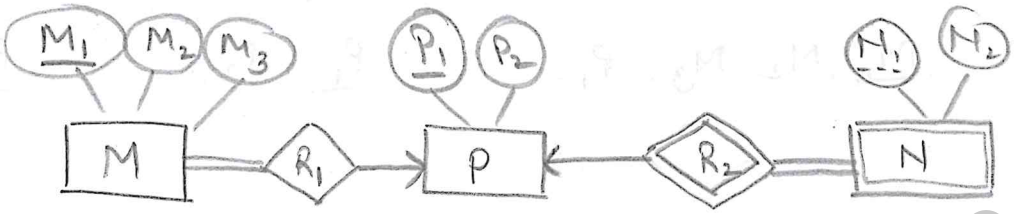
(C) It is incorrect, a tuple/row of a table cannot store more than ~~one~~ one value in a cell.

(D) It is correct, it is an opposite statement of option (C)

Hence, option (C) is incorrect.

(Q4) The minimum number of tables needed to represent M, N, P, R_1, R_2 is:

(A) 2
(B) 3
(C) 4
(D) 5

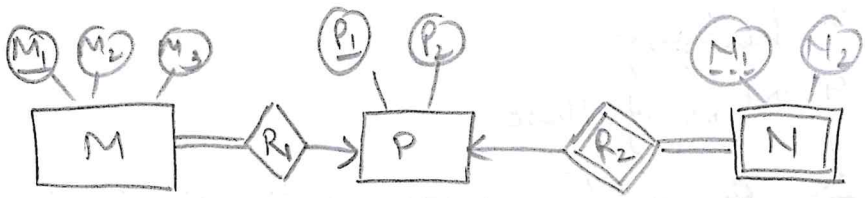


⇒ If we recall, it is same/similar to Fig: 21 problem.
- option (C) and (D) are the trivial option.
- We cannot do it in 2 relations because neither M and P could be combined together nor P and N could be combined.

Therefore, option (B) is correct.

(Q5) Consider the following E-R diagram. Which of the following is a correct attribute set for one of the tables for the minimum number of tables needed to represent M, N, P, R_1, R_2 ?

(A) M_1, M_2, M_3, P_1
(B) M_1, P_1, N_1, N_2
(C) M_1, P_1, N_1
(D) M_1, P_1.



⇒ Since, it is an extension of the (Q4), we know that minimum no. of relation required are 3 : M+R_1, P, R_2+N

So, the 3 relations formed are:

M_1	M_2	M_3	P_1

Table 1

P_1	P_2

Table 2

P_1	N_1	N_2

Table 3

Hence, option 1 is obviously correct as these are the attributes of table 1.

(86) Consider the entities "hotel-room" and "Person" with a many-to-many relationship "lodging" as shown below. If we wish to store information about rent payments to be made by person(s) ~~occuy~~ occupying different hotel rooms, then this information should appear as an attribute of

1. Person
2. Hotel room
3. lodging
4. None of these.

⇒ Rent payment will be paid only when a "person lodged in a hotel room". (Person on his own will not pay rent and hotel-room on ~~its~~ its own will not have rent) Only when a person stays in a hotel room than he is supposed to pay rent. Hence, there is no logic to keep attribute 'rent' in person or Hotel-Room entity. Therefore, it should be the attributes of lodging (2)

INTRODUCTION TO RELATIONAL MODELS

I. (3.1) Mathematical model of Tables:

Relational models or Relational databases are nothing but tables and these tables are the mathematical relations implicitly.

Relational database or models uses the same terminology That is:-

* Relations are also called tables.
* Columns are also called attributes. The set of values a column can take is called domain
* Rows are also called tuples.

~ Overview of relation in mathematics ~

In mathematics, a relation is defined as a subset of two sets $A \times B$

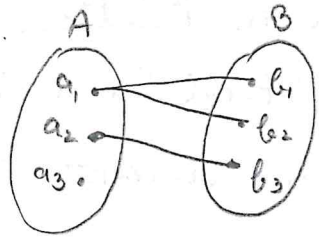
ie. Given set $A = \{a_1, a_2, a_3 \cdots a_n\}$

$B = \{b_1, b_2, b_3 \cdots b_n\}$

$R \subseteq A \times B$

where $A \times B = \{(a_i, b_i) \mid a_i \in A, b_i \in B\}$

example: $R: A \rightarrow B$ and $R \subseteq A \times B$

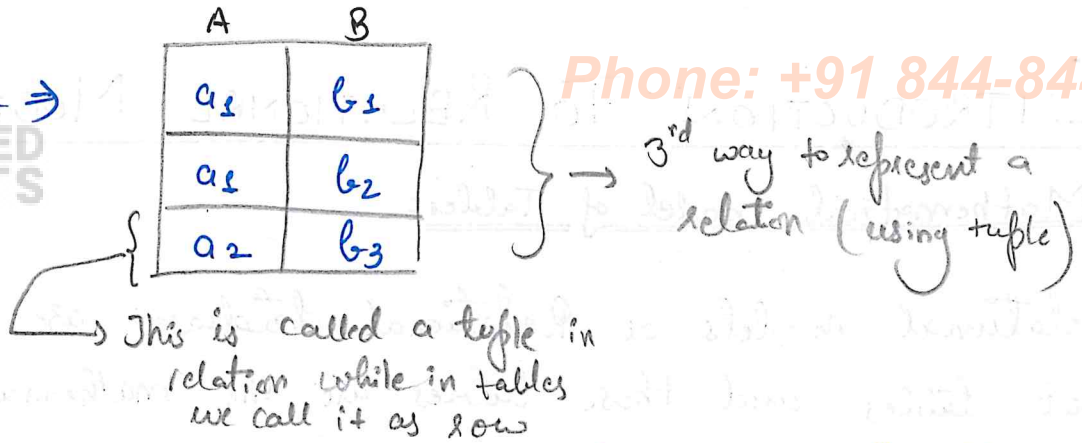


→ 1st way to represent a relation

So, R could be any subset of $A \times B$, ie.

$R \subseteq A \times B \Rightarrow R = \{(a_1, b_1), (a_1, b_2), (a_2, b_3)\}$

↓ 2nd way to represent a relation



here, domain of the attribute A is nothing but the set A which we have defined ie $A = \{a_1, a_2, a_3 \ldots a_n\}$ Similarly, for attribute B.

Hence, DBMS relation models and tables are related to the concept of mathematical relation. and all the properties of mathematical sets and relation are applicable to dbms relations / tables also. like

→ Sets cannot have duplicate values and same with the database models. As we have already studied that in relational model is no two tuples could be exactly same. (Concept of primary key)

→ In mathematics relation $R \subseteq A \times B$ ie $R = \{(a_1, b_1), (a_1, \{b_1, b_2\})\}$. This relations is neither allowed in mathematical relations nor in relations of dbms. A tuple cannot have multivalues stored in a cell.

Here $\{b_1, b_2\}$ is not an element of B they are subset of B.

Hence, we can now prove that why there are such rules in dbms relational model / E-R diagrams as at its core they are the mathematical relations / tables.

Note:- In relational database, the outline of database relationships and structure is given by relational schema. Relational schema is written as: <relation name> (<set of attributes>) + integrity constraint

Ex:- R_1 (Cid, cname, cgjp, cadds) + integrity constraints

II. (3.3) Solved Problems:

(Q1) Consider the following tables T_1 and T_2
In table T_1, P is the primary key and Q is the foreign key referencing R in table T_2 with on-delete cascade and on-update cascade. In table T_2, R is the primary key and S is the foreign key referencing P in table T_1 with on-delete set NULL and on-update cascade. In order to delete record (3, 8) from table T_1, the number of additional records that need to be deleted from table T_1 is _____

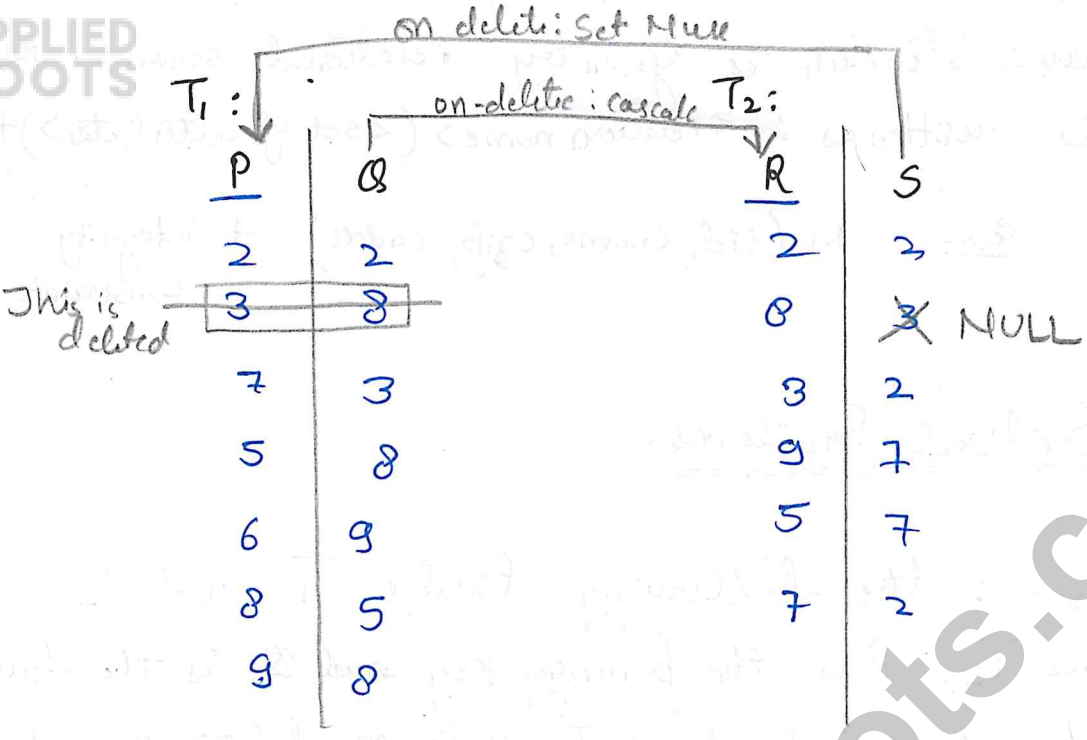
T_1:

P	Q
2	2
3	8
7	3
5	8
6	9
8	5
9	8

T_2:

R	S
2	2
8	3
3	2
9	7
5	7
7	2

⇒ The relation T_1 and T_2 are related as follows



P	Q
2	2
~~3~~	~~8~~
7	3
5	8
6	9
8	5
9	8

R	S
2	2
8	~~3~~ NULL
3	2
9	7
5	7
7	2

On deleting (3,8) in T_2, only ~~tomg~~ change which will occur is in ~~the~~ table T_2. The cell having attribute S value as 3 will become NULL.

Hence, no tuples are deleted on deleting (3,8)

Therefore, Answer is 0 (zero)

(Note if the question how many tuples got modified then answer = 1)

(Q 2) The following table has two attributes A and C where A is the primary key and C is the foreign key referencing A with on-delete cascade.

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

The set of all tuples that must be additionally deleted to preserve referential integrity when the tuple (2,4) is deleted is:

(A) (3,4) and (6,4)

(B) (5,2) and (7,2)

(C) (5,2), (7,2) and (9,5)

(D) (3,4), (4,3) and (6,4)

⇒

	A	C	
①	~~2~~	~~4~~	This is deleted initially.
	3	4	
	4	3	
②	~~5~~	~~2~~	This will get deleted beg ① got deleted
③	~~7~~	~~2~~	
④	~~9~~	~~5~~	This will get deleted because of ② got deleted
	6	4	

Tuple ~~①~~, ②, ③, ④ ie (~~5~~, ~~2~~), (7,2) and (9,5) will get deleted on deleting (2,4)

Hence, correct option is (C)

(Q 3) Let R(a, b, c) and S(d, e, f) be two relations in which d is the foreign key of S that refers to the primary key of R. Consider the following four operations on R and S

I. Insert into R

II. Insert into S

III. Delete from R

IV. Delete from S

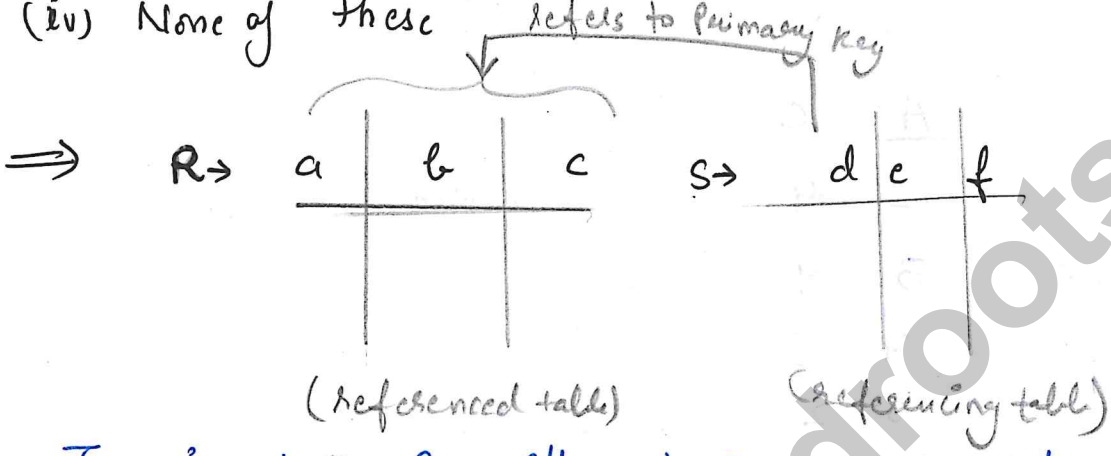
Which of the following can cause violation of the referential integrity constraint above?

(i) Both I and IV

(ii) Both II and III

(iii) All of these

(iv) None of these



I. insert in R will not cause any referential integrity problem

II. insertion in S, required to check whether inserted value of d is valid or not. Therefore, it may cause referential integrity constraint

III. Deletion from R required tuple to either cascaded deleted or set to Null. so, this may cause referential integrity problem.

IV. Deletion from S will not cause any problem.

Hence, I and III may cause referential integrity constraint violation. So, correct option is (ii)

(84) The maximum number of superkeys for the relation schema R(E, F, G, H) with E as the key is ____

⇒

E ∪ ϕ
F
G
:
FGH

∵ There are 3 elements left i.e. subsets possible = 2^3 = 8

∴ max number of superkeys possible = 8

(85) Given the STUDENTS relation as shown below. For (StudentName, StudentAge) to be the key for this instance, the value X should not be equal to ____

Student id	StudentName	StudentEmail	StudentAge	CPI
2345	Shakar	Shankar@math	X	9.4
1287	Swati	Swati@cc	19	9.5
7853	Shankar	Shankar@csc	19	9.4
9876	Swati	swati@mech	18	9.3
8765	Ganesh	ganesh@civil	19	8.7

⇒ Since, we have two entries in the name of 'Shankar' ∴ {StudentName ~~and~~ StudentAge} is key. So in order to avoid ~~these~~ that duplication and to uniquely determine a row/tuple, X should not be 19.

(Q6) Which of the following is NOT a superkey in a relational schema with attributes V, W, X, Y, Z and primary key VY?

1. VXYZ
2. VWXZ
3. VWXY
4. VWXYZ

⟹ A superkey is primary ~~key~~ or candidate key ∪ other attribute

1. It is a union of VY ∪ XZ ∴ It is a superkey
2. It is not a ~~super~~ superkey because it doesn't include the primary key.
3. It is a union of VY ∪ WX ∴ it is a superkey
4. It is a union of VY ∪ WXYZ ∴ it is a superkey

Hence, correct option is (2)

(Q7) Consider a relational table with a single record for each registered student with the following attributes:

1. Registration_Num: Unique registration number of each registered student.
2. UID: Unique identity number, unique at the national ~~level~~ level for each citizen.
3. Bank_Account_Num: Unique account number at bank. A student can have multiple accounts or joint accounts. This attributes stores the primary account number.
4. Name: Name of the student
5. Hostel_Room: Room number of the hostel.

Which one of the following option is INCORRECT?

(A) Bank_Account_Num is a candidate key

(B) Registration_Num can be primary key

(C) UID is candidate key if all students are from the same Country

(D) If S is a superkey such that S ∩ UID is NULL then S ∪ UID is also a superkey.

⟹ (A) Candidate Key should uniquely determine a tuple. A student can have multiple bank account or joint accounts so, there is possibility that two students S_1 and S_2 are having joint account and they will share the Bank_Account_Num. Hence, this is incorrect.

(B) It is correct because Registration_Num is unique for each student.

(C) It is correct as UID can uniquely determine a student from the same country

(D) Suppose we have superkey say S = {Registration_Num, Bank_Account_Num}. It is able to uniquely ~~ide~~ identify a tuple. UID is one of the attribute which is not included in the superkey as it is given in the option (ie S ∩ UID = NULL) So, even if we include UID in the superkey S, than also it will be uniquely identify each tuple and is ~~a~~ possible superkey. Hence, it is correct.

Therefore, Correct option is (A)

RELATIONAL ALGEBRA

I. Introduction to Relational Algebra and Basic Operators:

(4.1)

We are going to study 3 types of query languages (tools to obtain data from a DBMS):

[1] Relational Algebra
[2] Relational Calculus } → These are more towards mathematical view

[3] SQL } → This is more towards programmatic view.

→ Procedural vs Declarative languages:-

PROCEDURAL	DECLARITIVE
✳ Clear instructions on "how to generate outputs for given outputs" are given	✳ It tells "what we want" and donot tell anything regarding "how to get/obtain it".
✳ Example: Relational Algebra, C, C++, Java. But relational algebra is not a programming language, its a mathematical language.	✳ Example: SQL

→ Basic Relational Operators: (These operators are called relational operators because they operate on relations)

✳ Projection:-

It is represented as 'Π' (Pi)

Customer :- Relation

Cid	Cname	Czip
i_1	n_1	z_1
i_2	n_1	z_2
i_3	n_2	z_2

fig: 22

Projection operation is a relational operator which takes only one relation as an input (it is a unary operator) and outputs the attributes / columns listed in the form of a relation.

Syntax: $\Pi_{<attribute\_name>}(<Relation\_name>)$

Example 1: $\Pi_{cid, cname}(Customer)$ =

↓ Π has taken a relation as i/p

cid	Cname
i_1	n_1
i_2	n_1
i_3	n_2

↓ output's another relation with only mentioned fields.

Example 2: $\Pi_{cname}(Customer)$ =

Cname
n_1
n_1
n_2

✗ Wrong output.

Since, a relation is a set of tuples, therefore, it should not have duplicates. Hence, the output of example 2 query is:

Cname
n_1
n_2

Selection :-

It is denoted by symbol 'σ' (sigma)

Selection operation is a relational operator which selects a subset of tuples satisfying the given condition.

Syntax: $\sigma_{<condition>}(<relation\_name>)$

↳ it can be any Propositional logic statement

Example 1: On considering the relation given in Fig: 22 the output of following query is

$\sigma_{Cname = n_1}(customer) =$

↑ Takes 1 relation as i/p

Cid	Cname	Czip
i_1	n_1	z_1
i_2	n_1	z_2

↑ o/p's another relation.

Example 2:

$\sigma_{(Cname = n_1) \wedge (Czip = z_1)}(customer) =$

Cid	Cname	Czip
i_1	n_1	z_1

NOTE- We can combine projection and selection operation together.

Example: $\Pi_{Cid}\left(\sigma_{Cname=n_1}(customer)\right)$

This will get executed first

o/p of $\sigma_{Cname=n_1}(customer)$ is ⇒

Cid	Cname	Czip
i_1	n_1	z_1
i_2	n_1	z_2

(Lets name it as R_1)

Now $\Pi_{Cid}(R_1)$ =

Cid
i_1
i_2

final o/p

Rename :-

It is represented by 'ρ' (rho). It is used to rename the output relation.

Syntax: $\rho_{<new\_name>|<old\_name>}(<relation\_name>)$

Example: Considering the relation given in fig: 22, the output of the following query is

$\rho_{(cust\,name/cname}(customer)$ =

Cid	Cust name	Czip
i_1	n_1	z_1
i_2	n_1	z_2
i_3	n_2	z_2

↓

It will only rename the attribute name. Rest of the relation will remain

Set Operations : $(\cup, \cap, -)$

C_1	C_2	C_3
1	a	α
2	b	β
3	c	γ

C_1	C_2	C_3
1	a	α
4	d	δ
2	c	γ

Fig: 23

All set operations are applicable to relations also because relations are the set of tuples.

NOTE- Set operation - Complement is not often used in relations. This is not often used because

R_1^C = 

where $U = C_1 \times C_2 \times C_3$ (this condition is not often.)

In Fig: 21 $R_1^C = U - R_1$

(i) Union : (Combines all unique tuples of both R_1 and R_2 into 1 relation)

Considering relations given in fig: 23

$R_1 \cup R_2$ =

C_1	C_2	C_3
1	a	α
2	b	β
3	c	γ
4	d	δ
2	c	γ

Note (1, a, α) won't repeat because relation is a set

(ii) Intersection: (combines all the common tuples in between 2 relations into a ~~single~~ new relation)

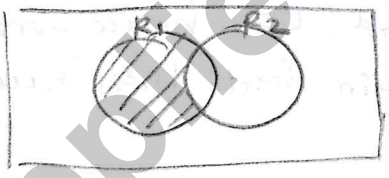
On considering relations given in Fig: 2.3

$R_1 \cap R_2$ =

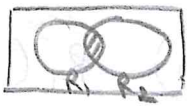
C_1	C_2	C_3
1	a	α

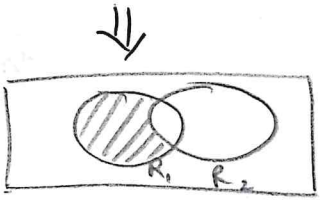
NOTE- Intersection is possible only when all the attributes values matches. In above example there is only one tuple whose all the three attributes are same.

(iii) Difference: (outputs all the tuples which are in relation R_1 but not in relation R_2)



$R_1 - R_2 = R_1 - (R_1 \cap R_2) =$ $R_1 -$





On considering the relations given in Fig: 2.3 the output will be:

$R_1 - R_2 = R_1 - (R_1 \cap R_2)$ =

C_1	C_2	C_3
2	b	β
3	c	γ

NOTE: Compatibility of relations:

We cannot perform set theoretic operations if # of attributes in R_1 and R_2 are not equal or if they are having different domain.

Ex-

1) $R_1(C_1, C_2, C_3)$, $R_2(C_1, C_2)$ ⟶ NOT COMPATIBLE

2) $R_1(C_1, C_2, C_3)$, $R_2(C_1, C_2, C_4)$ ⟶ NOT COMPATIBLE

↳ Datatype is diff

here, C_3 and C_4 have different domain.

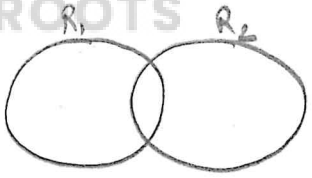
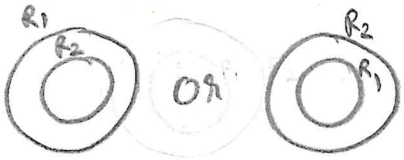
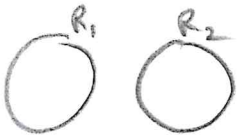
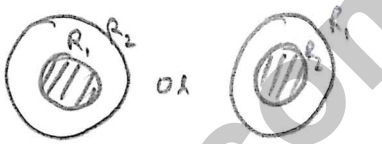
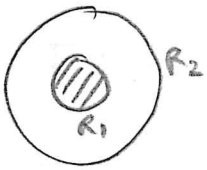
3) $R_1(C_1, C_2, C_3)$, $R_2(C_1, C_2, C_5)$ ⟶ NOT COMPATIBLE

C_3 and C_5 have same domain but they have different name

4) $R_1(C_1, C_2, C_3)$, $\rho_{C_3/C_5}(R_2(C_1, C_2, C_5))$ ⟶ COMPATIBLE

here, C_3 and C_5 have same domain and we have renamed the C_5 attribute to C_3.

NOTE: Given 2 relations R_1 and R_2 having m and n tuples respectively. Suppose that R_1 and R_2 are compatible and hence, we can perform set operations on them. Then, the minimum and maximum number of tuples possible on performing set operation are:

SET OPERATION	MINIMUM TUPLE	MAXIMUM TUPLE
 $R_1 \cup R_2$ (m tuple) (n tuple)	 max(m, n) tuples	 m+n tuples
$R_1 \cap R_2$	 0 tuples	 min(m,n) tuples
$R_1 - R_2$ $(R_1 - R_2 \neq R_2 - R_1)$	 0 tuples	 m tuples

Cross-product / Cartesian product:

C_1	C_2	C_3
1	a	x
2	b	y

C_3	C_4
α	β
1	α

Fig: 24

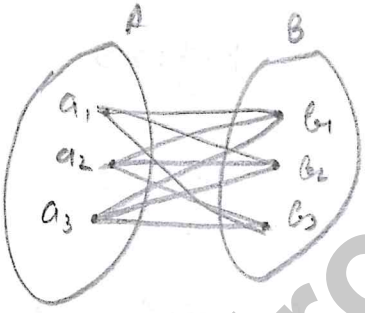
Cartesian product is helpful to merge columns from two relations. It is given as $R_1 \times R_2$.

$R_1 \times R_2$ =

C_1	C_2	C_3	C_3	C_4
1	a	x	α	β
1	a	x	1	α
2	b	y	α	β
2	b	y	1	α

2×4 = tuples

If R_1 has m tuples and R_2 has n tuples, then $R_1 \times R_2$ will have $m * n$ tuples. It can be understood as:



NOTE:

→ Suppose a relation R_1 with m attributes and R_2 with n attributes

$$R_1(r_1, r_2 \cdots r_m) \times R_2(s_1, s_2 \cdots s_n)$$

($r_1 \cdots r_m$: attributes)

$$\Downarrow$$

$$= \{(\underbrace{r_1, r_2, \cdots r_m, s_1, s_2, \cdots s_n}_{m+n \text{ attributes}}) \mid (r_1, r_2 \cdots r_m) \in R_1 \text{ AND } (s_1, s_2 \cdots s_n) \in R_2\}$$

→ We can do cross product of any relation with any number of relations.

Ex: $R_1 \times R_1$, $R_1 \times R_2 \times R_3$, $R_1 \times R_1 \times R_1$ etc all are valid cartesian product

→ Note that, duplicate rows are not allowed but duplicate columns are allowed.

Derived Operators

These operators are derived using basic operators.

Ex Natural join, division etc.

C_1	C_2	C_3
1	a	α
2	b	β

R

C_2	C_3	C_4
a	α	1
a	γ	2

S

Fig: 25

(i) Natural join:

It is represented using '$\bowtie$'. It is a binary operator which takes 2 relations as i/p and outputs a new relation. It is similar to cross product but with a twist. The twist is, there should be atleast one column which is common in between the two relations in order to join them. (atleast column should be compatible ie having same domain)

On considering relation given in fig: 25

$R \bowtie S =$

C_1	C_2	C_3	C_4
1	a	α	1

Here only one set of (C_2, C_3) in R got matched with set of (C_2, C_3) in S Hence, we have only one tuple in resultant relation.

It could be written as:

$$R \bowtie S = \Pi_{C_1, C_2, C_3, C_4}\left(\sigma_{R.C_2 = S.C_2 \wedge R.C_3 = S.C_3}(R \times S)\right)$$

(it is expanded expression of natural join using basic operators)

- Π_{C_1, C_2, C_3, C_4}: This will select a bunch of columns.
- $\sigma_{R.C_2 = S.C_2 \wedge R.C_3 = S.C_3}$: This will do filtering of tuples obtained in cartesian product
- $(R \times S)$: This will give us 6 columns ie $C_1, C_2, C_3, C_2, C_3, C_4$. So, we will use $R.C_2$ and $R.C_3$ in order to avoid confusion.

(ii) Conditional join:

It is a slight modification of natural join. It is also referred as "theta join". It is same as natural join with a condition associated with it. It is written as '$\bowtie_\theta$' (θ-join) or '$\bowtie_c$' (c stands for condition)

$$R \bowtie_c S = \sigma_c(R \times S)$$

For given relation R and S in fig: 23,

$$R \underset{R.C_1 < S.C_4}{\bowtie} S = \sigma_{R.C_1 < S.C_4}(R \times S)$$

with these condition

$R.C_1$	$R.C_2$	$R.C_3$	$S.C_2$	$S.C_3$	$S.C_4$
1	a	α	a	γ	2

Note: Conditional join do not have projection operation associated with them. This is one of the difference between Conditional join and theta join. Another difference btw them is

The conditional join can have any condition which could be defined using operators $<, >, \leq, \geq, =$ etc. While natural join can have only one condition which is '=' and that too in between common attributes or fields of two relations R and S.

These are the fundamental difference between natural join and conditional join.

(iii) Left outer join :

It is denoted as '⟕'. This operation allows us to keep all tuples in the left relation. If there is no matching tuple found in right relation, then the attributes of right relation in the join result are filled with NULL values.

It could be written as

$R ⟕ S = R ⋈ S \cup$ all tuples from the left relation that failed the join with NULL values appended

On considering relation R and S in fig 25

$R ⟕ S =$

C_1	C_2	C_3	C_4	
1	a	α	1	→ matched tuple
2	b	β	NULL	→ unmatched tuple with appended NULL

Fig: 26

NOTE - Left outer join could be both conditional and non-conditional. If it is conditional, then condition will be applied at selection part of Natural join.

It may look like:

$R \overset{\subset}{\bowtie}_c S = R \bowtie_c S \cup$ {all tuples from the left relation that failed join with NULL values appended.}

NOTE- Using basic operators, it could be written as:

$$R \overset{\subset}{\bowtie} S = R \bowtie S \cup \left(R - \Pi_{r_1, r_2 \ldots r_m}(R \bowtie S)\right) \times (\text{NULL}, \text{NULL}, \ldots)$$

No. of NULL's = No. of additional column for R.

Where, $R \circledcirc (r_1, r_2 \ldots r_m)$
$S (s_1, s_2, \ldots s_n)$

Here, in our example there is only one additional column ie 'C' so we will do cross product of only one NULL in the given case.

(iv) Right outer join:-

It is denoted as '$\overset{\subset}{\bowtie}$'. This operation allows us to keep all tuples in the right relation. If there is no matching tuple found in left relation, then the attributes of left relation in the join result are filled with NULL values.

It could be written as:

$R \overset{\subset}{\bowtie} S = R \bowtie S \cup$ all tuples from the right relation that failed the join with NULL values appended

Everything else is same as that of left-outer join.

Using basic operators it could be ~~left~~ written as

$$R \overline{\bowtie} S = R \bowtie S \cup (NULL, NULL \ldots) \times (S - \Pi_{S_1, S_2 \ldots S_n}(R \bowtie S))$$

↓

NULL is equal to no. of additional column for S

On considering relation R and S given in fig: 25

$R \overline{\bowtie} S =$

C_1	C_2	C_3	C_4
1	a	α	1
NULL	a	γ	2

Fig: 27

(V) Fullouter join :

It is represented as '$\overline{\bowtie}$'. It could be given as

$$R \overline{\bowtie} S = R \bowtie S \cup R \overline{\bowtie} S$$

$=$

C_1	C_2	C_3	C_4
1	a	α	1
2	b	β	NULL
NULL	a	γ	2

} it is union of all tuples obtained in $R \overline{\bowtie} S$ and $R \bowtie S$

(vi) Division:

Cust_id	Category
1	C_1
1	C_2
2	C_1
3	C_2
4	C_1
4	C_2

R

Category
C_1
C_2

S

Fig: 28

Suppose we are asked a query/question that – (from fig:28)

"Customers who purchased from every category"

→ According to the relations given in fig: 28, there are only two categories available ~~and~~ (C_1 and C_2) and there are only 2 customers who purchased product from both of them. Hence, the output will be

Cid
1
4

T

Note that we are able to arrive to this solution with the help of division operation ($R \div S$) also written as 'R/S'.

$R \div S$ or R/S ⇒ It outputs those tuples from R, for which it holds that all their combinations with tuples in S are present in R. Here, only 1 and 4 have their

Combination with all tuples in S (and are present in R)

$\therefore R \div S = T$

now, if we do $T \times S$ =

Cust_id	category
1	C_1
1	C_2
4	C_1
4	C_2

$\longrightarrow$ As we can see here, $R \supseteq T \times S$

We can write division operator using basic ~~attribut~~ operators as:

$R(cust\_id, category) \div S(category) =$

$$\Pi_{cust\_id}(R) - \Pi_{cust\_id}\Big(\big(\Pi_{cust\_id}(R) \times S\big) - R\Big)$$

① ② ③ ④

Output of ①:

Cust_id	Category
1	C_1
1	C_2
2	C_1
2	C_2
3	C_1
3	C_2
4	C_1
4	C_2

Output of ②:

cust_id	category
2	C_1
3	C_2

Output of ③:

cust_id
2
3

Output of ④:

Cust id
1
2
3
4

custid
2
3

$=$

custid
1
4

NOTE- Completeness :- We can represent any relational operator using $\{\cup, -, \times, \Pi, \sigma\}$ in relational algebra. This is a very popular complete set of operators in relational algebra.

II Solved Problems:- (4.3)

(Q1) Consider the following relation $P(X, Y, Z)$, $Q(X, Y, T)$ and $R(Y, V)$:

How many tuples will be returned by the following relational algebra query?

$$\Pi_X(\sigma_{(P.Y = R.Y \wedge R.V = V_2)}(P \times R)) - \Pi_X(\sigma_{(Q.Y - R.Y \wedge Q.T > 2)}(Q \times R))$$

X	Y	Z
x_1	y_1	z_1
x_1	y_1	z_2
x_2	y_2	z_2
x_2	y_4	z_4

P

X	Y	T
x_2	y_1	2
x_1	y_2	5
x_1	y_1	6
x_3	y_3	1

Q

Y	V
y_1	v_1
y_3	v_2
y_2	v_3
y_2	v_2

R

$\Rightarrow$ The query given is:

$$\underbrace{\Pi_x(\sigma_{(P.Y=R.Y \wedge R.V=V_2)}(P \times R))}_{①} - \underbrace{\Pi_x(\sigma_{(Q.Y=R.Y \wedge Q.T>2)}(Q \times R))}_{②}$$

① will return only one tuple because of the 2 conditions $P.Y = R.Y$ and $R.V = V_2$

hence, output will be:

X
x_2

② will return also return only one tuple because of two conditions $Q.Y = R.Y$ and $Q.T > 2$

hence, the output will be:

X
x_1

final we need to find ① − ② = ① − (① ∩ ②)

∴ ① ∩ ② = NULL

∴ ① − ② = ① . Hence only 1 tuple will be returned.

(Q2) Consider the relation $r(A, B)$ and $S(B, C)$, where $S.B$ is a primary key and $r.B$ is a foreign key referencing $S.B$.

Consider the query:

$$Q : r \bowtie (\sigma_{B<5}(S))$$

Let LOJ denote a natural left outer join operation. Assume that r and s contain no NULL values. Which one of the following is NOT equivalent to Q?

(1) $\sigma_{B<5}(r \bowtie S)$

(2) $\sigma_{B\leq 5}(r \text{ LOJ } S)$

(3) $r \text{ LOJ } (\sigma_{B<5}(S))$

(4) $\sigma_{B<5}(r) \text{ LOJ } S$

⇒ We are given that:

A	B
α	4
β	5

r

B	C
4	a
5	b

S

Query given is: $r \bowtie (\sigma_{B<5}(S)) \Rightarrow Q \Rightarrow$

A	B	C
α	4	a

Let us consider an example here (mentioned above)

(1) the output of 1st query will be:

A	B	C
α	4	a

(2) the output of 2nd query will be:

A	B	C
α	4	a

(3) the output of 3rd query will be:

A	B	C
α	4	α
β	5	NULL

Hence, (3) is the answer ($(3) \neq (8)$)

(Q 3) Consider a database that has the relation schema CR(StudentName, CourseName). An instance of the schema CR is as given below:

The following query is made on the database:

$T_1 \leftarrow \Pi_{CourseName}(\sigma_{StudentName = 'SA'}(CR))$

$T_2 \leftarrow CR \div T_1$

The number of rows in T_2 is _____

StudentName	CourseName
SA	CA
SA	CB
SA	CC
SB	CB
SB	CC
SC	CA
SC	CB
SC	CC
SD	CA
SD	CB
SD	CC
SD	CD

SE	CD
SE	CA
SE	CB
SF	CA
SF	CB
SF	CC

CR

$\Rightarrow$ Here T_1 is $\Rightarrow$ $\Pi_{\text{CourseName}}(\sigma_{\text{StudentName}='SA'}(CR))$

This will o/p all tuples with name = 'SA'

it will project Course name of all Student having name as 'SA'

=

CourseName
CA
CB
CC

T_2 is $\Rightarrow$ $CR \div T_1$ } (o/p) look at all StudentName which are combined with {CA, CB, CC}

=

StudentName
SA
SC
SD
SF

} all these 4 are combined with {CA, CB, CC}

Hence No. of tuples = 4 //

(Q4) Consider two relations $R_1(A, B)$ with tuples $(1, 5), (3, 7)$ and $R_2(A, C) = (1, 7), (4, 9)$. Assume that $R(A, B, C)$ is the full natural outer join of R_1 and R_2. Consider the following tuples of the form (A, B, C): $a = (1, 5, \text{NULL})$, $b = (1, \text{NULL}, 7)$, $c = (3, \text{NULL}, 9)$, $d = (4, 7, \text{NULL})$, $e = (1, 5, 7)$, $f = (3, 7, \text{NULL})$, $g = (4, \text{NULL}, 9)$. Which one of the following statements is correct?

(1) R contains a, b, e, f, g but not c, d
(2) R contains all a, b, c, d, e, f, g
(3) R contains e, f, g but not a, b
(4) R contains e but not f, g.

$\Rightarrow$ here $R_1 \Rightarrow$

A	B
1	5
3	7

$R_2 \Rightarrow$

A	C
1	7
4	9

$\therefore R = R_1 \bowtie R_2$

$=$

A	B	C
1	5	7
3	7	NULL
4	NULL	9

} actual tuples in R.

As we can see in the option only option (3) and (4) are having 3 and 1 tuple respectively. Therefore, these 2 are

potential options.

As we can observe $e = (1, 5, 7)$

$f = (3, 7, \text{NULL})$

$g = (4, \text{NULL}, 9)$

Since, option (3) includes all of these 3 tuples hence correct option is (3).

(Q 5) Suppose (A, B) and (C, D) are two relation Schemas. Let r_1 and r_2 be the corresponding relational instances. B is a foreign key that refers to C in r_2. If data in r_1 and r_2 satisfy referential integrity constraints. Which of the following is ALWAYS TRUE?

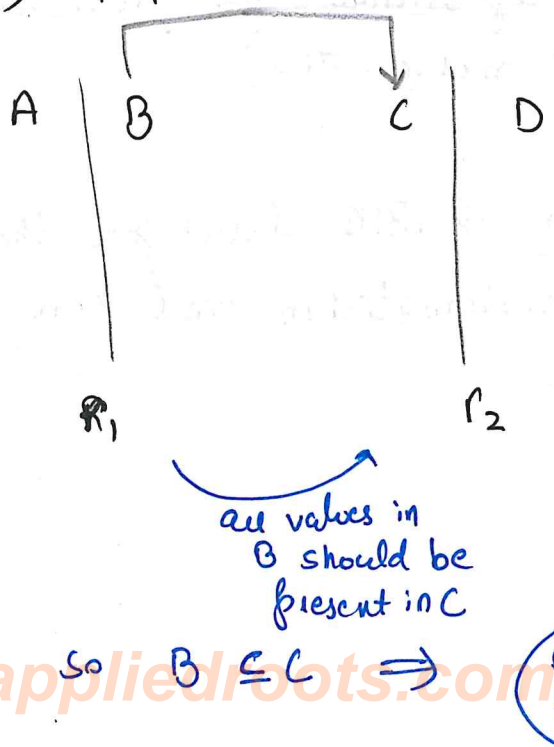
(A) $\Pi_B(r_1) - \Pi_C(r_2) = \phi$

(B) $\Pi_C(r_2) - \Pi_B(r_1) = \phi$

(C) $\Pi_B(r_1) = \Pi_C(r_2)$

(D) $\Pi_B(r_1) - \Pi_C(r_2) \neq \phi$

$\Rightarrow$ Given:



According to the options we can clearly say option (A) is appropriate answer as it will be always true. $(B - C = \phi)$

5. TUPLE RELATIONAL CALCULUS

I (5.1) Tuple Relational calculus – I :-

* TRC is a declarative language. Here we only mention "what we want and not how to get it"
* TRC is a mathematical foundation for SQL.
* TRC is heavily based on propositional and predicate logic.
  (Note: Propositional and predicate logic is covered in discrete mathematics)
* Edger Codd @ IBM developed the TRC, RA, SQL. He is a mathematician and computer scientist.

A query in TRC looks like:

$$\{ t \mid P(t) \}$$

- t: it refers to a tuple
- $P(t)$: it is a predicate logic expression involving operation like $\{\vee, \wedge, \neg, \exists, \forall, \in\}$

Cid	Cname	Czip
i_1	n_1	z_1
i_2	n_1	z_2
i_3	n_2	z_3
i_4	n_3	z_1

Customer

Fig: 29

Let us consider this example relation to understand TRC with the help of RA

Select rows from customer where zip code = z_1

In RA $\Rightarrow$ $\sigma_{Czip = z_1}(\text{Customer})$ (it is an example of Selection operation)

Both will give same o/p

In TRC $\Rightarrow$ $\{ t \mid t \in \text{customer} \wedge t.Czip = z_1 \}$

↳ Predicate logic expression

it will be read as: "Return me the set of tuples t, such that t belongs to customer table and $t.Czip = z_1$"

NOTE:

There are many ways of writting TRC query. For example, in place of writting "$t.Czip$" we can also write "$t[Czip]$". Both are equivalent. In place of writting "$t \in$ customer"

we can also write it as "Customer(t)". So, there are multiple notations of writing TRC.

Project cid from customer relation

In RA ⇒ $\Pi_{cid}(Customer)$

⇕

In TRC ⇒ $\{t \mid \exists S (S \in Customer \wedge S.cid = t.cid)\}$ → more popular notation

it will be read as: "Set of all tuples such that there exists S, such that S belongs to customer table and S.cid = t.cid"}

This seems to be very round about way to write as we are creating a new variable S.

Another way we can write this in TRC is

$\{t.id \mid t \in Customer\}$ → This is one of the way to write but it is not a popular way.

NOTE-

as far as propositional logic is concerned it will be equivalent to P(t.cid)

Let's break $\{t \mid (\exists S)(S \in Customer \wedge S.cid = t.cid)\}$

- $\exists S$: There exist S
- $S \in Customer$: S belong to customer means S is a tuple now. "it is a tuple" it has all 3 columns
- $S.cid = t.cid$: we want only cid that is why we use t.cid only. "only cid attribute is used here" it will return only cid attribute as o/p

NOTE- In previous example of projection, variable t is not associated with any ∃ and ∀ while S is associated with ∃ and ∀, therefore, variable S is called Bounded variable.

Bounded variable are those which are constrained or has either ∃ or ∀ attached to it.

While t is unbounded or free variable. t is not bounded by any first order logic or predicate logic condition (∃ and ∀)

✳ Project cid and cname from customer

In RA ⇒ $\Pi_{cid, cname}(customer)$

In TRC ⇒ $\{t \mid \exists s \in customer (s.cid = t.cid \wedge s.cname = t.cname)\}$

it invokes P(t.cid, t.cname)
∴ it will print only t.cid and t.cname and not t.czip

✳ Project cid for all czip = z, on customer (Selection + Projection)

In RA ⇒ $\Pi_{cid}(\sigma_{czip=z_1}(customer))$

In TRC ⇒ $\{t \mid \exists s \in customer (s.czip = z_1 \wedge s.cid = t.cid)\}$

P(t.cid)

cid

Selection operation

Projection operation

It will be read as: "Select set of all tuples t such that there exist s such that belongs to customer and s.czip = z, and s.cid = t.cid"

NOTE Same like RA, in TRC also we can write complete set of operations ie $\{\cup, -, \pi, \sigma, \times\}$ and hence, using these basic operators we can write all derived operators like join, divisions etc in TRC.

Cartesian product

Given relations:

{ Customer(cid, cname, zip)
Product(cid, pid, time) }

In RA ⇒ Customer × Product (it will o/p 6 attributes)

In TRC ⇒ $\{t \mid \exists t_1 \in \text{Customer}, \exists t_2 \in \text{Product} (t.cid = t_1.cid \wedge t.cname = t_1.cname \wedge t.zip = t_1.zip \wedge t.cid_2 = t_2.cid \wedge t.pid = t_2.pid \wedge t.time = t_2.time)\}$

(t_1 is row of Customer; t_2 is row of Product; every row of t_1 is combined with every t_2 row.)

this will have all 6 attributes

It will be read as: "set of all tuples t such that t_1 belongs to C and t_2 belongs to P (C for customer and P for product) such that {set of all conditions}"

Set operations

(1) Union

If two relations R and S are set operation compatible than

In RA ⇒ $R \cup S$

In TRC ⇒ $\{t \mid t \in R \vee t \in S\}$

OR

$\{t \mid R(t) \vee S(t)\}$

(ii) Set difference :

In RA → $R - S$

In TRC ⇒ $\{t \mid t \in R \wedge \forall t_1 \in S (t \neq t_1)\}$

($t \neq t_1$: t should not be equal to any tuple in (ie $t \notin S$))

(it can also be written as $\neg(t = t_1)$)

It will be read as "for all tuples set t, where t belongs to R AND for all t_1 belongs to S, i want t to be not equal to t_1"

→ Find customer id of customers who made atleast one purchase.

In RA ⇒ $\Pi_{cid}(\sigma_{c.cid = p.cid}(Customer \times Product))$

(Projection; selection; cross product)

In TRC ⇒ $\{t \mid \exists t_1 \in Customer, \exists t_2 \in Product(t_1.cid = t_2.cid \wedge t.cid = t_1.cid)\}$

(t will return t.cid; $P(t.cid)$)

It will be read as "a set of tuples t, there exists t_1 belongs to customer and t_2 belongs to product such that $t_1.cid$ should be equal to $t_2.cid$ and $t.cid = t_1.cid$"

NOTE-

formula / equations in predicate logic

1) $\neg(\forall x\, P(x)) = \exists x(\neg P(x))$

Similarly, $\neg(\exists x P(x)) = \forall x(\neg P(x))$

} De morgan's laws in Predicate logic

2) $P \rightarrow Q \equiv \neg P \vee Q$

3) $\forall x(P(x) \wedge Q(x)) = \forall x P(x) \wedge \forall x Q(x)$

4) $\forall x (P(x) \rightarrow Q(x)) = \forall x (\neg P(x) \vee Q(x))$

II (5.2) Safe queries of Domain Relational calculus:

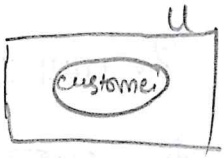
→ Safe vis unsafe Queries:

Let's take an example to understand this:

$\{t \mid \neg(t \in \text{customer})\}$ → It means, I want all tuples that do not belongs to customer.

This means t ∉ customer
OR
$\{t \mid t \notin \text{customer}\}$

This is a valid statement in predicate logic. But if we talk about this in relational calculus, it is nothing but compliment of relation customer.



In set theory compliment is given as:

$\text{Customer}^c = U - \text{customer}$ (U stands for compliment)
= infinite result.

So, the TRC statement written above is a valid statement but the resultant set is a non-finite set of tuples. (It is like every human being possible which is not a customer of Amazon). Therefore, it is an unsafe query.

Hence, any TRC query which results in a non-finite set of tuples is called a unsafe query.

NOTE - Unsafe TRC queries could not be written (represent) in Relational algebra. (In relational algebra it is already discussed that there is no concept of complement). for every safe TRC query there is an equivalent Relational algebra query.
The expressive power of Relational algebra is same as that of safe TRC. Unsafe TRC is a valid TRC statement query but there result is non-finite.

NOTE -
There is a popular trick to determine whether given TRC is safe or unsafe, If there is a complement of free variable than most likely it is going to be unsafe query.

$\{t \mid \neg(t \in \text{customer})\}$
↳ free variable

→ Domain Relational calculus (DRC):

RA, TRC and DRC are mathematical foundation or model. These are not used while writing a query in real world. It is used to optimise the query and not to work with them.

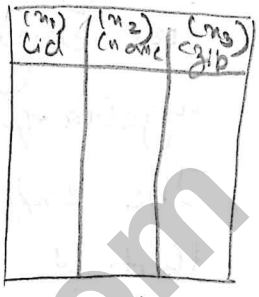
→ Selection: (For relation given in Fig: 2.9)
In RA ⇒ $\{t \mid t \mid t \mid t\ \sigma_{c3:ip=3,}(\text{customer})$
In TRC ⇒ $\{t \mid t \in \text{Customer} \wedge t.c3ip = 3,\}$
↳ tuple variable

In DRC ⇒ $\{ \langle x_1, x_2, x_3 \rangle \mid \langle x_1, x_2, x_3 \rangle \in \text{customer} \wedge x_3 = 3 \}$

(x_1, x_2, x_3: Domain variable)

x_1 can take all values that belongs to the domain cid.

x_2 can take all values that belongs to the domain cname. Similarly for x_3



customer

$\langle x_1, x_2, x_3 \rangle$ is an ordered triplet

In TRC we take whole tuple / row as variable while here every attribute is an variable

NOTE -

Syntax of DRC:

$\{ \langle x_1, x_2 \cdots x_n \rangle \mid P(x_1, x_2, \cdots x_n) \}$

($x_1, x_2 \cdots x_n$: domain variables; $P(x_1, x_2, \cdots x_n)$: Predicate logic on these domain variables.)

So only difference between DRC and TRC is that in TRC we have ~~tuple~~ variable for a Tuple while in DRC we have variables for all attributes.

* Projection:

In RA ⇒ $\Pi_{cid}(\text{customer})$

In DRC ⇒ $\{ \langle x_1 \rangle \mid \exists x_2, x_3 \ (\langle x_1, x_2, x_3 \rangle \in \text{customer}) \}$

($\langle x_1 \rangle$: this will select only cid as x_1 is for cid)

In TRC ⇒ $\{ t \mid \exists s \ (s \in \text{customer} \wedge s.cid = t.cid) \}$

Cartesian product:

Given Customer (cid, cname, city)
Product (cid, pid, time)

In RA ⇒ ~~Cross product~~ Customer × Product

In DRC ⇒ $\{ < x_1, x_2, x_3, x_4, x_5, x_6 > \mid \exists (x_1, x_2, x_3) \in \text{Customer} \wedge \exists (x_4, x_5, x_6) \in \text{Product} \}$

III (5.3) Solved problems:

(S1) Which of the following relational query languages have the same expressive power?

I. Relational Algebra
II. TRC restricted to safe expressions
III. DRC restricted to safe expressions

(A) II and III only
(B) I and II only
(C) I and III only
(D) I, II and III

⇒ As we know RA is equivalent to TRC and DRC (condition that TRC and DRC are restricted to safe expression)

Hence, all the 3 statements are equivalent.

Therefore, correct option is (D)

(Q2) Which of the following relational calculus expressions is not safe ?

(a) $\{t \mid \exists u \in R_1 (t[A] = u[A]) \wedge \neg \exists s \in R_2 (t[A] = s[A])\}$

(b) $\{t \mid \forall u \in R_1 (u[A] = \text{"x"} \Rightarrow \exists s \in R_2 (t[A] = s[A] \wedge s[A] = u[A]))\}$

(c) $\{t \mid \neg (t \in R_1)\}$

(d) $\{t \mid \exists u \in R_1 (t[A] = u[A]) \wedge \exists s \in R_2 (t[A] = s[A])\}$

⇒ While evaluating all the query we can see (a), (b) and (d) are very complex and we can spend our time to evaluate them but since we can see option (c) is definetly a unsafe query (as it is giving the complement as output). Therefore, we can ~~evale~~ say the (c) is the correct option.

(Q3) Consider the relation Employee (name, sex, SupervisorName) with name as the key. SupervisorName gives the name of the supervisor of the employee under consideration. What does the following tuple Relational calculus query produce ?

$$\{e.\text{name} \mid \text{employee}(e) \wedge (\forall x)[\neg \text{employee}(x) \vee x.\text{supervisorName} \neq e.\text{name} \vee x.\text{sex} = \text{"male"}]\}$$

(A) Names of employees with a male supervisor
(B) Names of employees with no immediate male subordinates
(C) Names of employees with no immediate female subordinates
(D) Names of employees with a female supervisor.

⇒ Given query

{ e.name | employee(e) ∧

(∀x) [¬ employee(x) ∨ x.superviserName ≠ e.name
∨ x.sex = "male"]}

(it will return names → e ∈ employee)

We will try to convert this ~~negation by~~ "for all" expression to "there exists" because we are not comfortable with the negations.

∴ ∀x {P(x)} = ¬¬∀x P(x) = ¬(∃x ¬P(x))

So it could be written as:

{ e.name | employee(e) ∧

¬(∃x) (employee(x) ∧ x.superviserName = e.name ∧
x.sex = "female"]}

This whole expression is now negated.

So, it will be read as: "Find employee name, ~~such that~~ there does not exist x, such that x is an employee and x is supervisor and is female". In short we want ename such that

(Q4) Which of the following tuple relational calculus is/are equivalent to:

$$\forall t \in r(P(t))\ ?$$

I. $\neg \exists t \in r(P(t))$

II. $\exists t \notin r(P(t))$

III. ~~$\equiv$~~ $\neg \exists t \in r(\neg P(t))$

IV $\exists t \in r(\neg P(t))$

(A) I only

(B) II only

(C) III only

(D) III and IV only

⇒ Here:

$r \rightarrow$ relation

$P(t) \rightarrow$ Predicate logic

It could also be written as

$$\forall t \in r(P(t)) = \underbrace{\forall t\,(t \in r(P(t)))}_{\text{more appropriate}}$$

$$= \neg(\neg \forall t(t \in r(P(t))))$$

$$= \underbrace{\neg \exists t\ (t \notin r(P(t)))}_{\text{This is not in option}}$$

⇓

This could be written as

$$t \notin r(P(t)) = \underbrace{t \in r(\neg P(t))}_{\Downarrow}$$

$$\neg \exists t \in r(\neg P(t)) \Rightarrow \text{III}$$

Hence, correct option is (C) //

7 STANDARD QUERY LANGUAGE (SQL)

* SQL stands for standard Query language
* SQL is a standard way to query / obtain / add / delete / modify data
* It is used by data scientist, machine learning engineer, S/w engineer etc
* SQL is not a general purpose programming language. It is a domain specific language whose primary task is to build data base. Other languages like C / C++ / Java / Python are general purpose programming language and therefore use to build web servers, operating system, while SQL is not used for such tasks.
* SQL is called declarative programming language where we do not specify step-by-step procedure to get / obtain data.

  ex- SELECT movie_name from movies WHERE year > 200

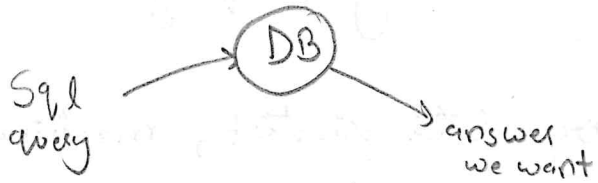
  here, we are declaring "what we want?" and not, "how to get it?"

  Because it is declarative, that is why it becomes simple ~~for the~~ to use.

* SQL is very powerful language.

7.3 Execution of SQL statement

Given the fact that SQL is an declarative programming language, "how does an SQL statement is executed?"



So, internally it works as follows:

Sql → Parser/compiler → Query optimizer

Parser/compiler
- → tries to understand the query
- → Breaks the query into Parts
- → Compiles it into Procedural Prog. language ie generate code (C/C++/python)
- → If throws errors, if found any

Query optimizer
- → given this query it gives optimal way to execute the query (Best DS, Best seq of exe.)

↓

Query executor
- → it executes the optimized code/query on DB

↓

Data base

↓

Results

Fig:30 Simplified execution cycle of an SQL query

7.4 IMDB Dataset :

* IMDB is a website owned by Amazon. This dataset will be considered as an set example to understand different SQL commands. This dataset is a movie dataset.

* This data set have 388,269 movies from 1888-2008 which have 817,718 actors and 86,880 directors.

→ Structure of the IMDB dataset (Relational database):

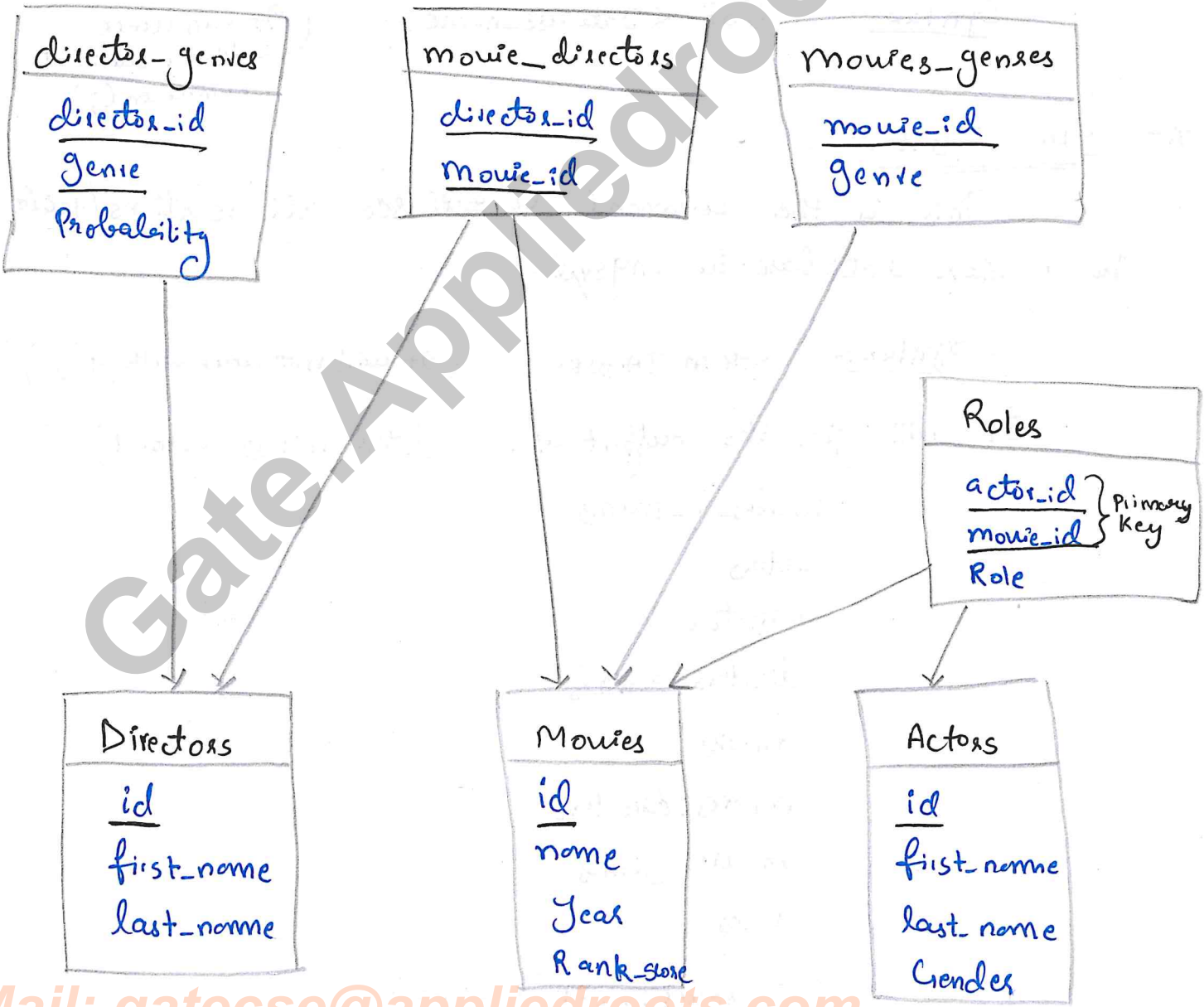


Fig: 31 IMDB schema

0th SQL COMMANDS

0.1 Use, Describe, Show-tables:

✳- USE:

It is used to tell mysql that which database we want use or work on.

Syntax- USE <Database_name>; (It can work with or without semicolon (;))

✳- SHOW TABLES:

This is the command to ~~see~~ see all relations/tables in a given database in mysql.

Syntax:- SHOW TABLES; (It will not work without ';')

It will give the output as: (for IMDB dataset)

Tables_in_IMDB
actors
directors
directors_genres
movies
movies_directors
movies_genres
roles

7 rows in set (0.00 sec)

DESCRIBE:

To understand each of this table, we use a command DESCRIBE. It will give complete info about a relation like, data type, default values, keys etc.

Syntax - DESCRIBE <Table_name>; (will not work without semicolon (;))

Example - DESCRIBE actors;

output ⇒

Field	Type	Null	Key	default	Extra
id	int(11)	NO	PRI	0	
first_name	Varchar(100)	YES	MUL	NULL	
last_name	Varchar(100)	YES	MUL	NULL	
Gender	Char(1)	YES		NULL	

Data type

4 rows in set (0.00 sec)

→ it means more than 1 actor could have same first_name or last_name.
ex- both salman and shaurukh have last name 'Khan'

2. Select:

Suppose we are given with a task: "list all movies in the database".

Given a query "SELECT * FROM movies;"

Keywords

- SELECT → it says select all columns mentioned.
- * → list of columns. Here * means everything
- FROM → from is a keywords which mention the table_name from which data
- movies → table_name

It will be read as "Select everything (*) from the table name movies". So, as soon as we run it, it will give complete table content as the result.

NOTE- Describe movies will only result in the description of every attribute present in table movies. While, select basically says which column to select and will give all values / content present under that column.

Example 2: "SELECT name, year FROM movies;"

→ it will give 388,209 row as o/p with only two attributes "name and years". from table movies.

NOTE- Here we are not explaining how to generate output, we only mention what we want.

NOTE- The output generated by "Select" in SQL query is called "result set". It is nothing but another table that it creates. (It is a set of rows with Column name)

NOTE- Example 2 is faster than example 1 (list all movies) bcoz amount of data to be transferred is less in example 2 than in example 1 (∵ it is selecting *, ie everything / every column from table movies).

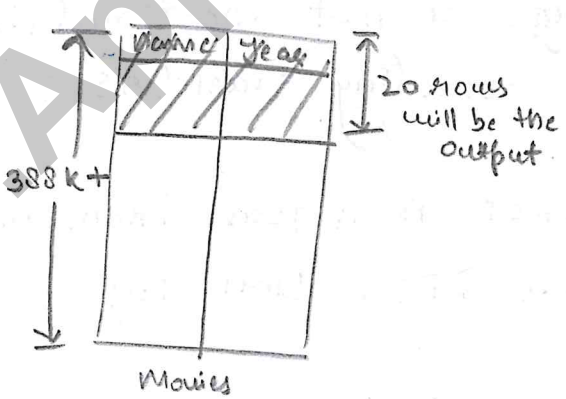
So, it is suggested to query only those attributes which are required rather than using *.

NOTE: The order of column name could be anything. It need not to be in the same order as it is given in the table/relation. The o/p or result set will have the order of the attributes as given in the query.
Note that, the order of rows in result set will be same as order in the original table/relation. This is called row-order preservation (So, row orders are preserved by simple select queries, there are some queries where row order is not preserved)

8.3 Limit, Offset:

Limit is a keyword which tells the number of rows we need to limit in our result set.

Example 1: SELECT name, year FROM movies LIMIT 20;
↳ # of rows



it will give only first 20 rows as outputs.

Here, we got the first 20 rows only. What if we want to get the next 20 rows now. For that we use an another keyword called OFFSET.
When we add offset <value> it means "ignore the starting rows equal to offset value and print the next rows equal to limit value."

Example 2: SELECT name, year FROM movies LIMIT 20, OFFSET 20;

o/p ⟹ it will ignore first 20 rows and will print next 20 rows of table movies with columns name and year.

So, offset basically says number of rows from start to ignore. If offset is equal to 40, it will ignore first 40 rows of table movies.
Limit and offset makes the data more readable as they provide us subset of data.

8.4 Order by:

This keyword is used to get the ordered output for a given ~~column~~ column name.
If the order type is not mentioned, it by default order the data in ascending order (ASC)

Example 1: SELECT name, year FROM movies ORDER BY year DESC LIMIT 10;

- year → The result is ordered based on year
- DESC → The result is in descending order.

The feature like "sort by" and "filter" on various websites run the sql command with ORDER BY at backend.

Hence, it helps us to sort the data and therefore the order of rows in result set will not be same as the original table rows order.

Q.5 Distinct:

This keyword is used to get unique/distinct values from a column/attribute.

Example- SELECT DISTINCT genre FROM movies_genres;

keyword — column_name

It will o/p all distinct genres. The o/p set will have 21 rows for 21 different genres for the given IMDB dataset.

If we want to find distinct over multiple fields we can write it as:

Distinct will work/applied on both of these.

Example 2: SELECT DISTINCT first_name, last_name
FROM Directors ORDER BY first_name;

It will give 86844 rows as o/p, there ~~is one~~ are two row with first_name = "Zvonko" because there are two "Zvonko" with different last name ie 'Coh' and "Saksida" respectively.

So, when distinct is applied over multiple columns then complete row should be distinct (one column value can repeat but should have different value in other columns)

8.6 Where, Comparison operators, NULL:

WHERE:

This keyword is used to filter the result and apply conditions.

Example 1: SELECT name, year, rankscore FROM movies WHERE rank_score > 9;

condition

it says generate only those rows which have rank_score > 9

and finally for the rows which have rank_score > 9 we will select their name, year and rank_score.
It will output 1069 rows which are rated > 9.

Example 2: SELECT name, year, rank_score FROM movies WHERE rank_score > 9 ~~and~~ ORDER BY rank_score DESC LIMIT 20;

→ It will give top 20 movies, where rank_score > 9.
name, year, rank_score

NOTE: Whatever comes after WHERE clause / command is called 'condition'.

So, a condition's output could be True, False and NULL. and it may have comparison operators like =, <>/!= (not equal to), <, <=, >, >=

Example 3: SELECT * FROM movies_genres WHERE genre = "Comedy";

⇒ It will select all columns where genre is equal to comedy. It will o/p 56425 rows.

Example 4: SELECT * FROM movies_genres WHERE genre <> "Horror";

OR

SELECT * FROM movies_genres WHERE genre != "Horror";

⇒ It will select all columns ~~where~~ of all genres except "horror".

✳ NULL:

NULL in SQL means that: Value is unknown / missing / doesnot exist

NULL is a special keyword in SQL. for any attribute value as NULL means that value doesnot exists.

NOTE The operator " =" doesnot work with NULL

Example 1: SELECT name, year, rankscore ~~to~~ FROM movies WHERE rankscore = NULL;

⇒ O/p will be an empty set. We can't compare NULL with "equal to" symbol.

In order to make it work, we write the query using "IS" and "IS NOT"

Example 2- SELECT name, year, rankscore FROM movies WHERE rank_score IS NULL; LIMIT 20;

↓

it will work to give all rows where rank_score is NULL

- It will o/p 20 rows with rank_score as NULL

Example 3- SELECT name, year, rank_score FROM movies WHERE rankscore IS NOT NULL LIMIT 20;

Will o/p all rows where rankscore is not having NULL value.

8.7 logical operators:

AND, OR, NOT, ALL, ANY, BETWEEN, EXISTS, IN, LIKE, SOME are some of the logical operators in SQL. logical operators are required whenever we need to put multiple filters.

AND

Example 1: SELECT name, year, rank_score FROM movies WHERE rank_score > 9 AND Year > 2000;

condition 1 — condition 2

These 2 conditions are combined using AND operator.

⇒ It will list all movies that are released after year 2000 and having rank_score > 9. It will o/p 250 rows.

NOT

Example 2: SELECT name, year, rank_score FROM movies WHERE NOT year<=2000 LIMIT 20;

this condition is equivalent to year > 2000

⇒ It will list all movies that are released after year 2000 and limit the result set to the top 20 rows.

OR

Example 3: SELECT name, year, rankscore FROM movies WHERE rankscore > 9 OR year > 2007;

condition 1 | condition 2

Combined condition 1 and 2 and will result to TRUE if any of the condition gets true.

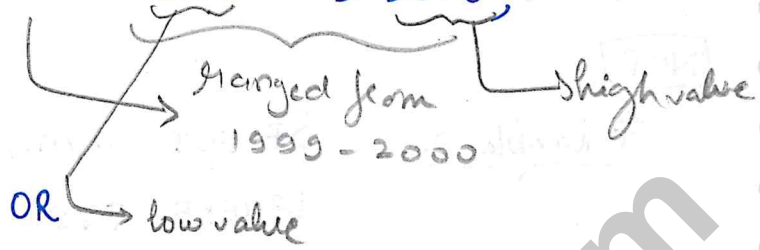
⇒ It will list all movies that are either released after 2007 ~~and~~ or having rank_score > 9.

NOTE- ALL and ANY are discussed in topic Subqueries.

BETWEEN

Example 4: SELECT name, year, rankscore FROM movies
WHERE year BETWEEN 1999 AND 2000;

(with between)



(without between) SELECT name, year, rankscore FROM movies.
WHERE year >= 1999 AND year <= 2000;

⇒ It will list all movies released in year between 1999 to 2000.

This is called inclusive range because both the values ie 1999 and 2000 are included in result.

Example 5: SELECT name, year, rankscore FROM movies
WHERE year BETWEEN 2000 AND 1999

⇒ This query will not work because low value is greater than high value

NOTE low value should be <= high value otherwise we will get an empty set as result.

IN

Example 6: SELECT director_id, genre FROM directors_genres
(with IN) WHERE genre IN ('Comedy', 'Horror');

OR

(without IN) SELECT director_id, genre FROM directors_genres
WHERE genre = 'Comedy' OR genre = 'Horror';

⇒ It will list all director_id and genre where genre is either Comedy or horror. So, IN clause allows us to test if expression matches any value present in the list or set of values.

LIKE (used for text matching)

Example 7: SELECT name, year, rank_score FROM movies
WHERE name LIKE 'Tis%';

(here % is the wildcard character to imply zero or more ~~than one~~ characters)

⇒ It will list all movies name, year, rankscore where names start with 'Tis' (ie 'Tis' is followed by anything or zero)

(In shell command 'grep' or regular expression in Toc works like this)

Example 8: SELECT first_name, last_name FROM actors WHERE first_name LIKE '%es';

⟹ This will o/p all first_name, last_name of actors whose first_name are ending with 'es'

Example 9: SELECT first_name, last_name, FROM ~~movies~~ actors WHERE firstname LIKE '%es%';

⟹ This will o/p all first_name, last_name of actors whose first_name contains substring 'es'

Example 10: SELECT first_name, last_name FROM actors WHERE first_name LIKE 'Agn_s';

('_' (underscore) is also a wildcard character which implies exactly one character)

⟹ This will o/p all first_name, last_name of actors whose first_name is like 'Agn_s'

↑ only 1 character can fix in to this.

NOTE

Consider the below relation:

T:

name	percentage
n1	59%
n2	63%
n3	96%

Query: SELECT * FROM T WHERE Percentage = '9%'

As we know % is a wildcard character which means zero or more ~~one~~ symbol.

While here % is not a wildcard character, it is a symbol. In such situations, in order to use '%' or '_' (underscore) as symbol and not as wild card character we use backslash (\) before the '%' or '_'

Example - SELECT * FROM T WHERE percentage = "9%\%"

This will be interpreted as symbol and not as wild card.

Note: backslash (\) : is called / interpreted as escape character in SQL.

8.8 Aggregate functions: COUNT, MIN, MAX, AVG, SUM:

These functions compute a single value on a set of rows and returns the aggregate value.

MIN

Example 1: SELECT MIN(year) FROM movies;

MIN(year): MIN = aggregate function; year = column name on which aggregate function is applied

⇒ It will return a single value which is the minimum year value in table movies.

MAX

Example 2: SELECT MAX(year) FROM movies;

⟹ It will return a single value which is the maximum value of year in column year in table movies.

SUM

Example 3: SELECT SUM(year) FROM movies;

⟹ It will return a single value which is sum of all values in column year.

Similarly, **avg** function will find the average of all values given in an column.

So, sum and AVG are the numerical operators while MIN and MAX are comparison operators.

COUNT(*)

Example 4: SELECT COUNT(*) FROM movies;

⟹ Count(*) will consider all columns and count the number of rows. It will not look for distinct rows, it will just count each and every row. Hence, output will be 388269 rows.

NOTE - Count(*) consider the rows containing NULL values also.

COUNT

Example 5: SELECT COUNT (year) FROM movies;

⇒ It will consider column year and will count every row without considering ~~do~~ of distinct rows.

It will output the same no. of rows as we got in example 4 ie 388269 rows.

NOTE- Except count(*), none of the aggregate function considers NULL values.

8.9 Group by:

This command is use to collect data across multiple records and group the results by one or more column.

Example 1: SELECT year, COUNT (year) FROM movies
GROUP BY year;

⇒ It will create a table with attribute year and count all the rows having release of that year. ie all the rows having the same year are grouped together. (Grouping of all rows having same year). Here count (year) will work on the group of year formed by GROUP BY clause. COUNT(year) actually gives no. of rows in that year group.

Example 2: SELECT year, COUNT(year) FROM movies
GROUP BY year ORDER BY year;

→ Everything will be same except result will be sorted in asc. order

Example 3: SELECT year, COUNT(year) year_count FROM movies GROUP BY year ORDER BY year_count;

(It is an alias / another name for count(year))

⇒ It will generate same o/p but the rows will be ordered in ascending order of count(year) or year_count (year_count is an alias). It will show the year as first row which has minimum releases (as it is ordered in ascending order) and last row of o/p will have maximum releases.

NOTE- Here we have used alias because we cannot use aggregate functions in Order by clause.

NOTE- Group_by are often used with COUNT, MIN, MAX or SUM.

NOTE- If grouping column contain NULL values, all NULL values are grouped together.

8.10 Having:

This command is often used in combination with the Group by clause to restrict the groups of returned rows to only those whose the condition is TRUE.

Example 1: SELECT year, COUNT(year) year_count FROM movies GROUP BY year HAVING year_count > 1000;

⇒ Here, first we are grouping and from that we will get year_count and now we will print only those year groups whose year_count > 1000 ie only those years which have > 1000

Here, order of execution will be :

Step 1: GROUP BY year to create groups

Step 2: Now apply COUNT(year) or any ~~aggregate~~ aggregate function

Step 3: finally, apply the HAVING condition.

NOTE- Typically HAVING is used in combination with GROUP BY, but it is not mandatory.

~~Eg:~~ example :- SELECT name, year FROM movies HAVING year > 2000;

⇕

SELECT name, year FROM movies WHERE year > 2000;

Hence, HAVING clause without GROUP BY is same as or works same as WHERE clause.

Example 2: SELECT year, COUNT(year) year_count FROM movies WHERE rank_score > 9 GROUP BY year HAVING year_count > 20;

(3) (1) (2) (4)

(1) → applied over individual rows of movies table

⇒ here, in this query we have both Where and having clause. The seq order of execution is numbered above

NOTE Where is applied on individual rows while Having is applied on top of groups. If we don't have group by, having will also be applied on individual rows.

~~Having is applied after grouping~~ while where is used before grouping.

⇒ This query will return year and COUNT(year) such that, that year has more than 20 movies and each of them have rank_score > 9.

8.11 Order of keywords:

While writting a sql query the order of keywords will be as follows:

[1] SELECT
[2] FROM
[3] WHERE
[4] GROUP BY
[5] HAVING
[6] ORDER BY (contains ASC and DESC after it)
[7] LIMIT

JOIN'S

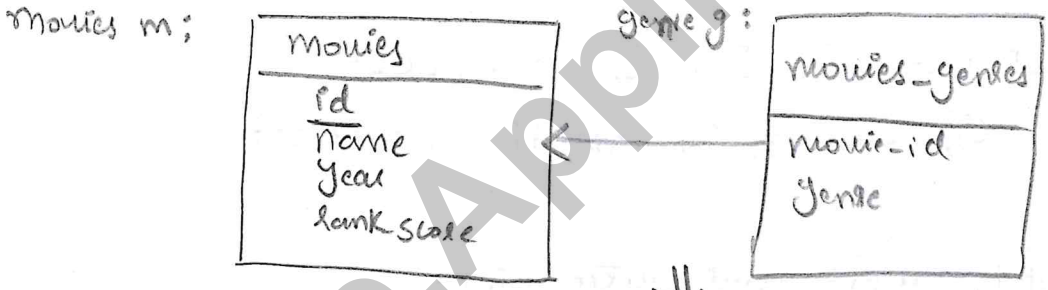
9.1 Join and Natural Join:

Join's are used to combine data in multiple tables/relations.

Example: SELECT m.name, g.genre FROM movies m
JOIN movies_genres g ON m.id = g.movie_id
LIMIT 20;

(alias of movies; alias of movie_genres; condition based on which m and g are joined)

On considering fig: 31, we can see that relation movies and movies_genres are given as:-



id	name	year	rank_score
1	n_1		
2	n_2		
3	n_3		

movie_id	genre
1	g_1
1	g_2
2	g_3
3	g_2

(A movie can have multiple genre)

⇒ It will print name and genre by combining table movies m and genre g where m.id = g.movie_id.

Natural join:

A join where we have the same column-names accross two tables.

example

T_1 :

C_1	C_2

T_2 :

C_1	C_3	C_4

Select * FROM T_1 JOIN T_2;

⇒ It will automatically joint T_1 and T_2 based on C_1. It is equivalent to saying:

SELECT * FROM T_1 Join T_2 ON $T_1.C_1 = T_2.C_1$;

We can also write it as:

SELECT * FROM T_1 JOIN T_2 USING (C1);

It will return C_1, C_2, C_3, C_4 in the result set.

9.2 Inner, left, right and Outer joins:

Consider the below relation set:

T_1 :

C_1	C_2	C_3
1	a	b
2	c	d
4	e	f
5	g	h

T_2 :

C_4	C_5	C_6
1	0	1
2	1	0
3	0	1
5	1	0

Inner join:

Example 1: SELECT * FROM T_1 JOIN T_2 ON $T_1.C_1 = T_2.C_4$

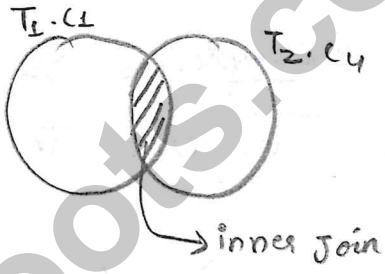
⇒ o/p will be

T ⇒

$C_1 = C_4$

C_1	C_2	C_3	C_4	C_5	C_6
1	a	b	1	0	1
2	c	d	2	1	0
5	g	h	5	1	0

Inner join diagramatically :-



Left outer join:

Example 2: SELECT * FROM T_1 LEFT OUTER JOIN T_2 ON $T_1.C_1 = T_2.C_4$

⇒ o/p will be

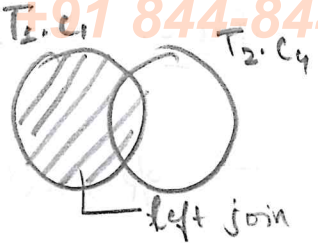
T ⇒

C_1	C_2	C_3	C_4	C_5	C_6
1	a	b	1	0	1
2	c	d	2	1	0
4	e	f	NULL	NULL	NULL
5	g	h	5	1	0

no match for this in right relation hence, we have added NULL for C_4, C_5, C_6

here, we are combining the left relation with right relation if there are matching rows and if no matching rows are found the left relation row is clubbed with NULL

Left outer join diagramatically ⇒



NOTE- We can either use keyword LEFT OUTER JOIN or LEFT JOIN, both are same.

* Right Outer join:

Example 3: SELECT * FROM T_1 RIGHT OUTER JOIN T_2 ON $T_1.C_1 = T_2.C_4$

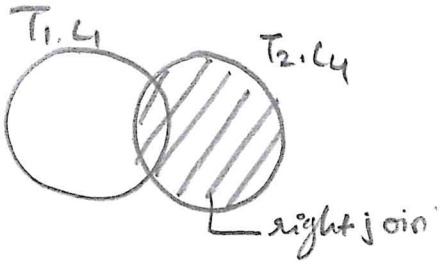
⟹ o/p will be:

T_3 =

C_1	C_2	C_3	C_4	C_5	C_6
1	a	b	1	0	1
2	c	d	2	1	0
5	g	h	5	1	0
Null	Null	Null	3	0	1

← In this row right relation do not have $T_1.C_1 = T_2.C_4$ hence, we have filled C_1, C_2, C_3 with NULL

Right outer join diagramatically ⇒



NOTE- We can either use keyword RIGHT OUTER JOIN or RIGHT JOIN, both are same.

Full outer join:

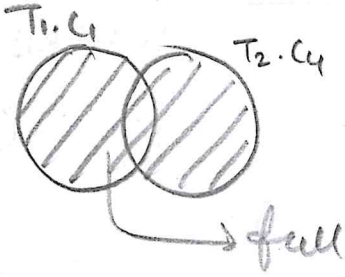
Example 4: SELECT * FROM T_1 FULL OUTER JOIN T_2 ON $T_1.C_1 = T_2.C_4$

$\Longrightarrow$ o/p will be:

Left join $\cup$ Right join

C_1	C_2	C_3	C_4	C_5	C_6
1	a	b	1	0	1
2	c	d	2	1	0
4	e	f	NULL	NULL	NULL
5	g	h	5	1	0
NULL	NULL	NULL	3	0	1

Full outer join diagramatically $\Rightarrow$



NOTE- We can use either FULL OUTER JOIN or FULL JOIN while writting a query, both are same.

NOTE- INNER JOIN $\equiv$ JOIN (Both are same)
(inner is optional keyword here)

NOTE- We can have 3-way join (Joining 3 relations) and K-way joins (Joining k relations)

Example- SELECT a.first_name, a.last_name FROM actors a JOIN roles r ON a.id = r.actor_id JOIN movies m on

This query is combining 3 relations ~~at~~ in a sql query

NOTE- Joins are the most expensive — computationally when we have large tables. It is one of the famous area of research that how to reduce its ~~time~~ computation time.

SUB QUERIES / NESTED QUERIES / INNER QUERIES

Syntax: SELECT <column_names> FROM <table_names>
WHERE <column_name> OPERATOR
(SELECT <column_name> FROM
<table_name> WHERE);

NOTE: OPERATORS here are:- IN, NOT IN, EXISTS, NOT EXISTS, ANY, ALL etc.

OR

```
SELECT  Column_name [, Column_name]
FROM  Table1 [, Table2]
WHERE  Column_name OPERATOR ( SELECT
        Column_name [, Column_name]
       FROM  Table1 [, Table2] [WHERE]);
```

IN

Examples: list all actors in the movies Schindler's list

⟹ Note that schindler's list is the whole series of movies.

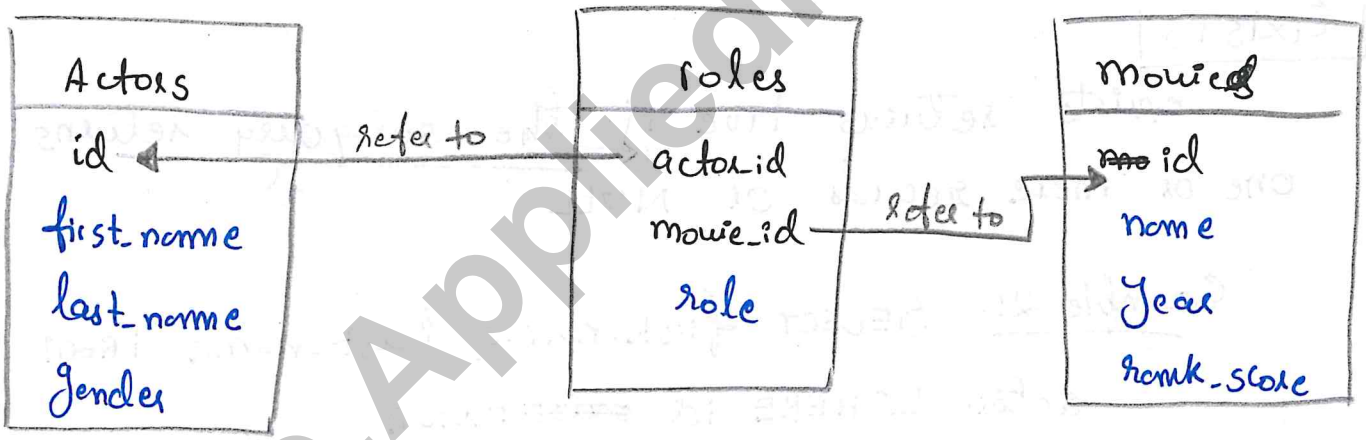
Q_1

```
SELECT first_name, last_name FROM actors WHERE
id IN ( SELECT actor_id FROM roles WHERE
movie_id IN ( SELECT id FROM movies WHERE
name = "Schindler's List") );
```

Q_2 Q_3 — nesting

Keyword to give a select value from a set or list

⟹ Here we are accessing 3 relations in a query using sub-queries. One way to do this is by writting it in the form of sub query while another way to solve this is using "Join's".
This could be understood as:



It will first execute the innermost query (Q_3) ie "SELECT id FROM movies WHERE name = "Schindler's List";"

⇒ It will result in the set of 126 tuples.

Next query which will get executed is Q_2 ie "SELECT actor_id FROM roles WHERE movie_id IN (126 tuples);"

⇒ It will return set of actor_id's.

At last, the outer most query (Q_1) got executed ie " SELECT first_name, last_name FROM actors WHERE id IN ({ set of all actor id's});

⇒ It will o/p all first_name, last_names whose ~~the set~~ actor id is in set o/p of Q_2.

NOTE In SQL 'IN' can be thought as belongs to($\in$) in set theory. While 'NOT IN' can be thought as do not belongs to($\notin$) in set theory.

EXISTS

Exists returns true if the subquery returns one or more records or NULL

Example 2: SELECT first_name, last_name FROM actors WHERE id ~~exist~~ EXISTS
(SELECT actor_id FROM Roles WHERE movie_id
EXISTS (SELECT id FROM movies WHERE
name = "Schindler's list"));

⇒ If the sub queries return a non-empty set ~~say {1}, {1}~~, having one or more than 1 element Or return a NULL than ~~what~~ we will get whatever we want in the result set of outer query.

NOTE Similarly it works for NOT EXISTS.

ALL

All operator returns TRUE if all of the subquery values meet the condition.

Example 3: SELECT * FROM movies WHERE rank_score >= ALL (SELECT MAX (rank_score) FROM movies);

↳ will return true if all subquery meet condition.

⟹ It will get us all movies whose rank_score is same as the maximum rank_score.

Here, subquery will return only one value which is max (rank_score) = {9.9}. Now outer query will print/select all columns ~~from~~ from table movies where rank_score >= {9.9}.

NOTE- Subqueries / inner queries are more easy to read and write when compared to Join's.

ANY

Any operator returns TRUE if any of the subquery ~~retu~~ values meets the condition.

* Correlated sub queries ⇒

In SQL, Correlated subquery / synchronized subquery is a subquery that uses values from the outer query. Because the subquery may be evaluated once for each row processed by the outer query, it can be efficient.

Example:

```
SELECT emp_no, name FROM Employees emp
WHERE salary > ( SELECT AVG (salary)
                FROM Employees
                WHERE department =
                emp.department );
```

(inner query)

⇒ Here, inner query will return :- it is going to compute average salary of employees and finally will print those emp_no and names of employees whose salary is greater than average salary of employees with in that department. (we know this fact of with in that department because we are using alias emp in our inner query).

ie. per department list all the employees whose salary's are greater than any salary in that department.

Note

It is called Correlated subquery because for every employee in outer query we need to run inner subquery again and again.

So, here we using info of outer query in the inner query and this inner query needs to be run for every employee.

NOTE- Correlated queries are computationally ~~Exfe~~ expensive.

DATA MANIPULATION LANGUAGE

11.1 Insert:

NOTE → SQL → Data manipulation language (DML)
SQL → Data Definetion language (DDL)
SQL → Data Control language (DCL)

DML includes SELECT, INSERT, UPDATE, DELETE. These four commands ~~are~~ belongs to data manipulation language.

INSERT is used to insert data / row / tuples in a relation.

Syntax- INSERT INTO <table_name> (< attribute_names>)
VALUES (< values_as_per_attribute_sequence);

Example:

```
INSERT INTO movies (id, name, year, rank_score)
VALUES
(412321, 'THOR', 2011, 7);      → 1st row
(412323, 'IRON MAN 2', 2010, 7); → 2nd row
```

inserting multiple rows in relation movies

NOTE- Since, 'id' is the primary key in table "movies" so, we cannot insert ~~two~~ rows with same 'id' value.

NOTE- We can also insert one table into another

Example

```
INSERT INTO <table_name1 (Phone_book_2)>
SELECT *
FROM <table_name2 (Phone_book)>
WHERE NAME IN ('John Doe',
               'Peter Doe')
```

⟹ It will copy all rows from Phone_book to phone_book_2 whose names are either 'John Doe' or 'Peter doe'.

11.2 Update, Delete:

Update:

Syntax: UPDATE <Table_name> SET Col1 = val1, Col2 = val2 WHERE <Condition>;

Example- UPDATE movies SET rank_score = 9 WHERE id = 412321;

⟹ It will update the rank_score value of the tuple whose id = 412321.

We can also update multiple column values by seperating them using commas or using subquery.

Delete:

Syntax: DELETE FROM <table_name> WHERE <Condition>;

Example:- DELETE FROM movies WHERE id = 412321;

⟹ It will delete the row which is having id = 412321

NOTE- If we want to delete all rows from table we use a command called TRUNCATE. It is not DML it is a DDL. Its syntax is: TRUNCATE TABLE <table_name>;

NOTE- We can also delete all rows from a table using 'DELETE' command. This could happen if we remove WHERE clause.

Syntax - DELETE FROM <table_name>;

DATA DEFINITION LANGUAGE (DDL)

12.1 Create Table:

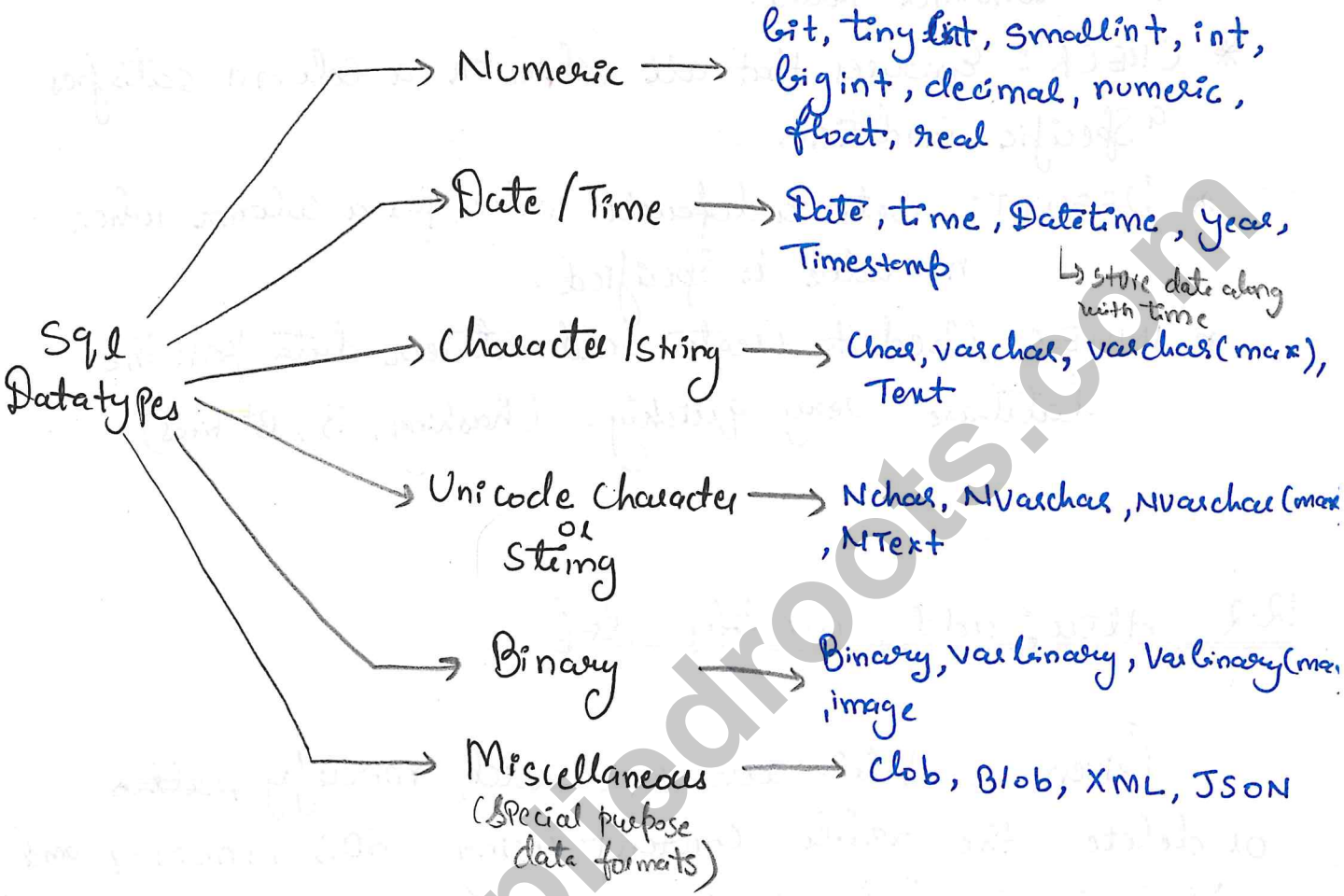
* DDL helps us to define, alter, drop a table.
* To create a new table in database is created as follows:

Syntax: CREATE TABLE <table_name> (
<attribute_name> <datatype> <Specification>);
or Properties or Constraint

Example CRETE TABLE language (
id INT PRIMARY → it is integer type
lang VARCHAR(50) NOT NULL); → it is a character type

⟹ If we try to insert any duplicate id or lang as NULL, it will throw an error.

→ SQL datatypes:



Note that, not every database support all datatypes, Some may support some datatypes while other may support other datatypes.

→ Constraints / Properties:

* NOT NULL: Ensures that a column cannot have a NULL value.
* UNIQUE: Ensures that all values in a column are different.
* PRIMARY KEY: A combination of NOT NULL and UNIQUE. It uniquely identifies a row/tuple in a table.

* FOREIGN KEY: Uniquely identifies a row/record in another table.
* CHECK: Ensures that all values in a column satisfies a specific condition.
* DEFAULT: Sets a default value for a column when no value is specified.
* INDEX: Used to create and retrieve data from the database very quickly. (Hashing, B, B+ trees)

12.2 Alter: add, modify, drop:

Given a table we can add, modify ~~, alter~~ or delete the table using ADD, MODIFY and DROP commands in DDL

→ ADD: It helps us to add new columns or constraints in DB

Syntax: ALTER TABLE <table_name>
ADD <column_name> <data_type>
<constraints>;

Example ALTER TABLE language ADD
Country VARCHAR(50);

→ MODIFY: It helps to modify column name, datatype or constraints.

Syntax: ALTER TABLE <table_name> MODIFY
<Column_name> <data_type> <Constraints>;

Example- ALTER TABLE language MODIFY country VARCHAR (60);

✳ DROP: It will drop (delete) the mentioned column:-

Example ALTER TABLE language DROP country;

⇒ It will remove whole column "country".

12.3 Drop Table, Truncate, Delete:

✳ Drop table: It will remove both the table and all of the data permanently.

Syntax: DROP TABLE <table name>;

Example: DROP TABLE language;

It will wipe off whole Table, while delete command is use to delete some or all rows of table. It will never delet the table structure.

NOTE- We can also add one modification to it

DROP TABLE <table_name> IF EXISTS;

⇒ It will drop the table (whole table) if it exists in data base. So that we donot get any Error if we donot know whether it exist or not.

→ Truncate: It will remove the content of table but not table structure.

Syntax - TRUNCATE TABLE <table name>;

⇕

DELETE FROM <table-name>

DATA CONTROL LANGUAGE (DCL)

13.1 Grant, Revoke:

In Data base, different people have different access to different parts of data because same database is being accessed by multiple people in an organization. Hence, a s/w developer, data scientist, admins etc all have different access permission of the database

- s/w developer → may have permission to add, delete, modify etc
- data scientist → Can only access the data (read). Cannot modify it
- admins → have permission to create, insert relation in a database.

DCL helps us to control who can access what. It is not implemented by all databases.

GRANT:

It allows specified users to perform specified tasks.

Example:- GRANT ALL ON db1.* To 'jeffrey'@'localhost';

- ALL stands for all permissions
- db1 is the name of database and * stands for all tables of database.

Example 2:- GRANT 'role1', 'role2' To 'user1'@'localhost', 'user2'@'localhost';

↳ here these are different roles like admin etc which is given to user1 and user2.

Example 3: GRANT SELECT ON world.* To 'role3';

it has given Select Permission on database named world over all of its tables/relations

REVOKE:

It cancels previously granted or denied permissions.

Example 1: REVOKE INSERT ON *.* FROM 'jeffrey'@'localhost';

- here we are revoking insert permission of this user for
- *.* → all databases and all tables of each database

Example 2: REVOKE SELECT ON world.* FROM 'role 3';

⇒ Revoking Select permission from role 3 on database named world over all of its tables.

NOTE- Best place to learn sql is "reference manual" of any database we are using.

SQL : EXAMPLES AND SOLVED PROBLEMS

Q1) A relational database contains two tables student and performance as shown below:

Student:

Roll no	StudentName
1	Amit
2	Priya
3	Rohan
4	Smita
5	Vinit

Performance:

	Roll_no	Subject Code	Marks
1	1	A	86
2	1	B	95
3	1	C	90
4	2	A	89
5	2	C	92
6	3	C	80

The primary key of the student table is Roll_no. For the performance table, the columns Roll_no and Subject_code together from the primary key. Consider the SQL query given below:

```
SELECT S.student_name, sum(P.marks)
FROM Student S, Performance P
WHERE P.marks > 84
GROUP BY S.student_name;
```

The number of rows returned by the above SQL query is _____

⟹ Given the query:

```
SELECT S.student_name, sum(P.marks)
FROM Student S, Performance P
WHERE P.marks > 84
GROUP BY S.student_name;
```

executed 1st FROM ... } → This indicates cross product of S and P

executed 2nd WHERE ... } except 6th row in "Performance" is satisfied.

executed 3rd GROUP BY ... } → this will generate 5 groups with 5 distinct student name

∴ group by will create 5 groups for 5 distinct student name hence, no. of rows / tuples = 4

(Q2) Consider the following two tables and four queries in SQL.

Book (isbn, bname), stock (isbn, copies)

Query 1:

```
SELECT B.isbn, S.copies
FROM Book B INNER JOIN stocks S
ON B.isbn = S.isbn;
```

Query 2:

```
SELECT B.isbn, S.copies
FROM B B LEFT OUTER JOIN stocks S
ON B.isbn = S.isbn;
```

Query 3:

```
SELECT B.isbn, S.copies
FROM Book B RIGHT OUTER JOIN stocks S
ON B.isbn = S.isbn;
```

Query 4:

```
SELECT B.isbn, S.copies
FROM B B FULL OUTER JOIN stocks S
ON B.isbn = S.isbn;
```

Which one of the queries above is certain to have an output that is superset of the o/p's of the other three queries?

(A) Query 1
(B) Query 2
(C) Query 3
(D) Query 4

⇒ As we know tuples we get in full outer join is union of $B \bowtie S \cup B \bowtie S \cup B \bowtie S$

Hence, correct option is the query with FULL OUTER JOIN ie Query 4 (D).

(Q.43) Consider a database that has the relational schema Emp (EmpId, EmpName, DeptName). An instance of the schema Emp and a SQL query on it are given below:

Emp:

EmpID	EmpName	DeptName
1	X4A	AA
2	X4B	AA
3	X4C	AA
4	X4D	AA
5	X4E	AB
6	X4F	AB
7	X4G	AB
8	X4H	AC
9	X4I	AC
10	X4J	AC
11	X4K	AD
12	X4L	AD
13	X4M	AE

Query —

```
SELECT AVG (EC.Num)
FROM EC
WHERE (DeptName, Num) IN
(SELECT DeptName,
   COUNT(EmpID) AS
   EC(DeptName, Num)
FROM Emp
GROUP BY DeptName);
```

The output of executing the SQL query is ———

⇒ Here the nested query is:

```
SELECT AVG (EC.Num)
FROM EC
WHERE (DeptName) IN ( SELECT DeptName,
                      COUNT (EmpID) AS
                      EC (DeptName, Num)
                      FROM Emp
                      GROUP BY DeptName);
```

This will be created first

This the alias for the new table created by the inner query

This table EC has DeptName and Num (Num → Count(EmpId))

The inner query will be executed first. It will groupby ~~the~~ based on DeptName ie AA, AB, AC, AD, AE and Count(EmpId) will give no. of employees in each department. Hence output of inner query will be:

EC:

DeptName	Num
AA	4
AB	3
AC	3
AD	2
AE	1

Now the outer query is asking for AVG on Num from resultant relation EC.

ie $\frac{4+3+3+2+1}{5} = \frac{13}{5} = 2.6$.

(84) Consider the following database table named top_scorer:

Top_scorer:

Player	Country	goals
Klose	Germany	16
Ronaldo	Brazil	15
G Muller	Germany	14
Fontaine	France	13
Pele	Brazil	12
Klinsmann	Germany	11
Kocsis	Hungary	11
Batistuta	Argentina	10
Cubillas	Peru	10
Lato	Poland	10
Lineker	England	10
T Muller	Germany	10
Rahn	Germany	10

Consider the following query:

```
① SELECT ta.player FROM top_scorer AS ta
  WHERE ta.goals > ALL (SELECT tb.goals
    ② FROM top_scorer AS tb
      WHERE tb.country = 'Spain')
    AND ta.goals > ANY (SELECT tc.goals
      ③ FROM top_scorer AS tc
        WHERE tc.country = 'Germany'));
```

The number of tuples returned by the above SQL query is _____

⟹ Here we have 2 sub-queries/nested query's.

~~First we will~~ Before start solving the innermost query (③) if we look at the first sub-query (②) we can see that it will result empty set because there is no country = "Spain" in our table given. Therefore, it will return an empty set.

Hence, it will always be true because ALL (empty set) = TRUE

Since, there is logical AND between ② and ③
∴ ② is already TRUE, So now we need to execute ③

The innermost query (③) will return:
The goals made by players whose country = "germany".

O/P ⇒

tc.goals
16
14
11
10
10

The condition before logical AND is tq.goals > Any {16, 14, 11, 10} ie tq.goals should be greater than 10.

Since query ② is ignored as it is TRUE ∴ outer query will print Player names whose goals are >10
Hence, no. of rows returned = 7

(Q5) Consider the following database table named water_schemes:

water_schemes:

Scheme_no	district_name	capacity
1	Ajmer	20
1	Bikaner	10
2	Bikaner	10
3	Bikaner	20
1	Churu	10
2	Churu	20
1	Dungargarh	10

The number of tuples returned by the following SQL query is:

```
① Q1: With total (name, capacity) as
        Select district_name, sum (capacity)
        From water_schemes
        group by district_name
② Q2: with total_avg (capacity) as
        Select avg(capacity)
        from total
③ Q3: Select name
        from total, total_avg
        Where total.capacity >= total_avg.capacity
```

(A) 1
(B) 2
(C) 3
(D) 4

⟹ With is a keyword is introduced by oracle. Newer mysql version also uses "with".
With creates a temporary relation / table with the result returned by a query and using the given table and attributes name.

Here, Q_1 will return ⇒ Total:

→ This is a temporary table which is accessed by the query coming part of further.

name	Capacity
Ajmer	20
Bikaner	40
Churu	30

This name is given by "With"

} sum of capacity of each city

NOTE - Using "with" we can access the resultant table as it get saved in memory

The output of Q_2 will be: (This query is using table created by Q_2)

Total_avg:

Capacity
25

→ avg of

$$\frac{20+40+30+10}{4} = 25$$

New table created by Q_2

The output of Q_3 will be: It will use both total and total_avg to get the result. It is performing cartesian product of total and total_avg and returns the tuples whose capacity is greater than average Capacity 25.

hence, only Bikaner and Churu are having Capacity 30 and 40 respectively which is greater than 25

Therefore, no. of tuples returned = 2 //

15.2 Solved problems-2:

(Q6) Select operation in SQL is equivalent to

(A) The selection operation in relational algebra

(B) The selection operation in relational algebra, except that select in SQL retains duplicates

(C) The projection operation in relational algebra

(D) The projection operation in relational algebra, except that Select in SQL retains duplicates.

⇒ As we know select operation in SQL select columns while selection operation in RA select rows / tuples. While Projection in RA selects columns but it removes duplicates but select ~~set~~ in SQL retains duplicates.

Hence, Correct option is (D).

(Q 7) Consider the following relations:

Students:

Roll_No	Student_Name
1	Raj
2	Rohit
3	Raj

Performance:

Roll_No	Course	Marks
1	Math	80
1	English	70
2	Math	75
3	English	80
2	Physics	65
3	Math	80

```
SELECT S.student_Name, SUM(P.marks)
FROM student S, Performance P
WHERE S.Roll_No = P.Roll_No
GROUP BY S.student_Name;
```

} This is equivalent to JOIN

The number of rows that will be returned by the SQL query is _____

(A) 0
(B) 1
(C) 2
(D) 3

⇒ Since, here we are using group by using Student_name and we

(Q8) Consider the following relation:

Cinema (theatre, address, capacity)

Which of the following options will be needed at the end of the SQL query:

```
SELECT P1.address
FROM Cinema P1
```

Such that it always find the addresses of theatres with maximum capacity?

(A) WHERE P1.capacity >= ALL (SELECT P2.Capacity FROM Cinema P2)

(B) WHERE P1.capacity >= ANY (SELECT P2.Capacity FROM Cinema P2)

(C) WHERE P1.capacity > ALL (SELECT MAX(P2.capacity) FROM Cinema P2)

(D) WHERE P1.capacity > ANY (SELECT MAX(P2.capacity) FROM Cinema P2)

⟹ 'Any' will not work here because ~~any~~ even if there is even 1 theatre whose capacity is max it will get picked up. So, only options left are (A) and (C). When I'm selecting maximum capacity of cinema hall nothing could be greater than it so (C) will not work because ALL will return only one value (which is maximum value) and ∴ it should be >= and not >

Hence, correct and feasible option is (A)

(89) Given the following statements:

S_1: A foreign key declaration can always be replaced by an equivalent check assertion in SQL.

S_2: Given the table R(a, b, c) where a and b together from the primary key, the following is a valid table definition.

```
CREATE TABLE S (
    a Integer,
    d Integer,
    c Integer,
    PRIMARY KEY (d),
    FOREIGN KEY (a) References R )
```

Which one of the following statement is CORRECT?

(A) S_1 is TRUE and S_2 is FALSE

(B) Both S_1 and S_2 are TRUE

(C) S_1 is FALSE and S_2 is TRUE

(D) Both S_1 and S_2 are FALSE

→ S_1 : It is not always delete because we do have Cascade option on delete or update.

S_2: In table S, primary key is d and foreign key a, should point to primary key of R. In R (a, b) is a primary key not 'a' alone is a primary key. So, 'a' in table S should refer to (a, b) and not only to 'a'. Hence, it is false

Therefore, Correct option is (D) Both are false.

(Q10) Given the following schema:

employees (emp_id, first_name, last_name, hire_date, dept_id, salary)

department (dept_id, dept_name, manager_id, location_id)

You want to display the last names and hire dates of all latest hires in their respective departments in the location ID 1700. You issue the following query:

```
SQL > SELECT last_name, hire_date
      FROM employees
      WHERE (dept_id, hire_date) IN
      (SELECT dept_id, MAX(hire_date)
       FROM employees JOIN departments USING (dept_id)
       WHERE location_id = 1700
       GROUP BY dept_id);
```

(inner query)

What is the outcome?

(A) It executes but does not give the correct result.

(B) It executes and gives the correct result.

(C) It generates an error because of pairwise comparison.

(D) It generates an error because of the GROUP BY clause cannot be used with table joins in a sub query.

⇒ Since there is no logical and syntactical error in query therefor, it will execute and o/p generated by the query is same as what given in english statement. Hence, it will execute and will give correct answer. Therefore, (B) is the correct answer.

15.3 Solved Problem - 3 :

(Q 11) SQL allows tuples in relations, and correspondingly defines the multiplicity of tuples in the result of joins. Which of the following queries always gives the same answer as the nested query shown below:

SELECT * FROM R WHERE A in (SELECT S.a FROM S)

(A) SELECT R.* FROM R, S WHERE R.a = S.a

(B) SELECT DISTINCT R.* FROM R, S WHERE R.a = S.a

(C) SELECT R.* FROM R, (SELECT DISTINCT A FROM S) AS S1 WHERE R.a = S1.a

(D) SELECT R.* FROM R, S WHERE R.a = S.a AND IS UNIQUE R.

$\Longrightarrow$ Let us consider a sample relation instance of R and S:

R :

a
1
1
2
2
3

S :

a
1
1
2
2

So, the o/p of original query will be :

a
1
1
2
2

Option A will return:

q
1
1
1
1
2

Option B will return:

q
1
2

Option C will return:

q
1
1
2
2

It will do join

S.q

a
1
2

⋈

R.

q
1
1
2
2
3

Hence, correct option is (C)

(Q 12) Consider the following relational schema:

Employee (empid, empName, empDept)

Customer (custid, custName, salesRepid, rating)

SalesRepid is a foreign key referring to empid of the employee relation. Assume that each employee makes a sale to at least one customer. What does the following query return?

```
SELECT empName
    FROM employee E
    WHERE NOT EXISTS (SELECT custid
    FROM Customer C
```

```
WHERE C.salesRepid = E.empid
AND   C.rating <> 'GOOD');
```

(A) Names of all employees with at least one of their customers having a 'GOOD' rating.

(B) Names of all employees with at most one of their customers having a 'GOOD' rating.

(C) Names of all the employees with none of their customers having a 'GOOD' rating.

(D) Names of all the employees with all their customers having a 'GOOD' rating.

$\Rightarrow$ The assumption made in question "Each employee makes a sale to at least one customer" means that in SalesRepid, every empid exists at least once.

The inner query will give : Custid for SalesRepid = empid and rating is not equal to "good".
Outside the inner query, we have NOT EXISTS. So, its kind of like negation over negation (as in inner query we have rating != "Good"). So, any employee who have even 1 customer whose rating is not equal to good will not be there in the result set.
So, option A, B and C are wrong. Hence, D is the most appropriate option.

(Q 13) Consider the following tables A, B and C:

Table A:

Id	Name	Age
12	Arun	60
15	Shreya	24
99	Rohit	11

Table B:

Id	Name	Age
15	Shreya	24
25	Hari	40
98	Rohit	20
99	Rohit	11

Table C:

Id	Phone	Area
10	2200	02
99	2100	01

How many tuples does the result of the following SQL query contains?

```
SELECT A.id
FROM A
WHERE A.age > ALL (SELECT B.age
                   FROM B
                   WHERE B.name = 'arun')
```

(A) 4
(B) 3
(C) 0
(D) 1

⇒ The inner sub-query will return empty set and we know ALL (empty set) = True

Hence, A.age > ALL( ) is true for all tuples. and it will return A.id for all A.age > all - Therefore, it will return 3 tuples.

(Q 14) Database table by name loan_Records is given below:

Borrower	Bank_manager	loan_Amount
Ramesh	Sundarajan	10000.00
Suresh	Ramgopal	5000.00
Mahesh	Sundarajan	7000.00

What is the o/p of the following SQL query?

```
SELECT COUNT(*)
FROM ((SELECT Borrower, Bank_Manager
       FROM Loan_Records) AS S
      NATURAL JOIN (SELECT Bank_manager, loan_Amount
                    FROM Loan_Records) AS T);
```

(A) 3
(B) 9
(C) 5
(D) 6

⇒ According to the query we will have

S:

Borrower	Bank_manager
Ramesh	Sundarajan
Suresh	Ramgopal
Mahesh	Sundarajan

NATURAL JOIN

T:

Bank manager	Loan_Amou
Sundarajan	10000.00
Ramgopal	5000.00
Sundarajan	7000.00

It will result in distinct 5 tuples :

Borrower	Bank_manager	Loan_Amount
Ramesh	Sundarajan	10k
Ramesh	Sundarajan	7k
Suresh	Ramgopal	5k
Mahesh	sundarjanan	10k
Mahesh	Sundarajan	7k

5 distinct tuples

Hence, output will be 5 tuples (as it is count(*))

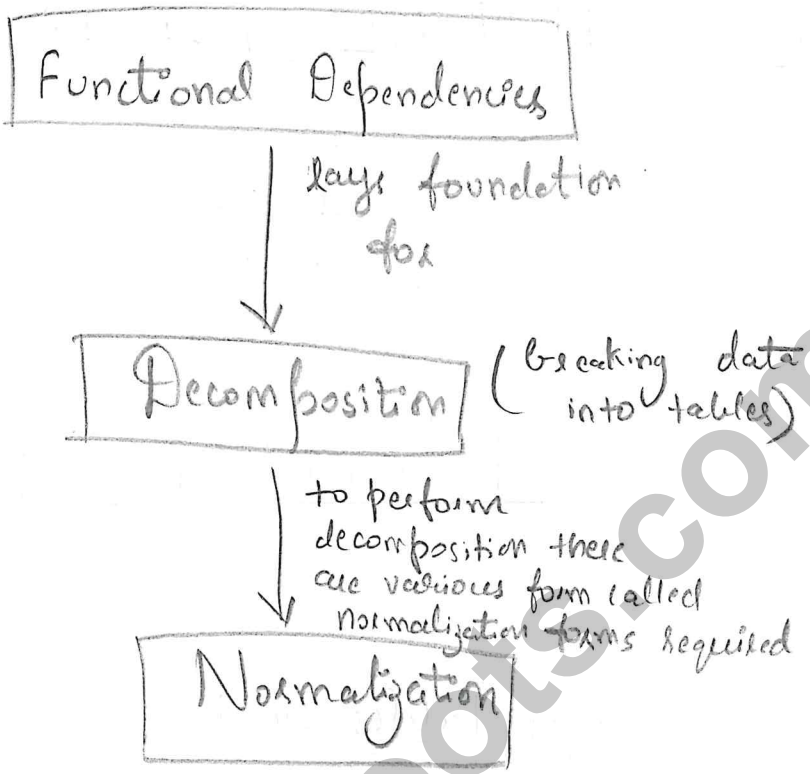
FUNCTIONAL DEPENDENCIES AND DECOMPOSITION

16.1 Designing tables in DB:

Here, database corresponds to the Relational database. The objectives we carry while designing a database is:

* From ER diagrams → Tables, our objective is to minimize the number of tables.
* Another objective while designing a db is minimizing storage or disk space while not discarding any data/information. (This could be done by avoiding duplicate data, it is one of the way to minimize disk storage)
* One of the main objective while designing database is fast retrieval of data. In current scenario the storage cost has fallen drastically while demand for speedy retrieval of data.

16.2 Functional Dependencies:



Consider, a real world example to understand the concept of functional dependencies

Customer (id, name, zip)

There are certain properties of each relations which are not given till now (explicitly). like:

⟹ Given ~~a~~ an 'id' we can uniquely determine name of a customer. It is represented as:

$$id \longrightarrow name$$

⟹ Given an 'id' we can uniquely determine ~~the~~ zip code of a customer.

$$id \longrightarrow zip$$

⟹ Given an 'id' we can uniquely determine both name and zip code of a customer. ie

$$id \longrightarrow name, zip$$

Note Here, '$\longrightarrow$' is a notation which means LHS functionally determines RHS.

It means, 'id' functionally determines name and zip.

In general we can give it as:

Given relation, $R(A_1, A_2 \ldots A_n)$

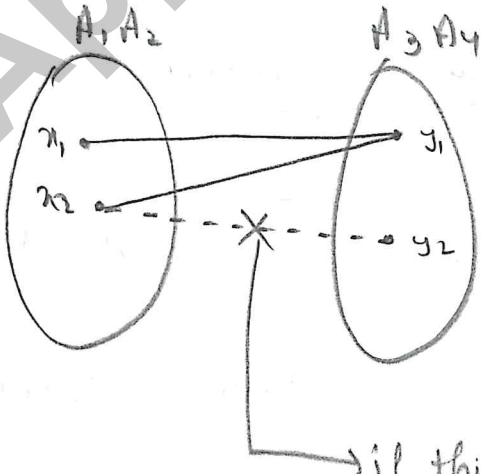
let

$$\{A_1, A_2\} \longrightarrow \{A_3, A_4\}$$

Here, for given values of A_1 and A_2 we can functionally determine A_3 and A_4.
(uniquely determine)

> Formally, Functional dependency is defined as a constraint between sets of attributes in a relation.

Another way to understand FD is set theoretic way:

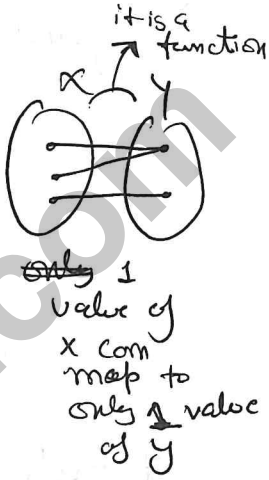


if this line exists than for give value x_2 we cannot uniquely determine A_3A_4 and hence, this FD will fail. So, inorder to hold this FD this dotted line cannot exist.

NOTE - Logically: $X \rightarrow Y$ iff $((t_1.x = t_2.x) \Rightarrow (t_1.y = t_2.y))$ where t_1 and t_2 are two different tuples.

Mathematically: $\Pi_{x,y}(R)$ is a function

$X \rightarrow Y$



In practice, FD's help us to encode business logic.

For ex- Given a name we cannot uniquely determine an id of a customer

name $\not\rightarrow$ id (There could be 2 same name with diff id)

⇒ Types of Functional dependencies:

- Trivial FD
- Non-Trivial FD

(I) TRIVIAL FD's:

For given relation Customer (id, name, zip), the trivial FD's will be:

- id $\rightarrow$ id
- id, name $\rightarrow$ id
- id, name $\rightarrow$ name

} all are trivial FD's

$X \to Y$

An FD is called trivial FD in relation R, if X is a superset of Y ($X \supseteq Y$). In previous 3 examples the L.H.S is the superset of R.H.S. Hence, they are trivial.

(II) NON-TRIVIAL FD's:

Given a relation R, an FD $X \to Y$ is said to be non-trivial if X is not a superset of Y ($X \not\supseteq Y$).

For the customer relation, FD id → name is a non-trivial functional dependency.

⇒ Inference Rules:

* Based on real world problem / system (Properties of system) that we are modelling, we derive / get some FD's (Based on ground reality, or system we get some FD's).

  For example: Multiple people can have same name, (this is one of the property of the system) hence we cannot have name → Id

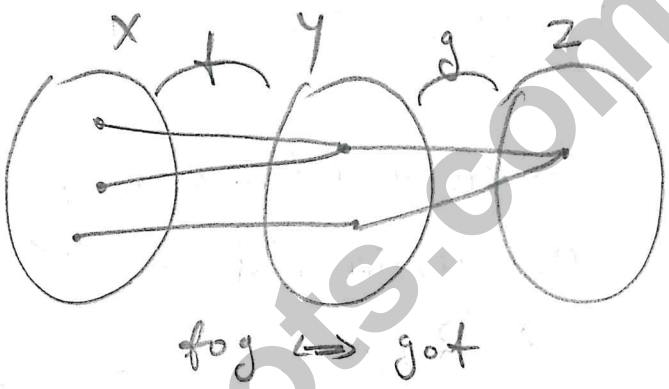
* Based on the predefined properties of the system and existing FD's we can define some rules to obtain new FD's. These are as follows:

Assume, there is Relation R with x, y, z as its attributes

R(x, y, z)

- Reflexivity ⇒ if $x \supseteq y$ (x is superset of y) then $x \rightarrow y$.

- Transitivity ⇒ if $x \rightarrow y$ and $y \rightarrow z$ then $x \rightarrow z$



- Augmentation ⇒ if $x \rightarrow y$ then $xz \rightarrow yz$ $(z \rightarrow z)$

- Decomposition ⇒ if $x \rightarrow yz \Rightarrow x \rightarrow y$ and $x \rightarrow z$

- Union ⇒ if $x \rightarrow y$ and $x \rightarrow z \Rightarrow x \rightarrow yz$

- Composition ⇒ if $x \rightarrow y$ and $z \rightarrow w \Rightarrow xz \rightarrow yw$

(here x, y, z, w all are attributes of relation R)

- Pseudo-transitivity ⇒ if $x \rightarrow y$ and $wy \rightarrow z \Rightarrow wx \rightarrow wy \Downarrow wx \rightarrow z$

(Augmentation + transitivity)

NOTE- A common mistake comitted by people is: Given $xy \to z$, they decompose LHS, which is wrong. We always decompose RHS.

Hence $xy \to z \not\Rightarrow x \to z$ and $y \to z$

Example:

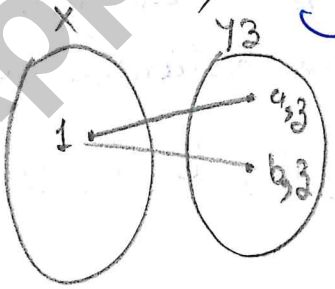
R:

X	Y	Z
1	a	α
1	b	α
2	a	α
2	b	α

$\Rightarrow$ here the non-trival dependency which will hold are:

$x \to z$ (given x we can uniquely determine z)

$\Rightarrow x \to y$ and $x \to yz$



These doesnot hold because given x we cannot uniquely determine y or yz.

$\Rightarrow y \to z$ (it will hold) (valid)

$\Rightarrow y \to x$ and $y \to xz$ (It doesnot hold) (not valid)

$\Rightarrow xy \to z$ (It will hold) (valid)

$yz \to x$ and $xz \to y$ } (It it will not hold) (not valid)

Other non-trivial FD's are $xy \to yz$, $yx \to zx$.

16.3 Attribute closure, Keys and Solved problems:

⇒ Attribute closure :-

It is a set of attributes that can be functionally determined using attributes in X. (R is a relation with X as a set of attributes). It is represented as 'X^+' (here, X can be single att. or multiple attribute (set))

Example 1: Given relation R(A, B, C, D) and

FD = { A → B, B → C, C → D }

(Set of FD's)

⟹ now A^+ will give all the attributes we can determine or get using A

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

↳ (Because it is, trivial)

Similarly, $B^+ = \{B, C, D\}$ (using B we can determine B, C and ∴ we can determine C hence using (B, C) we can determine D)

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

$D^+ = \{D\}$

Example 2: Given relation $R(A,B,C,D,E,F,G)$ and

$FD = \{AB \to CD,\ AF \to D,\ DE \to F,\ C \to G,\ F \to E,\ G \to A\}$

⟹ Now if we want to find closure of CF ($\{CF\}^+$) it will be given as:

$\{CF\}^+ = \{C, F, E, G, A, D\}$

- C, F — trivial
- E — using F we can determine E ($F \to E$)
- G — using C ($C \to G$)
- A — ∵ we got G and $G \to A$
- D — ∵ we got A and $AF \to D$

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

- B, G — trivial
- A — using G, $G \to A$
- C, D — ∵ A is there and hence $AB \to CD$

$\{AF\}^+ = \{A, F, D, E\}$

- A, F — trivial
- D — using A and F, $AF \to D$
- E — using F, ∵ $F \to E$

Note:

*Application of Attribute closure concept is to determine super keys and candidate keys. If we have a relation and FD for a system then we can determine all of its candidate key using this concept.

If X^+ = all attributes in/of R ⇒ uniquely determine a tuple/row

then X is a super key.

∴ A minimal super key is a candidate key. Minimal means, if no subset of or strict subset of a superkey is a key,

Then, that superkey is a candidate key.
That is, if X is a superkey $\Rightarrow X^+$ = all attributes of R

Then, X is a candidate key iff $\forall Y \subset X$
$Y^+ \neq$ all attributes of R.

Hence, we can pick any of the candidate key to be the primary key.

Example 3: Given relation R(ABCDE) and FD = { A→BC, D→E }. What is the superkeys (SK) and Candidate keys (CK)

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

↑ Superkey.

Now this superkey to be the candidate key, no subset of it is A and D should uniquely determine all attributes. Therefore we will check A^+ and D^+

$A^+ = \{A, B, C\}$
$D^+ = \{D, E\}$
} not superkeys

Hence, no subset of {AD} is a superkey. Therefore {AD} is the candidate key.

Example 4: Given relation R(A B C D E) and FD = { AB → C , C → D , B → E } what is the CK?

⟹ $\{AB\}^+ = \{A, B, C, D, E\}$

$\{A\}^+ = \{A\}$
$\{B\}^+ = \{B, E\}$
} Not a super key.

Hence, AB is a candidate key.

2nd

NOTE- Attributes which are part of atleast one candidate key are called as prime attribute. In the above example 4, A and B are prime attributes.
While rest of the attributes are non-prime attributes.

The prime attribute concept is important because:
lets understand it using an example. Given relation R(A, B, C, D, E) and FD's = { AB → C, C → D, B → E, E → B }

⟹ here CK will be as

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

while $A^+ = \{A\}$ and $B^+ = \{B, E\}$

So here prime attributes are {A, B}
In FD's we can see there is functional dependency E → B
So whenever we have any FD such that

X ⟶ prime attribute

then, we may have more candidate keys.

So, if we replace that prime attribute with X then it will also become a candidate key.

ie $AE^+ = \{A, B, C, D, E\}$ is a candidate key

as $A^+ = \{A\}$ and $E^+ = \{E, B\}$.

And prime attributes are $\{A, B, E\}$ now

Hence, this will have 2 candidate keys AB and AE.

$$CK = \{AB, AE\}$$

So, concept of prime attribute helps us to determine all possible candidate keys.

Example 5: $R(ABCDE)$

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

CK?

$\Longrightarrow$

$AB^+ = \{A, B, C, D, E\}$ } This is not a CK because subset of it can uniquely determine all attributes.

$A^+ = \{A\}$

$B^+ = \{B, A, C, D, E\}$ } ∵ it can determine all attributes ∴ it is a CK

~~$E^+ = \{$~~

So prime attributes are: $\{B\}$

But, Here we have FD = $C \to B$ (ie X → Prime attribute)

hence C is also a CK

Hence, Prime attributes are = $\{B, C\}$

NOTE- C is being determined by both AB ($AB \to C$) but AB could not be a CK because B itself is a CK ∴ AB is a SK.

Example 6:

$R(ABCDE)$

$FD = \{A \to BCDE, BC \to ADE, D \to E\}$

Ck?

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

Here A is also being determined by BC

$BC \to ADE$

∴ we will check for BC also

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

∴ $B^+ = B$, $C^+ = C$ ∴ BC is a ck

Hence, prime attributes are : $\{A, B, C\}$

candidate keys are : $\{A, BC\}$

Example 7:

$R(ABCD)$

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

Ck?

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

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

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

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

Hence, all A, B, C, D are ck.

This type of FD is called cyclic FD.

Example 8: $R\{ABCDE\}$

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

CK?

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

AB is a CK Bcoz $A^+ = A$, $B^+ = B$

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

AF is a CK Bcoz $F^+ = FB$

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

AE is also a CK Bcoz $E^+ = E, B, F$

Hence CK = {AB, AF, AE, AD, AC}

and prime attributes are = {A, B, E, F, D, C}

Similarly for AD and AC.

Example 9: $R(ABCDE)$

$FD = \{AB \to C,\ BC \to D\}$

CK?

$\Rightarrow$ Here, E is not part of any FD

$\Downarrow$

E has to be part of every Candidate Key as it could not be determined by any other attribute

$ABE^+ = \{A, B, E, C, D\}$ is a CK because $AE^+ = A, E$ and $BE^+ = B, E$

Example 10 : $R(A\ B\ C\ D\ E\ F\ G)$

$FD = \{ AB \to CDEF,\ C \to ADE,\ D \to EBF,\ F \to DA,\ BE \to AF \}$

CK ?

$\Longrightarrow$ Here, G is not in any FD $\Longrightarrow$ G Has to be part of every CK.

$AB\,G^+ = \{A, B, G, C, D, E, F\}$

$AG^+ = \{A, G\}$

$BG^+ = \{B, G\}$

Hence ABG is a CK.

∵ we have a FD $C \to ADE$ ($X \to$ Prime attribute)

$CBG^+ = \{C, B, G, A, D, E, F\}$

$CG^+ = \{G, C, A, D, E, B, F\}$

$BG^+ = \{B, G\}$

∴ CBG is ~~also a ck~~ not CK. CG is a CK

∵ we also have FD $D \to BEF$ ($X \to$ Prime attrik

$DG^+ = \{D, G, E, B, F, A, C\}$

Similarly $BEG^+ = \{B, E, G, A, C, D, F\}$

like This we have multiple CK here

$\{ABG, CG, DG, FG, BEG\}$

Prime attributes = $\{A, B, C, D, E, F, G\}$

16.4 Functional dependency Sets and Properties + Solved Examples:

→ Properties of functional dependency set:

Membership:

Let F be a FD set on R, if $X \to Y$ using FD in F then, $X \to Y$ is a member of F

Here, if $X \to Y$ using FD's in F means $X^+ \supseteq Y$ than using FD in F.

Example 1 R(A B C) and FD = {A → B, B → C}

(Q1) Is A → C is a member of F ?

⟹ $A^+ = \{A, B, C\}$ ⟹ Using FD in set F we can say that A → C which means A → C is member of F.

(Q2) Is C → A is a member of F ?

⟹ To prove this true we should be able to get C → A using FD's in F.

$C^+ = \{C\}$ ⟹ using FD's in F C^+ can give only C and they cannot determine any other attributes.

Hence C → A and it is not member of F.

Example 2 Given R(A B C D E) and FD = {AB → C, A → D, CD → E}

Is AC → E is a member of FD set F?

⟹ $AC^+ = \{A, C, D, E\}$ ⟹ ∵ AC can determine E
∴ AC → E holds and it is a member of F.

Example 3: Given two conditions A ⊇ B and B → C does A → C holds?

⟹ Here we can construct our FD set. Since,
A is superset of B
∴ A → B
and B → C is already given
∴ FD = {A → B, B → C}

∴ $A^+ = \{A, B, C\}$ ⟹ hence C could be determined by A and ∴ A → C is member of FD set F.

Example 4: R(A, B C D) and FD = {AB → C, BC → D}
Does A → C is a member of FD set F?

⟹ $A^+ = \{A\}$ ⟹ C could not be determined by A hence A → C does not hold and hence, it is not a member of F.

(Please note, all closures are performed under FD set F)

Closure of a FD set:

It is also written as F^+. It is a set of all FD that can be determined using FD in F.

Example 1: $R\{A\ B\ C\}$ and $F = \{A \to B, B \to C\}$ what is F closure (F^+)?

$\Rightarrow$ F^+ will include all possible trivial and non-trivial dependencies.

$$F^+ = \left\{ \begin{matrix} A \to A \\ A \to B \\ A \to C \end{matrix} \middle| \begin{matrix} AB \to A \\ AB \to B \\ AB \to C \end{matrix} \middle| \ldots \right\}$$

It is a very large set

A question which use to get framed from this concept is

(Q) Number of FD's in F^+?

$\Rightarrow$ For the same example R(ABC) and FD = {A→B, B→C} we will try to get ~~all~~ total number of FD in F^+

* FD's with 0 attributes $\Rightarrow$ $\phi \longrightarrow \phi$ // Valid FD bcoz any dependency to be FD $X \to Y$ $X \supseteq Y$. This subset could be empty.

So, 1 FD is possible with 0 attributes.

* FD's with 1 attribute (by 1 attribute we mean LHS is having a single attribute) in FD

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

$A \rightarrow$ B, C, BA, AB, ..., ABC

any subset of it $= 2^3 = 8$

+

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

$B \rightarrow$ B, C, BC

any subset of it $2^2 = 4$

+

$C^+ = \{C\}$

$C \rightarrow C$

any subset of it $2^1 = 2$

So, total $8 + 4 + 2$ FD's are possible with 1 attribute in LHS

* FD's with 2 attributes $\rightarrow AB^+ = \{A, B, C\}$

$AB \rightarrow$ A, B, ..., ABC

all subset of it $2^3 = 8$

+

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

$BC \rightarrow$ B, C, BC

all subset of it $2^2 = 4$

+

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

$AC \longrightarrow$ A, B, ..., ABC — all subset of it $= 2^3 = 8$

So, total 8 + 4 + 8 ~~attr~~ FD possible with 2 attributes in LHS

* FD's with 3 attributes $\Rightarrow ABC^+ = \{A, B, C\}$

$ABC \longrightarrow$ A, B, ..., ABC — all subset of it $2^3 = 8$

So, 8 FD's are possible with 3 attributes in LHS

Hence, total no. of FD's in F^+ given $= 8 + 8 + 4 + 8 + 8 + 4 + 2 + 1$

$= \boxed{43 \text{ FD's}}$

So, ~~error~~ on seeing F, among 43 FD's we need only 2 FD's to determine rest of 41. Therefore, these 41 FD's are called redundant FD's as we can derive these 41 just using these 2 FD's ($\{A \to B, B \to C\}$).

Equality of FD set :

Given two functional dependency set F and G, equality of FD set means that F and G are equal iff

(i) $F^+ = G^+$ (But it is very tedious to compute)

OR

(ii) F covers G and G covers F

it means if all FD's in G can be determined using FD's in F and viceversa.

Note
it can be thought as 2 diff engineer come up with 2 FD set based on their understanding. Now to check if both set are equivalent this property is used.

Example 1:

$R(ABCD)$

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

$G = \{AB \to C, AB \to D, C \to D\}$

Is $F = G$?

$\Rightarrow$ F covers G?

$AB^+_F = \{A, B, C, D\} \Rightarrow AB \to C$ and $AB \to D$ in G

(it means AB^+ under F)

$C^+_F = \{C, D\} \Rightarrow C \to D$ in G

Hence, F Covers G.

G covers F?

$AB^+_G = \{A, B, C, D\} \Rightarrow AB \rightarrow CD$ ↵ (in F)

(it means AB^+ under G)

$B^+_G = \{B\} \Rightarrow B \not\rightarrow C$

∵ F have $B \rightarrow C$ but G doesnot hold it

$C^+_G = \{C, D\} \Rightarrow C \rightarrow D$ (in F)

∴ G doesnot cover F.

∴ $F \neq G$

(For F and G to be equivalent both F and G should cover each other)

Example 2:

$R\{A\ B\ C\ D\ E\ F\ H\}$

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

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

$F = G$?

⟹ F Covers G –

$A^+_F = \{A, C, D\} = A \rightarrow CD$ (Could be determined using F)

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

Hence, F Covers G

G Covers F –

$A^+_G = \{A, C, D\} \Rightarrow A \rightarrow C$ (in F)

$AC^+_G = \{A, C, D\} = AC \rightarrow D$ (in F)

$E^+_G = \{E, A, H, C, D\} = E \rightarrow AD$ and $E \rightarrow H$

Hence, G Covers F

Hence F = G

16.5 Minimal / Canonical Covers of FD's + Solved Problems:

Objective: Eliminating functional dependencies which are redundant in an FD-set F.

Redundant FD's are those whose removal doesn't impact F^+.

Example 1: $R(A\,B\,C)$, $F = \{A \to B,\ B \to C,\ A \to C\}$

What is minimal cover of given set F.

$\Rightarrow$

$A^+ = \{A, B, C\}$ $\xrightarrow{\text{using } A \to B}$

$B^+ = \{B, C\}$ $\xrightarrow{\text{using } B \to C}$

So, $F_1 = \{A \to B,\ B \to C\} \to A \to C$

$\therefore F^+ = F_1^+$

Hence, $A \to C$ is redundant as we can get C using the remaining FD's.

So this F_1 is called minimal cover of F as it is the smallest set of dependency possible to get all the FD's preserved.

Note

The need of minimal cover is to minimize the number of FD's. In SQL FD's are enforced using CHECK and CONSTRAINTS. So smaller the no. of FD's lesser the CHECK operation to be performed and faster the database queries will be executed.

NOTE

Extraneous variable / attribute :- The variable / attribute ~~which is not responsible~~ is said to be extraneous, if we can remove it and its removal doesnot bring any change in the Closure of the set of FD's.

Ex- $R(A\ B\ C)$

$F = \{ A \to C$
$AB \to C \}$

$\Rightarrow$ here B is extraneous variable

Minimum cover of F $= F_1 = \{ A \to C \}$

Getting rid of all extraneous attribute is a way to obtain a minimal cover of FD set F.

Key idea is : if we have $X \to \omega$ and $XY \to \omega$ then we can easily remove Y as we can already determined ω using X. So removal of this attribute in a FD doesnot change the expressive power of F.

ie $F^+ = F_1^+$

Task : Given FD sets find minimal covers of F.

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

$\Rightarrow$

Steps : Find FD's that have same LHS and combine them using Union principle.

$\Rightarrow$ $A \to BC$, $A \to B$ } Combine them using Union

$\Downarrow$

$A \to BC$ } ~~New~~ in place of having both $A \to BC$ and $A \to B$ we will have $A \to BC$

So, now new FD set is

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

Step 2: Find FD's that have extraneous attribute/variables in LHS or RHS

⇒ we have $AB \to C$ and $B \to C$

∵ B can uniquely determine C than we dont need AB to determine C

So, now new FD set will be

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

Step 3: Remove any extraneous attribute which you can find now.

⇒ we have $A \to BC$ and $B \to C$

∵ Using A we can determine B and C and using B we can determine C

that means A is not only one who determine C. Therefore, even if we drop C from $A \to BC$, it will not bring any change

Hence, new FD set will be

$$F_3 = \{ A \to B,\ B \to C \}$$

NOTE –

Here, we can notice that in step 3 we have not decrease/removed any FD, we can see here that we are even minimizing the individual FD's also. So using minimal cover concept we not only remove the redundant FD's we also minimize the individual FD's.

Step 4: Repeat step 1, 2 and 3 till minimal cover is obtained (ie F_i doesn't change)

$\Rightarrow$ The ~~minimal cover~~ FD's obtained in step 3 is the minimal cover of F as we cannot remove any FD further and cannot minimize the individual FD's also

Hence $F^+ \Longleftrightarrow F_3^+$

Example 2: $F = \{ ABCD \to E,\ E \to D,\ A \to B,\ AC \to D \}$

$\Rightarrow$ Step 1: No common/same LHS FD's are there.

Step 2:

$$\left.\begin{matrix} A \to B \\ ABCD \to E \end{matrix}\right\} \Rightarrow \begin{matrix} A \to B \\ ACD \to E \end{matrix}$$

We can remove B from $ABCD \to E$ because given A we can uniquely determine B.

$F_1 = \{ A \to B,\ ACD \to E,\ E \to D,\ AC \to D \}$

Step 3:

$$\left.\begin{matrix} ACD \to E \\ AC \to D \end{matrix}\right\} \Rightarrow \begin{matrix} AC \to D \\ AC \to E \end{matrix}$$

Now these 2 have same LHS Hence we can combine them.

$$\left.\begin{matrix} AC \to E \\ AC \to D \end{matrix}\right\} \Rightarrow AC \to DE$$

$F_2 = \{ AC \to DE,\ E \to D,\ A \to B \}$

Step 4: $E \to D$, $AC \to E$ } $E \to D$, $AC \to E$

$F_3 = \{ E \to D, AC \to E, A \to B \}$

* Step 5: Common mistake:

People try to ~~from~~ drop C from $AC \to E$. But it will be wrong

ie $F' = \{ E \to D, A \to E, A \to B \}$ ✗

because E could be determined only using A and C and not only using A.

In this case $F \neq F'$

(minimal cover should not impact F^+)

So, always verify at the end whether the minimal covers ~~cover~~ F and viceversa

Hence, the minimal cover of F is F_3

$\{ E \to D, AC \to E, A \to B \}$

Example 3: $F = \{ ABC \to CD, BC \to D, A \to B, C \to D \}$

$\Rightarrow$ Step 1: no Common / ~~LHS~~ Same Variable in LHS

Step 2: $A \to B$, $ABC \to CD$ } $A \to B$, $AC \to CD$

$F_1 = \{ A \to B, AC \to CD, BC \to D, C \to D \}$

Step 3: $C \to D$, ~~$AC \to CD$~~ $\Rightarrow C \to D$

(Because C already determined D alone ∴ we don't need A seperately. And $C \to CD \Rightarrow C \to C$ (trivial) and $C \to D$ (repeated)

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

~~Step~~ Again, ~~$BC \to D$~~, $C \to D$ $\Rightarrow C \to D$

(C can determine D ∴ we don't need B hence we don't need rest of $C \to D$ as it is already there

$$\therefore F_3 = \{ A \to B,\ C \to D \}$$

Hence, minimal cover of F is $F_3 = \{ A \to B,\ C \to D \}$

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

$\Rightarrow$ We can write it as:

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

(using decomposition laws)

Since, we have $A \to B$ and $B \to C$

than using transitivity we can get $A \to C$

Hence, we can remove it.

Similarly, $B \to C$ and $C \to A$, than using transitivity again we can drop $B \to A$.

Similarly, we can drop $C \to B$ using transitivity because we have $C \to A$ and $A \to B$.

So now FD set is $F_2 = \{ A \to B, B \to C, C \to A \}$

(This FD set is cyclic FD's)

So F_2 is the minimal set/cover of F.

This problem is solved by minimizing the individual FD's only (step 3) as we have not removed any FD.

Example 5: $F = \{ A \to BC, CD \to E, E \to C, D \to EAH, ABH \to BD, DH \to BC \}$

$D \to EAH$ ⟹ $\{ D \to E, D \to A, D \to H \}$

⟹ **Step 1:** No common LHS

Step 2: ∵ $D \to EAH \Rightarrow D \to E, D \to A, D \to H$

~~$CD \to E$~~

∵ D can determine E uniquely ∴ we don't need C and hence we can drop ~~$CD \to E$~~

$D \to EAH$

$F_1 = \{ A \to BC, E \to C, D \to EAH, ABH \to BD, DH \to BC \}$

∵ $AB H \to BD \} \Rightarrow AH \to D$ $(XY \to XZ \Rightarrow Y \to Z)$

$F_2 = \{ A \to BC, E \to C, D \to EAH, AH \to D, DH \to BC \}$

∵ $D \to EAH \Rightarrow D \to E, D \to A, D \to H$

~~$DH \to BC$~~

$DH^+ = \{ D, H, E, A, C, B \}$ and $D \to H$ ∴ no need to have

$F_3 = \{ A \to BC,\ E \to C,\ D \to EAH,\ AH \to D \}$

We cannot further reduce individual FD's further hence, $F^+ = F_3^+$. (F_3 is minimal cover of F)

16.6 Decomposition with Solved Problems:

Given a relation $R(A_1, A_2 \dots A_n)$ and set of FD's $F = \{ \dots \}$, breaking R into further relations $R_1, R_2 \dots R_n$ such that all attributes of R get covered in decomposed relations. (Breaking a larger relation into smaller relation)

→ Properties of decomposition: (Properties to be hold while performing decomposition)

- No redundancy or minimal redundancy:

It means that we want to minimize the disk storage by avoiding storage of duplicate / redundant data. Hence, ~~don't store du~~ decomposition should be done such that no duplicate / minimum duplicate data is stored.

- Lossless Decomposition:

Decomposition should be done such that there is no loss of data while combining / joining them later. (lossless join)

Dependency preserving:

It means that while decomposing R into $R_1, R_2 \ldots R_n$, all FD's in F should be preserved by $R_1, R_2 \ldots R_n$.

(Normalization is a principled and systematic approach to decomposition)

To understand the concept of decomposition, let us consider an example below:

Cid	Pid	Cname	Pname	time
i_1	j_1	n_1	p_1	t_1
i_1	j_2	n_1	p_2	t_2
i_2	j_2	n_2	p_2	t_3
i_3	j_3	n_3	p_3	t_4

Redundancy

Redundancy

Fig: 33 – Purchases

If we have only 1 relation like this, we will have lots of redundancy / repeated storage of lots of information.

Note:

There are several anamolies which we have to see in this concept of decomposition (anamolies which are present when we have only 1 relation)

* Update anamoly: If we want to change P1 value in Column Pname to P1'. In such cases, we need to update multiple cells which is time consuming

* Insertion anamoly: If we want to insert data about a new customer having tuple values as follows. (Note that this is a new customer and has not purchased anything yet, therefore entry of this customer in the purchase table will be like)

(i4, NULL, n4, NULL, NULL)

Here, we are wasting alot of space by having so many NULL values.

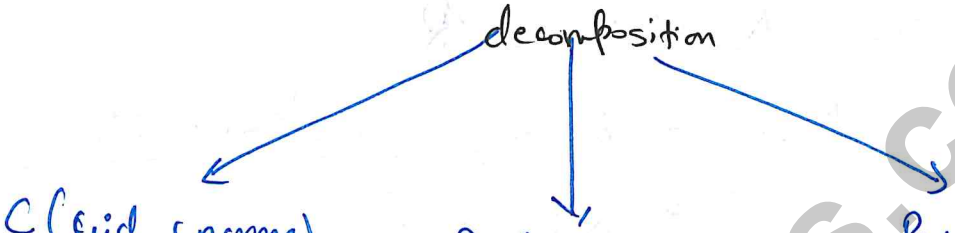
* deletion anamoly: Suppose we want to delete the information of product ie Pid and Pname. If we delete the whole tuple, we will also loose customer information, so the only ~~update~~ alternate option is to update them to NULL.
So deletion anamoly will result in too many NULL values in relation and updation of multiple cell to NULL values which is time consuming.

→ Eliminating Redundancy:

We can eliminate redundancy by decomposition.

Example:

Purchases (cid, cname, pid, pname, time)



→ lossless Decomposition (lossless join decomposition):

If we decompose a relation R into R_1, R_2 and R_3 and $R_1 \bowtie R_2 \bowtie R_2 = R$ (Natural join) then the decomposition is called as lossless join decomposition, Else it is lossy.

Example R (A B C)

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

R

decomposed

R_1 (A B)

A	B
1	2
2	2
3	1

R_1

R_2 (B C)

B	C
2	1
2	2
1	2

R_2

here, $R_1 \bowtie R_2$ =

A	B	C	
1	2	1	
1	2	2	→ This was not in original table
2	2	1	
2	2	2	
3	1	2	

Since, $R_1 \bowtie R_2 \supset R$

hence, it is a lossy decomposition

$(R_1 \bowtie R_2 \neq R)$

This didn't worked because, here we are joining using the common attribute B and B is having duplicates, it means that this common attribute is not a key or more specifically super key or it cannot determine a tuple uniquely neither in R_1 nor in R_2. The decomposition would not be lossy if B is the key of in either of the relation.

So, if the common attribute is not a super key of any of the tables R_1 or R_2 than the decomposition is lossy.

Example 2: R(A B C)

From the instance of table R in example 1 the functional dependency which we could derive is

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

$\Rightarrow$ Given relation R(A B C)

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

R

decomposed

R_1(A B)

A	B
1	2
2	2
3	1

R_1

R_2 (A C)

A	C
1	1
2	2
3	2

R_2

$R_1 \bowtie R_2 =$

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

$= R$

$\Rightarrow$ This decomposition is lossless as $R_1 \bowtie R_2 = R$

In table R_1, A is the superkey and A is also a superkey of R_2.

NB

So, if the common attribute in decomposition is a SK of either of the relation (resultant relations) then decomp is lossless.

Example 3: $R(A\,B\,C\,D)$

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

→ Given decomposition $R_1(A\,B\,C)$ and $R_2(C\,D)$ is lossless or not?

⇒ ∵ $C \to D$ ∴ in relation $R_2(C\,D)$ C is the SK. Hence, the condition is satisfied and Therefore, the decomposition is lossless.

→ Given decomposition $R_1(A\,B)$ and $R_2(C\,D)$ is lossless or not?

⇒ Since, there is no common attribute in between R_1 and R_2 and therefore, we cannot join them. Hence, this decomposition is lossy.

→ Given decomposition $R_1(A\,B)$ and $R_2(B\,C)$ is lossless or not?

⇒ It is also lossy because it don't even included the attribute D in any of the relation.

→ Given decomposition $R_1(A\,B)$, $R_2(B\,C)$ and $R_3(C\,D)$ is lossless or not?

⇒ ∵ $B \to C$ ∴ B is SK to R_2 ∴ we can Combine R_1 and R_2

∵ $C \to D$ ∴ C is SK to R_3. Hence, we can combine R_2 with R_3. Hence it is lossless.

NOTE

Algorithm to determine for lossy / lossless decomposition

① R, F

↳ $R_1, R_2 \ldots R_n$ is lossless if

ⓐ $R_1 \cup R_2 \ldots \cup R_n = R$

ⓑ Merging R_i and R_j into R_{ij}

if ⓐ $R_i \cap R_j \neq \phi$

ⓑ $R_i \cap R_j \rightarrow R_i$, or

$R_i \cap R_j \rightarrow R_j$

repeat till we get original R

Example: R(A B C D E G)

$F = \{ AB \rightarrow C, AC \rightarrow B, AD \rightarrow E, B \rightarrow D, BC \rightarrow A, E \rightarrow G \}$

⟶ Given decomposition $R_1(AB)$, $R_2(BC)$, $R_3(ABDE)$ $R_4(EG)$ is lossless or not?

⟹ Given the relations R_1, R_2, R_3 and R_2 union of all attributes are present in R. So, ⓐ condition is satisfied.

$R_1(AB)$
$R_2(BC)$
$R_3(ABDE)$
$R_4(EG)$

(lossless) $R_{13}(ABDE)$ (as AB is SK of R_1) $AB \rightarrow C$

(lossless) $R_{134}(ABDEG)$ (E is SK of R_4) $E \rightarrow G$

(lossy) R_{1234}

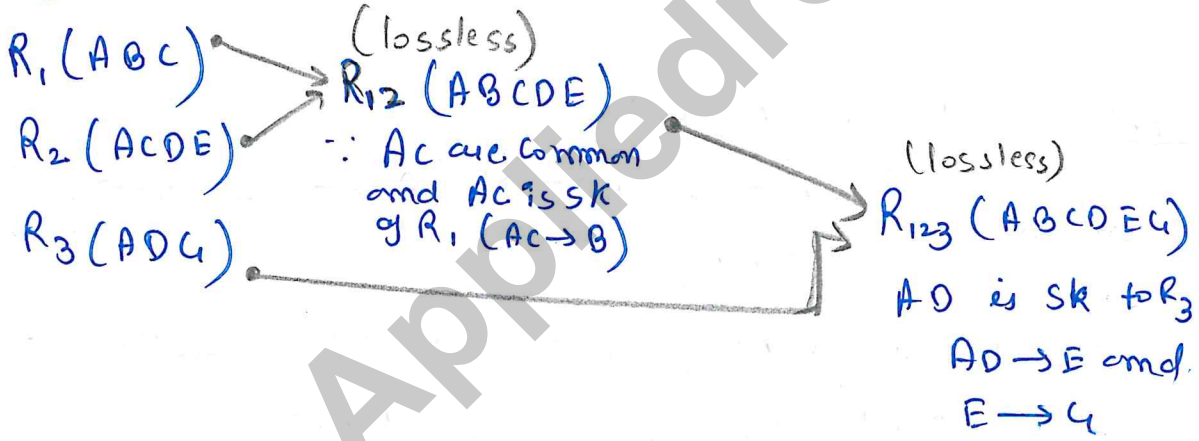
B is neither SK for R_2 nor for R_{134}

Hence, this whole decomposition is lossy.
We can try multiple ways to merge them but they will all return us a lossy decomposition.

⟶ Given decomposition $R_1(ABC), R_2(ACDE), R_3(AD)$ is lossless or not?

⟹

Since, decomposition covered all attributes ∴ first condition is satisfied.
Now to check second condition we need to merge them



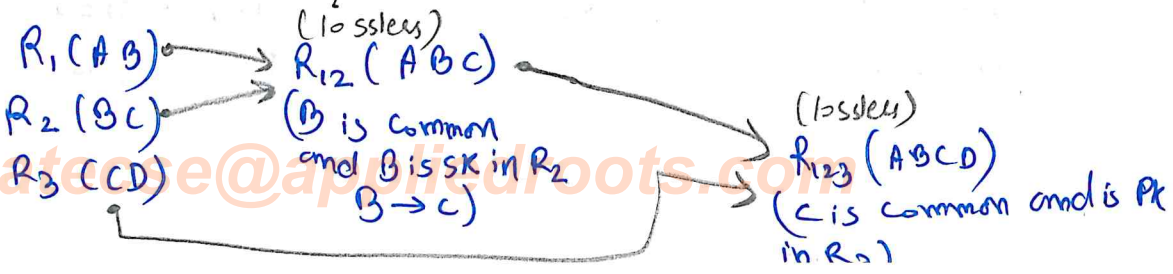
Hence, this decomposition is lossless

~~⟹~~

Example: $R(ABCD)$
$F = \{A \to B, B \to C, C \to D\}$

⟶ Given decomposition $R_1(AB), R_2(BC), R_3(CD)$ is lossless or not?

⟹

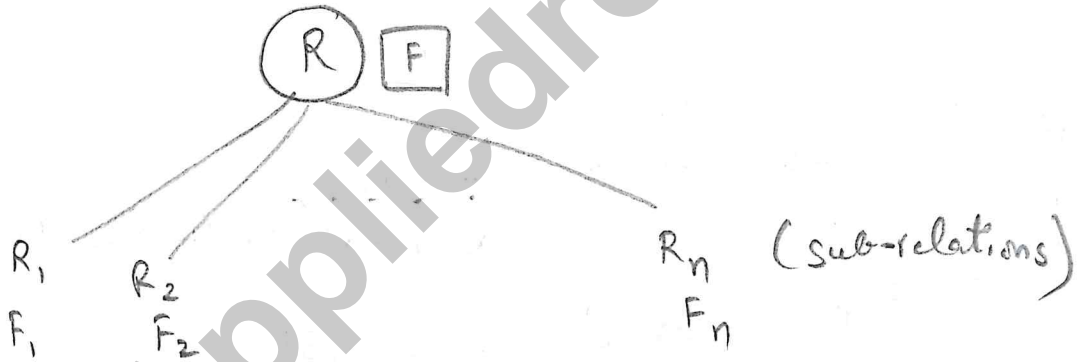


Hence, this decomposition is lossless.

16.7 Dependency preserving decomposition + Solved examples:

→ Dependency preserving decomposition:

When we try to decompose the relation R into $R_1, R_2 \dots R_n$, we also try to decompose the FD set F, into other smaller set of FD's $F_1, F_2 \dots F_n$



So, decomposition is said to be dependency preserving if $F_1 \cup F_2 \dots \cup F_n = F'$ and $F' = F$ i.e two FD set F' and F should be equivalent.

i.e. $(F_1 \cup F_2 \dots \cup F_n)^+ = F^+$

(Recover F only using F_i's)

Example 1: R(A B C D)

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

Given the decomposition of R as $R_1(AB)$, $R_2(BC)$ and $R_3(CD)$ is dependency preserving or not?

⟹ Step 1: Identify all non-trivial FD's using F for each sub relations.

$R_1(AB) \to F_1 = \{A \to B, B \to A\}$

$R_2(BC) \to F_2 = \{B \to C, C \to B\}$

$R_3(CD) \to F_3 = \{C \to D, D \to C\}$

F'

$A \to B$

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

Hence, we can add $B \to A$ (additional FD)

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

it is additional FD in R_2

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

it is additional FD in R_3

Step 2: Determine if $F_1 \cup F_2 \cup F_3 = F$ or not?

Here, we need to check whether we can get add FD's in F' using F or not or viceversa

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

$D \to A$ is not yet covered but we can infer it using transitivity. Hence, we can recover F using F'. It is DP.

Example 2: $R(ABCDEF)$

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

Given decomposition $R_1(AB)$, $R_2(CDE)$, $R_3(EF)$ is DP (dependency preserving) or not?

$\Longrightarrow$ **Step 1:**

$R_1(AB) \to F_1 = \{A \to B, B \to A\}$ $A^+ = A$ and $B^+ = B$ Hence, no FD is valid

$R_2(CDE) \to F_2 = \{C \to D, C \to E, D \to E, CD \to E\}$ $C^+ = \{C, D, E, F\}$ $D^+ = \{D, E, F\}$ $E^+ = \{E, F\}$, $CD^+ = \{C, D, E, F\}$

$R_3(EF) \to F_3 = \{E \to F, F \to E\}$ $E^+ = \{E, F\}$, $F^+ = F$

Step 2: Check $F_1 \cup F_2 \cup F_3 = F$

On checking F and F' we can never get $AB \to C$ and $AB \to D$

$\therefore$ This decomposition is not dependency preserving.

Example 3: $R(ABCDE)$

$F = \{A \to BC, CD \to E, B \to D, E \to A\}$

Given decomposition $R_1(ABC)$ and $R_2(ADE)$ is DP or not?

$\Longrightarrow$ **Step 1:**

$R_1(ABC) \Rightarrow F_1 = \{A \to BC, A \to B, A \to C, AB \to C, C \to A, BC \to A, AC \to B\}$ $C^+ = C$ $BC^+ = \{B, C, D, A\}$

$R_2(ADE) \to F_2 = \{E \to A, A \to E, A \to D, E \to D, AE \to D, AD \to E, DE \to A\}$ $A^+ = \{A, C, D, E\}$

Step 2: $\because A \to BC$ is covered in R_1

and ~~$CD \to E$~~ $E \to A$ is covered in R_2

But for $B \to D$ and $CD \to E$ we need to check.

$B^+_{F_1, F_2} = \{B\}$

(B^+ under F_1 and F_2)

$CD^+_{F_1, F_2} = \{C, D\}$

$\therefore B \to D$ and $CD \to E$ is not preserved

Hence, this decomposition is DP.

Example 4: $R(A\ B\ C\ D\ E\ F\ G\ H\ I)$

$F = \{AB \to C,\ A \to DE,\ B \to F,\ F \to GH,\ D \to IJ\}$

Decomposition is given as $R_1(ABC)$, $R_2(ADE)$, $R_3(BF)$, $R_4(FGH)$, $R_5(DIJ)$ is DP or not?

$\Rightarrow$ Step 1:

$R_1(ABC) \to F_1 = \{AB \to C, \ldots\}$

$R_2(ADE) \to F_2 = \{A \to DE, \ldots\}$

$R_3(BF) \to F_3 = \{B \to F, \ldots\}$

$R_4(FGH) \to F_4 = \{F \to GH, \ldots\}$

$R_5(DIJ) \to F_5 = \{D \to IJ, \ldots\}$

Here, each of the ~~relations are~~ decomposition are done such that for every FD there is a seperate relation. Since, there is no need to look for the additional non-trivial dependency as they are completely preserved. Hence, it is DP. It is by design DP.

Suppose the decomposition is like:

$R_1(ABCDE) \rightarrow F_1 = \{AB \rightarrow C, A \rightarrow DE \cdots\}$

$R_2(BFGH) \rightarrow F_2 = \{B \rightarrow F, F \rightarrow GH \cdots\}$

$R_3(DIJ) \rightarrow F_3 = \{D \rightarrow IJ \cdots\}$

It is also DP.

NORMALIZATION

17.1 Introduction to Normalization + 1NF and 2NF + examples:

* Principled approach to perform decomposition in order to ensure
  - → No or minimum redundancy
  - → lossless join
  - → Dependency preserving
  - → Reduce anamolies

Edgar Codd came up with Principled approach Normalizat

* Covers normal forms like 1NF, 2NF, 3NF, BCNF, 4NF etc.

Note that, every relation in 1NF is in 2NF, every relation in 2NF is also in 3NF and so on. So, every higher normal form implied lower normal form.

1 NF :

Each attribute in a relation is atomic (single valued or not multivalued)

Example 1: Relation R:

Cid	Purchased
i_1	P_1, P_2
i_2	P_2, P_3

This attribute is not atomic and ∴ R is not in 1 NF

decomposing it, so that it could be in 1 NF

Relation R_s':

Cid	Purchased
i_1	P_1
i_1	P_2
i_2	P_2
i_2	P_3

Redundancy

it is in 1NF But here PK is {Cid, Purchased}

One of the problems with 1 NF is that it introduce redundancy. But is the foundation of normal forms as we cannot have multivalued attributes or multiple values in a cell.

Example 2: R(A B C D) where D is multivalued attribute. FD's F = { A → B, B → C }

What are possible relation ~~is~~ which are in 1NF?

⟹ Possible CK are:

∵ D is not in FD's ∴ D should be in every CK.

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

Hence, CK = {AD}. Since, AD is a Candidate key ~~therefore~~ therefore D could not be multivalued. The relation will be

$R^*(A, D, BC)$ where AD is key D will be broken into $R_1(AD)$, $R_2(ABC)$

Here D will be splitted like shown in example 1

→ Partial dependency (PD) and Redundancy:

Partial dependency is given as ⇒

FD: Proper subset of a candidate key → non prime-attribute

Example: R

Cid	Cname	Pid
C_1	n_1	P_1
C_1	n_1	P_2
C_2	n_2	P_2
C_2	n_2	P_3

Redundancy due to partial dependency

Subset of CK

FD, F = {Cid → Cname}

non-prime attribute

here CK = {(Cid, Pid)}

Prime-attribute = {Cid, Pid}

non prime-attribute = {Cname}

The FD given in set F is a Partial dependency.

2 NF:

A relation R is in 2NF iff

① R is in 1 NF (all attributes are atomic)

② R has no partial dependency.

2 NF avoids the redundancy caused due to partial dependency.

Example: R (Cid, Pid, Cname)

FD = { Cid → Cname, Cid, Pid → Cname }

Partial dependency

R is already in 1 NF because all attributes are atomic). Now if we decompose

R (Cid, Pid, Cname)

R1 (Cid, Pid)

F1 = { ~~Cid, Pid →~~ }

(R1 is in 2NF)

CK = {(Cid, Pid)}

Hence, PD

R2 (Cid, Cname)

F2 = { Cid → Cname }

(R2 is in 2NF)

CK = {Cid}

Hence, no PD

Hence, decomposition of R into R1 and R2 is in 2NF and is a lossless ; dependency preserving decomposition.

Example: $R(ABCDEF)$ is in 1 NF

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

Is it is 2NF. If not than decompose.

$\Longrightarrow$ $CK = \{AB\}$ $\because AB^+ = A, B, C, D, E, F$

Prime attribute $= \{A, B\}$

here $B \to E$ and $B \to F$ are Partial dependencies

Therefore we need to decompose it:

$R(ABCDEF)$

$R_1(ABCD)$

$F_1 = \{AB \to C, C \to D\}$

$CK = \{AB\}$

$R_2(BEF)$

$F_2 = \{B \to E, B \to F\}$

$CK = \{B\}$

$\therefore$ All dependency's are preserved and it is a lossless decomposition as common attribute B is SK of R_2. Both R_1 and R_2 are in 2NF as there are no PD.

Example: $R(ABCDEFG)$ is in 1 NF

$FD = \{AB \to C, B \to F, A \to E, E \to G\}$

is R in 2NF? If not than decompose it.

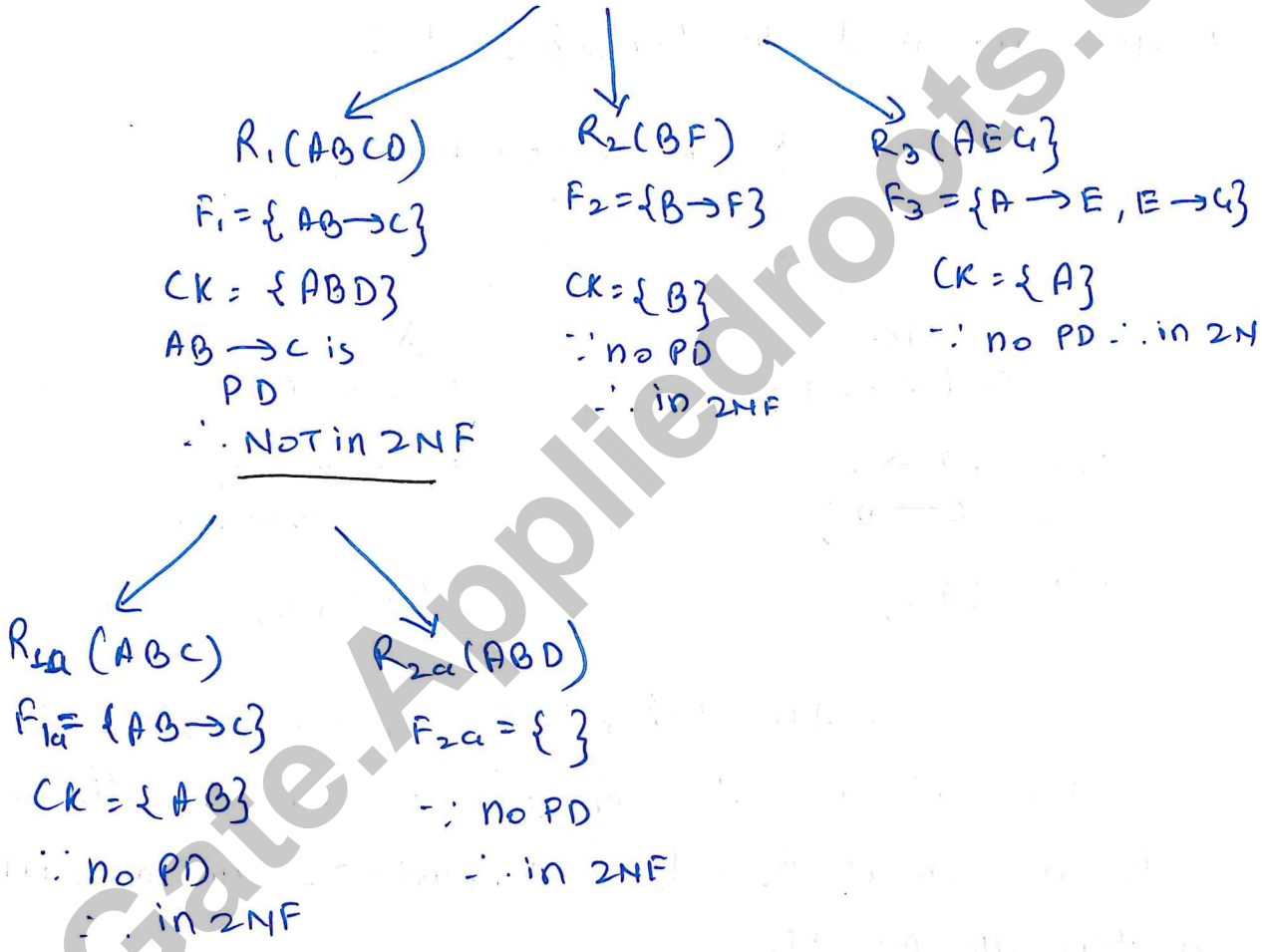
⟹ here, CK = { ABD }

Prime attibute = {A, B, D}

Here, AB → C, B → F, A → E are PD ∴ R is not in 2ND

We need to decompose R:

R (ABCDEFG)



NOTE- 2NF doesnot remove all redundancies. It eliminates only redundancy due to partial dependencies.

17.2 3NF and BCNF:

Tournament winners:

Tournament	Year	Winner	Winner date of birth
Indiana Invitational	1998	Al Fred	21 July 1975
Cleveland open	1999	Bob Albert	28 Sept 1968
Des Moines	1999	Al Fred	21 July 1975
Indiana Invitational	1999	Chip Masterson	14 march 1977

Tournament winners:

Tournament	Year	winner
Indiana Invitational	1998	Al Fred
Cleveland open	1999	Bob Albert
Des Moines	1999	Al Fred
Indiana Invitational	1999	Chip Masterson

Winner Date of birth:

Winner	Date of birth
Chip Masterson	14 march 1977
Al Fred	21 July 1975
Bob Albert	28 Sept 1968

Fig: 34 Wikipedia Example

→ Transitive Dependency:

Considering Tournament as T
Year as Y
Winner as W
Winner date of birth as D

Given, FD set $F = \{TY \rightarrow WD, W \rightarrow D\}$

In the relations given, there is some redundancy in R_3 though it is already in 2NF.

CK: $\{TY\}$, and there is no PD. Hence, it is an 2NF.

Since, prime attributes are = {T, Y}
non-prime attributes are = {W, D}

here $W \rightarrow D$ is like non-prime attribute determining another non prime attribute. The redundancy in R is due to this dependency. Such dependencies are called Transitive dependency.

3NF:

A relation R is in 3NF iff
ⓐ R is in 2NF, and
ⓑ All non-prime attribute must depend on a Key (SK)
'OR' for all non-trivial FD's $X \rightarrow Y$ either X should be a SK or Y is a prime attribute.

Better way to understand

'OR' No non-prime attribute is transitively is determined dependent on any key

'OR' No non-prime attribute depends on other non-prime attribute. such dependencies are called Transitive dependency

$X \rightarrow Y$ (X: SK or SK; Y: Prime attribute)

Example 1: $R(A, B, C, D, E, F)$

$F = \{ AB \rightarrow C, C \rightarrow D, D \rightarrow E, B \rightarrow F \}$

Is R in 3NF? If not decompose it.

$\Longrightarrow$ Given the FD set F, the CK is

$CK = \{ AB \}$

prime-attribute $= \{A, B\}$

Here, $B \rightarrow F$ is a partial dependency

and $C \rightarrow D$, $D \rightarrow E$ is a transitive dependency

Therefore, R is neither in 2NF nor in 3NF.

If the decomposition is

By design it is DP and lossless

- $R_1(ABC) \rightarrow F_1 = \{ AB \rightarrow C \}$
- $R_2(CD) \rightarrow F_2 = \{ C \rightarrow D \}$
- $R_3(DE) \rightarrow F_3 = \{ D \rightarrow E \}$
- $R_4(BF) \rightarrow F_4 = \{ B \rightarrow F \}$

Now they are in 2NF as well as in 3NF

Hence, R_1, R_2, R_3, R_4 are in 3NF. No other decomposition is possible to have it in 3NF.

BCNF:

It stands for Boyce-Codd Normal form. It is a stricter form of 3NF. Given a relation R and FD set F, every FD $X \rightarrow Y$ is in BCNF such that

(a) $X \supseteq Y$ (trivial)

(b) X is a superkey.

Every BCNF is in 3NF. Note that there are very few real world examples of 3NF and not in BCNF.

NOTE- Some computer scientist Beeri and Bernstein in 1979 said in their research papers. Given the following type of ~~BCNF~~ relation and FD

$$R(ABC)$$

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

We cannot obtain BCNF which is lossless, dependency preserving and ~~BCNF~~ in BCNF. ~~A~~ Any one condition will be compromised.

Also it is noted that, having a lossless join is a must / necessary condition while this is not the case with DP. We try to preserve all functional dependency's but it is not a must condition.

Proof:

$$R(ABC), F = \{AB \rightarrow C, C \rightarrow B\}$$

$R_1(AC) = \{\}$
$R_2(BC) = \{C \rightarrow B\}$

} Both are in BCNF

R_1 and R_2 are in BCNF and will have a lossless join as C is SK of R_2. But it could not preserve $AB \rightarrow C$. Hence, we need to compromise with the dependency preserving condition and thus we could not preserve $AB \rightarrow C$

Example: $R(A\ B\ C\ D)$

$F = \{ AB \rightarrow CD\ ,\ D \rightarrow A \}$

Is it in BCNF? If not than decompose it

$\Rightarrow$ $CK = \{ AB, BD \}$

Prime attribute = $\{A, B, D\}$

For the given FD set F, R is in 3NF but not in BCNF because given,

$D \rightarrow A$

↑ not SK — ↑ Prime attribute

} in 3NF but not in BCNF

Decomposition 1:

X

$R_1(BC) \rightarrow F_1 = \{\}$

$R_2(AD) \rightarrow F_2 = \{D \rightarrow A\}$

} Both are in BCNF But they are neither DP nor having lossless join

Decomposition 2:

$R_1(BCD) \rightarrow F_1 = \{\}$

$R_2(AD) \rightarrow F_2 = \{D \rightarrow A\}$

} Both are in BCNF and have lossless join but it is not DP as it could not preserve $AB \rightarrow CD$

17.3 Solved Problems + Properties of Normal Forms:

Example 1: $R(ABCDEFGHIJ)$

$F = \{AB \to C,\ A \to DE,\ B \to F,\ F \to GH,\ D \to IJ\}$

Check whether the given relation and FD set is in 2NF, 3NF, BCNF.

$\Rightarrow$ Candidate Keys = $\{AB\}$

Prime attributes = $\{A, B\}$

** Check for 2NF ** (No PD is part of key $\xrightarrow{X}$ non prime attribute)

Here, $A \to DE$ and $B \to F$ are partial dependencies

∴ We will decompose R such that, taking the dependency which are violating 2NF and then put / make a separate relation for them.

$R_1(ADEIJ) \longrightarrow F_1 = \{A \to DE, D \to IJ\}$ // A is key

$R_2(BFGH) \longrightarrow F_2 = \{B \to F, F \to GH\}$ // B is key

$R_3(ABC) \longrightarrow F_3 = \{AB \to C\}$ // AB is key

Since, there is not PD left ∴ R_1, R_2, R_3 are in 2NF now.

* * Check for 3NF * * (No Transitive dependency ie)

$X \longrightarrow Y$
SK or PA

Among R_1, R_2, R_3, the relation R_1 and R_2 are having transitive dependency

$D \rightarrow IJ$ and $F \rightarrow GH$ are TD

The decomposition will be:

By design in BCNF

$R_1(ADE) \rightarrow F_1 = \{A \rightarrow DE\}$ 3NF/BCNF

$R_2(BF) \rightarrow F_2 = \{B \rightarrow F\}$ 3NF/BCNF

$R_3(FGH) \rightarrow F_3 = \{F \rightarrow GH\}$ 3NF/BCNF

$R_4(DIJ) \rightarrow F_4 = \{D \rightarrow IJ\}$ 3NF/BCNF

$R_5(ABC) \rightarrow F_5 = \{AB \rightarrow C\}$ 3NF/BCNF

While decomposing it into 3NF, we also got it into BCNF

Hence, R_1, R_2, R_3, R_4, R_5 are DP, lossless and BCNF decomposition

Example 2: $R(ABCDEFGH)$

$F = \{AB \rightarrow C, AC \rightarrow B, AD \rightarrow E, B \rightarrow D, BC \rightarrow A, E \rightarrow G\}$

Check R for 2NF, 3NF and BCNF.

$\Rightarrow$ $CK = \{FHAB, FHAC, FHBC\}$

Prime attribute $= \{A, B, C, F, H\}$

2nd

NOTE: In practice people directly convert relations into BCNF rather than converting them into 2NF and 3NF first.

Because everything which is in BCNF will automatically is in 2NF and 3NF.

** Check for 2NF **

In FD set F, $B \rightarrow D$ is the partial dependency

∴ Decomposition will be:

$R_1(BD) \longrightarrow F_1 = \{B \rightarrow D\}$

~~$R_2(ABCEFGH) \longrightarrow F_2 = \{AB \rightarrow C, AC \rightarrow B, BC \rightarrow A, E \rightarrow G\}$~~

But we cannot put $AD \rightarrow E$

∴ we cannot have this R_2

$R_2(ABCFH) \longrightarrow F_2 = \{AB \rightarrow C, AC \rightarrow B, BC \rightarrow A\}$

$R_3(ADEG) \longrightarrow F_3 = \{AD \rightarrow E, E \rightarrow G\}$

∴ Now R_1, R_2, R_3 is in 2NF

* Check for 3NF and BCNF **

In the above decomposition R_3 voilates 3NF

∴ R_1 and R_2 will be same

$R_3(ADE) \longrightarrow F_3 = \{AD \rightarrow E\}$ 3NF/BCNF

$R_4(EG) \longrightarrow F_4 = \{E \rightarrow G\}$ 3NF/BCNF

Now R_1, R_2, R_3, R_4 are in 3NF

But R_2 is not in BCNF because CK = FHAB, FHBC, FHAC

Special case

∴ $R_{2a}(ABC) \longrightarrow F_{2a} = \{AB \rightarrow C, AC \rightarrow B, BC \rightarrow A\}$

$R_{2b}(ABFH/ACFH/BCFH) \longrightarrow F_{2b} = \{\}$

Now, $R_1, R_{2a}, R_{2b}, R_3, R_4$ are in BCNF.

Example 3: $R(ABCDE)$

$F = \{ A \to BC, CD \to E, B \to D, E \to A \}$

Check R for 3NF and BCNF.

$\Rightarrow$ $Ck = \{ A, E, CD, BC \}$

Prime attributes = $\{ A, B, C, D, E \}$

$R_1 (BD) \to F_1 = \{ B \to D \}$

$R_2 (ABC) \to F_2 = \{ A \to BC \}$

$R_3 (EA) \to F_3 = \{ E \to A \}$

} 3NF / BCNF But $CD \to E$ is not preserved

Hence R_1, R_2, R_3 are in BCNF and have lossless join but is not dependency preserving.

Properties of Normal forms:

* If every ck is made of a single attribute (simple key) then Relation R and FD set F is in 2NF

Ex- $R(ABCDE)$

$F = \{ A \to B, B \to C, C \to D, D \to E, C \to A \}$

$CK = \{ A, B, C \}$ // Since all of them are simple keys therefore, it is bydefault in 2NF

Because, we can check there is no PD. (Part of ck $\not\to$ non prime)

* If every attribute is a prime attribute then relation R, and FD set F is in 3NF

* If relation R is in 3NF and all Candidate Keys are simple than relation R and FD set F is in BCNF.
(if every CK is simple than every prime attribute are CK and R is in 3NF which means $X \rightarrow Y$ is satisfied (if X is a SK than BCNF and if X is not SK ~~then~~ will not arrive)

* A relation with only 2 attributes (Binary relation) is always in BCNF.

Ex- Possible cases for R(AB) are:

① { } ; BCNF
② {A → B} ; BCNF
③ {B → A} ; BCNF
④ {A → B, B → A} ; BCNF
⑤ {AB → AB} ; BCNF

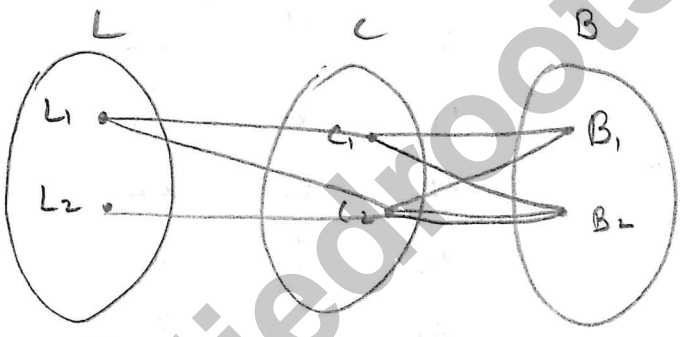
17.4 Multi-valued dependencies and 4NF:

R:

Lecture (L)	Course (C)	Book (B)
L_1	C_1 / C_2	B_1 / B_2
L_2	C_2	B_2

Fig: 35 Wikipedia example

As we can see in fig 35 Course (C) and Book (B) are multivalued attribute. Hence, it is not in 1NF



It could be written as

R':

L	C	B
L_1	C_1	B_1
L_1	C_1	B_2
L_1	C_2	B_1
L_1	C_2	B_2
L_2	C_2	B_2

4NF address this type of redundancy.

1 NF of R

Fig: 36

So, now R' is now in 1NF. But here no single attribute could be the key.

So, CK = {LCB}

There are no non-trivial FD in R'. ie F = { }.

Therefore, R' is already in BCNF. But there is lot of redundancy in R' even though it is in BCNF.

Note that, this redundancy occur because of multivalued attribute and ~~it~~ this type of redundancy is addressed by 4NF.

This type of redundancy could not be addressed by Functional dependencies, therefore, the concept of Multivalued dependency come into the picture (MVD). They are called so because the origin of the problem is multivalued attributes.

→ Multivalued Dependency (MVD):

X	Y	Z
x_1	y_1	z_1
x_1	y_1	z_2
x_1	y_2	z_1
x_1	y_2	z_2

MVD is represented either using '$\rightarrow\!\!\rightarrow$' or '$\twoheadrightarrow$'.

For a relation $R(X\,Y\,Z)$, $Z = R-(X \cup Y)$ than X multi-determins Y ($X \twoheadrightarrow Y$).

X multideterminies Y if we have $Z = R-(X \cup Y)$, and

① $t_1.x = t_2.x = t_3.x = t_4.x$

② $t_1.y_1 = t_2.y$ and $t_3.y = t_4.y$

③ $t_1.z = t_3.z$ and $t_2.z = t_4.z$

→ Properties of MVD:

* If $X \twoheadrightarrow Y$ then $X \twoheadrightarrow Z$ where $Z = R - (X \cup Y)$
(Complement property)

* Trivial MVD: ⓐ if $X \supseteq Y$ then $X \twoheadrightarrow Y$
ⓑ if $X \cup Y = R$ then $X \twoheadrightarrow Y$
* We need atleast 3 attribute for a MVD to exist.
* We need atleast 2 attribute for non-trivial FD to exist

* If $X \twoheadrightarrow Y$ and $Z \supseteq W$ then, $(Z \twoheadrightarrow W)$
$XZ \twoheadrightarrow YW$ (Augmentation)

* If $X \twoheadrightarrow Y$ and $Y \twoheadrightarrow Z$ then $X \twoheadrightarrow (Z - Y)$
(Transitivity)

* If ~~$X \to Y$~~ $X \to Y$ then $X \twoheadrightarrow Y$ (every FD is MVD but vice versa is not true)
(Replication)

* MVD do not obey:
⇒ if $X \twoheadrightarrow YZ$ then $X \twoheadrightarrow Y$ and $X \twoheadrightarrow Z$ ✗
(it is not obeyed in MVD but it is obeyed in FD)

⇒ if $X \twoheadrightarrow Y$ and $X \twoheadrightarrow Z \Rightarrow X \twoheadrightarrow YZ$ ✗
(FD obey this ie $X \to Y$ and $X \to Z \Rightarrow X \to YZ$)

→ 4NF:

4NF is extension to BCNF. A relation R is in 4NF iff (set of FD)

① R, F is in BCNF (SK's are determined using FD's F)

② All non-trivial MVD's of $X \twoheadrightarrow Y$. X should be SK and it should be determined using FD and not using MVD's

On considering relation given in fig: 36, ie

R

L	C	B
L_1	C_1	B_1
L_1	C_1	B_2
L_1	C_2	B_1
L_1	C_2	B_2
L_2	C_2	B_2

R_1

L	C
L_1	C_1
L_1	C_2
L_2	C_2

R_2

L	B
L_1	B_1
L_1	B_2
L_2	B_2

here, $F = \{\}$

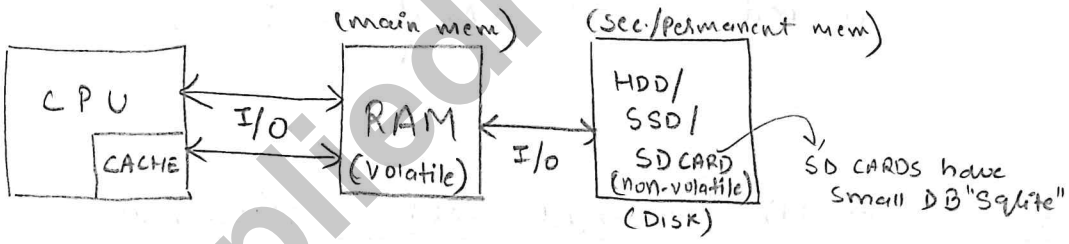
$MVD = \{L \twoheadrightarrow C, L \twoheadrightarrow B\}$

We can decompose R into 2 R_1 and R_2 and both R_1 and R_2 do not have any non-trivial MVD's

TRANSACTION AND CONCURRENCY CONTROL

18.1 Model of a computer for transaction and Concurrency:

→ model of computer:



SQL query gets executed in CPU on database which was stored in Disk.

Failures possible while executing an SQL query:

→ Power failure
→ Disk failure
→ Process failure

→ Transaction:

Transaction is a logical set of operations to perform a task.

Example: let us consider a bank transaction example:

① Read balance for A (→ Disk I/O operation)

② If A.balance ≥ 500 then A.balance = A.balance + 500
~~Write A~~ (It will be performed in CPU and updated in RAM)

③ Write A.balance (RAM or Disk operation)

④ Read B.balance (→ Disk I/O)

⑤ B.balance = B.balance + 500 (→ RAM operation)

⑥ Write B.balance (→ RAM or disk I/O)

⑦ Commit (Commit stores data to non volatile storage)

Each of these are logical operation.

Note - read data A ⇒ Corresponds to R(A)
Write data A ⇒ Corresponds to W(A)
Commit ⇒ Corresponds to C

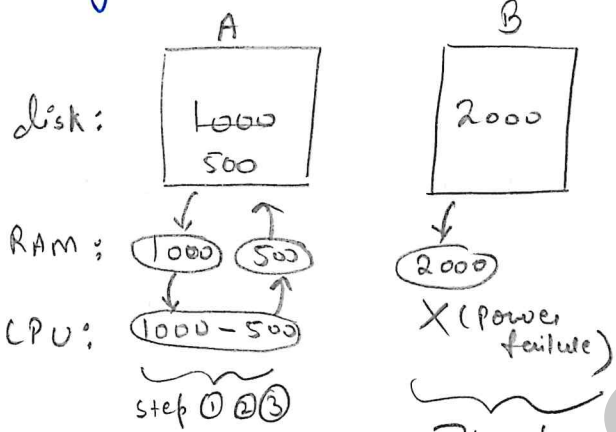
18.2 ACID Properties and Concurrency:

→ ACID Properties:

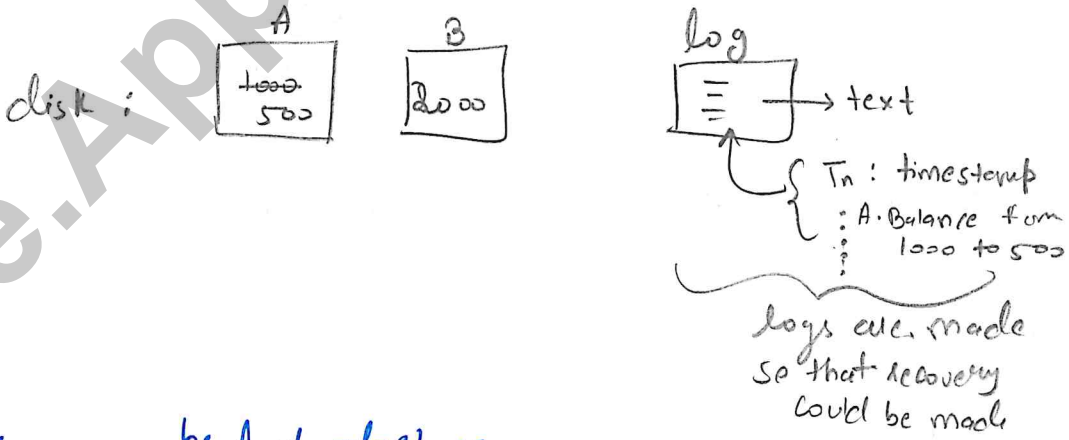
A → Atomicity
C → Consistency
I → Isolation
D → Durability.

Atomicity:

A transaction must execute all of ~~its~~ the operations and commit successfully OR none of the operations should be executed (Should not have partial completion of operation)



If there is partial execution, then recovery should be done. One of the recovery technique used is log-based recovery



So, during any kind of failure
- Read log file
- Observe partial completion
- Revert the transaction

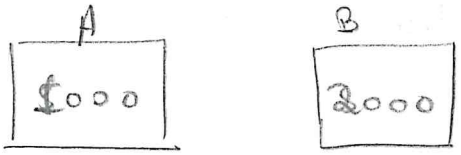
So, Atomicity is a must property which tells a transaction should be either completely done or not done.

Consistency:

The database state / state of system must be always consistent.

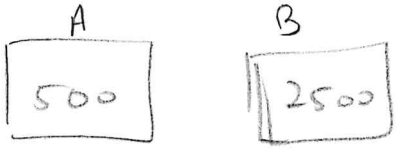
Ex- Transfer 500 from Acc. A to B (intra bank transfer)

Before transaction →



A + B = 3000 (before transaction)

After transaction →



A + B = 3000 (After transaction)

If the system follow the following than that it will always remain in consistent state

Consistent —follows→ Atomicity, Isolation and Durability —remains→ Consistent

We can enforce Consistency using CHECK in sql

NOTE-

Concurrency:

In real world DB, tons of Query are asked at some time (Concurrently), so Concurrent transactions are those transaction which occurs simultaneously.

Ex- T1: Transfer 10$ from A to B — T1: sub 10$ from A, T1: add 10$ to B

T2: Transfer 10$ from B to A — T2: sub 10$ from B, T2: add 10$ to A

Both are 2 diff-sql answer.

Consider these 2 sql queries as two different process in a system if the process are performed in serial ie S_1 followed by S_2 or vice versa then these will be called as sequential or serial schedule (A schedule is order of execution of operations).

So $S_1 \rightarrow S_2$ or $S_2 \rightarrow S_1$ are sequential schedule

Now suppose the transaction sequence is as follows

T_1: sub 10$ from A
T_2: sub 10$ from B
T_1: add 10$ to B
failure or crash → T_2: add 10$ to A

} This is called concurrent or interleaved process or transaction in case of sql

Goal of an OS: Higher throughput (complete as many transactions as possible in given or lesser time)

Isolation:

All transactions will be carried out and executed as if it is the only transaction in the system (though there are n no. of transactions taking place simultaneously)

Given 3 transactions T_1, T_2, T_3 with n, m and k operations respectively, number of concurrent schedules will be given as

T_1 : n ops (T_1: o_1, T_1: o_2, ..., T_1: o_n)

T_2 : m ops

T_3 : k ops

schedules ?

$\Rightarrow$ time ↓ 1 → 2 → 3 → ... → n+m+k →

} These are n+m+k slots in a schedule

So, steps will be

① pick n slots from n+m+k slots

$^{n+m+k}C_n$

now slots left are: m+k

② pick m slots from m+k slots

$^{m+k}C_m$

③ pick remaining k slots

$^{k}C_k$

So total no. of schedules are = $^{n+m+k}C_n \cdot {}^{m+k}C_m \cdot {}^{k}C_k$

$$= \frac{(n+m+k)!}{n!\,m!\,k!}$$

total no. of serial schedule = (no. of schedules) !

If no. of transactions given are T than total no. of

Concurrent schedules are = $\frac{(n+m+k)!}{n!\,m!\,k!} - T!$

So, Isolation says, ensure that DB state would be same if each transaction was executed in isolation and sequentially without interleaving. (concurrent schedule execution should be same as any of the serial schedule)

* Durability:

A DB should be able to recover from any case of failure. OR' Once a transaction is committed its effect must be stored in a non-volatile memory which itself must recoverable from failure.

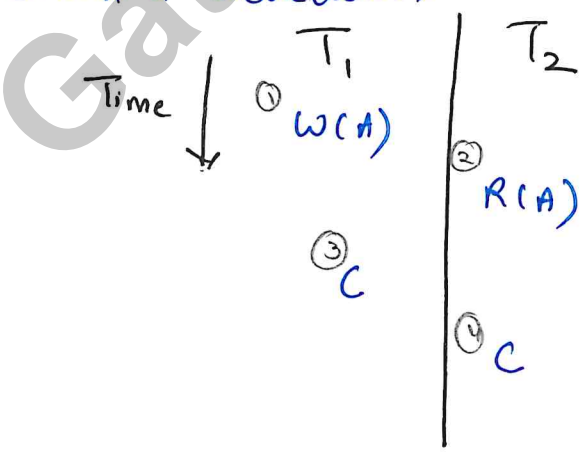
18.3 Problems due to concurrency:

Objective: higher throughput

Limitation: lead to inconsistent schedule.

* Write-Read (WR) Problem / Dirty read / uncommitted read:

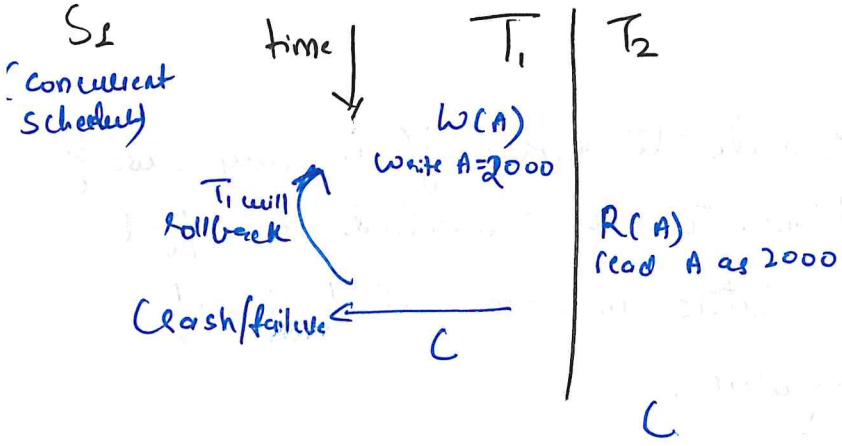
Given a schedule, $W_1(A)$, $R_1(A)$, C_1, C_2, represented as:



W(A) → write(A)
R(A) → Read(A)
C → commit

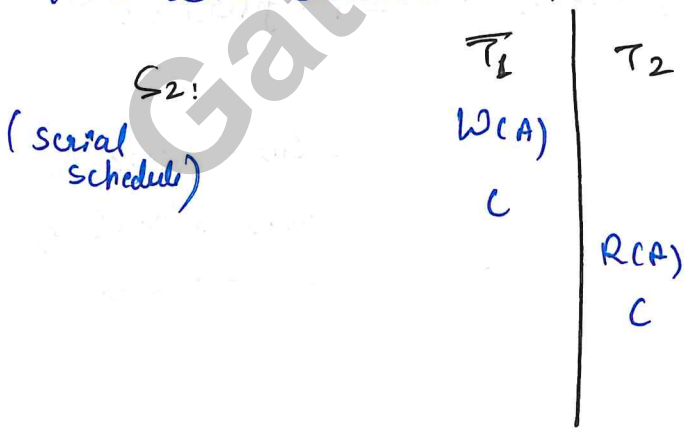
This is called WR problem because we are reading something which is not yet committed.

Suppose, T_1 fails after the execution of T_2 ② operation. So, when T_1 crashes, it will roll back (with the help of log based recovery)



∴ T_2 has already read the initial value and it will continue further with commit operation, this is called dirty read as it is wrong value which T_2 has read (because it is uncommitted read)

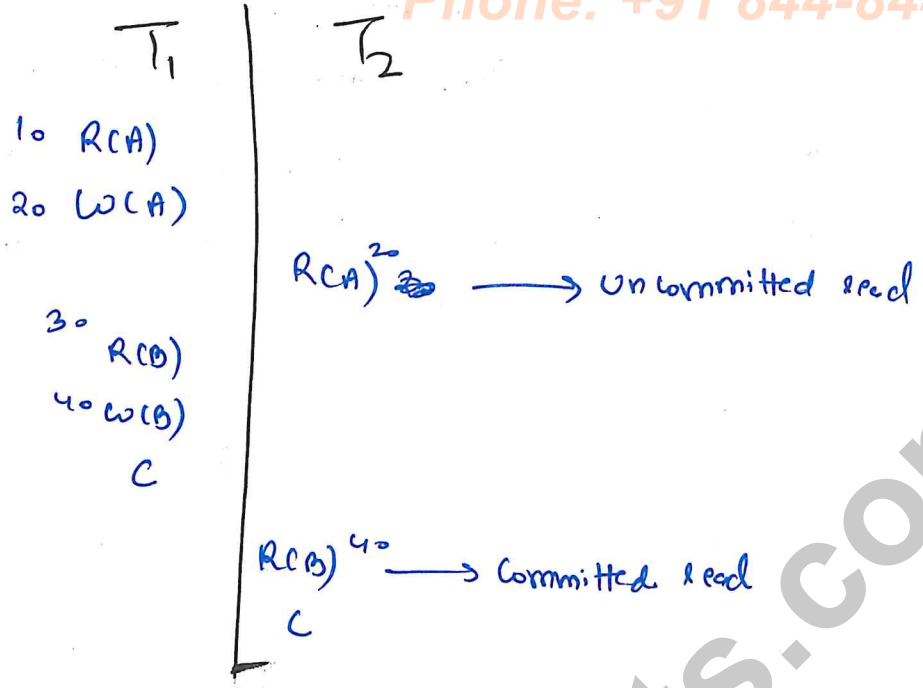
WR-problem may or may not create issues/inconsistency. Imagine if no crash happened than it will be equivalent to a serial schedule ($S_1 \Leftrightarrow S_2$).



S_1 is said to be serializable or consistent.

Example:

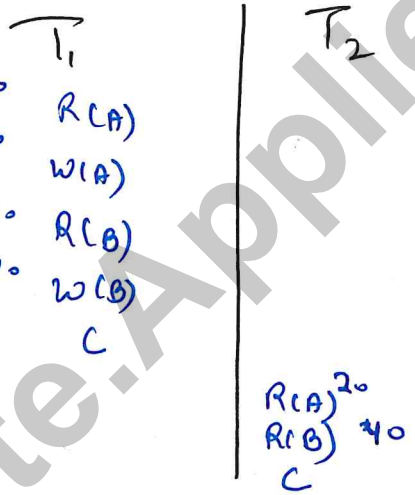
S_1:



If there is no crash, final value of A: 2o
B: 4o

In this case it is equivalent to a serial schedule

S_2: (Serial schedule)



final A: 2o
B: 4o

$\therefore S_1 \approx S_2$

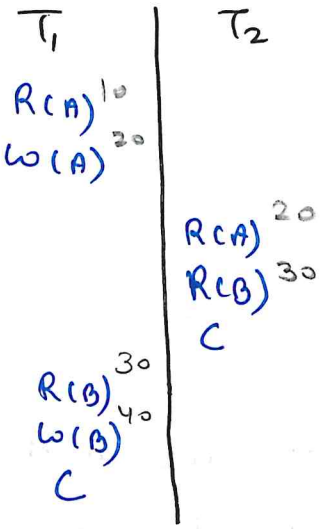
hence, S_1 is consistent schedule.

(Here, we have assumed that there is no crash)

Note: If commit is not clearly mentioned after all operations are done, there is a default commit.

Example -

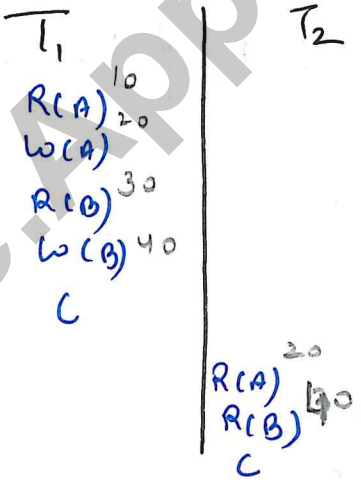
S_1:



If no crash happen then final values A : 20
B : 40

Let's check whether it is equivalent to serial schedule or not (if no crash happen)

S_2:



final values A : 20
B : 40
, The values finally we get are same

But the values read by T_2 finally are different.

Hence $S_1 \neq S_2$. Because even though final values are same but T_2 is reading wrong values.

Lets compare it some other serial schedule

S_3:

T_1	T_2
	10 R(A)
	30 R(B)
	C
10	
20 R(A)	
30 W(A)	
R(B)	
40 W(B)	
C	

final value of A: 20
B: 40

But still read of T_2 is not same as that of S_1 hence $S_1 \neq S_3$. S_2 and S_3 are only two possible serial schedule and $S_1 \neq S_2, S_2$

$\therefore$ S_1 is inconsistent schedule.

* Read-write problems (RW) / Unrepeatable reads:

S_1:

T_1	T_2
10 R(A)	
	R(A) 10
	W(A) 20
	commit
20 R(A)	
30 W(A)	
commit	

2 diff values by reading same variable A

Here, read (A) are performed twice in T_1 and both gives the diff value due to write operation performed by T_2 This problem is called unrepeatable read problem

Because we read once and then repeated the reads and values are changed as there is read followed by write. (no consistency in the values that same transaction is reading)

Example- In flight reservation system, suppose 2 person are trying to book the same ticket. Suppose P_1 reads the price and tries to book the ticket while in between P_2 has booked the ticket. As soon as P_1 reaches the final payment option, it is showing no tickets are available.

There is a special case of read-write problem called Phantom read problem. In this ~~row~~ new rows are added or deleted in between.

T_1	T_2
#emp >50k$ #rows obtained = 100	
	add new rows commit.
#emp>50k$ #rows obtained = 105	

* Write-Write problem (WW) / overwriting uncommitted data:

S_1:

	T_1	T_2
10	W(A)	
20		W(B)
30	W(B) ← overwriting uncommitted data	
	C	
20		W(A)
		C

NOTE-
Blind writes:- no read before writing.

final value A: 20
B: 30

Let S_2 be a serial schedule: (Suppose that no crash happened in S_1)

S_2:

T_1	T_2
10	
30 W(A)	
W(B)	
C	
	W(B) 20
	W(A) 20
	C

Final values of A: 20
B: 20

Here, final values of S_1 and S_2 do not match. Hence $S_1 \neq S_2$.

Let S_3 be another serial schedule:

S_3:

T_1	T_2
	W(B) 20
	W(A) 20
	C
10 W(A)	
30 W(B)	
C	

Final values of A: 10
B: 30

So, $S_1 \neq S_2 \neq S_3$.

$\therefore$ S_1 is inconsistent schedule.

Note that w-w problem may or maynot create inconsistency

Example

S_1:

	T_1	T_2
10	W(A)	
20		W(A)
30	W(B)	
	C	
40		W(B)
		C

final value of A: 20
B: 40

S_2:

	T_1	T_2
10		
30	W(A)	
	W(B)	
	C	
20		W(A)
40		W(B)
		C

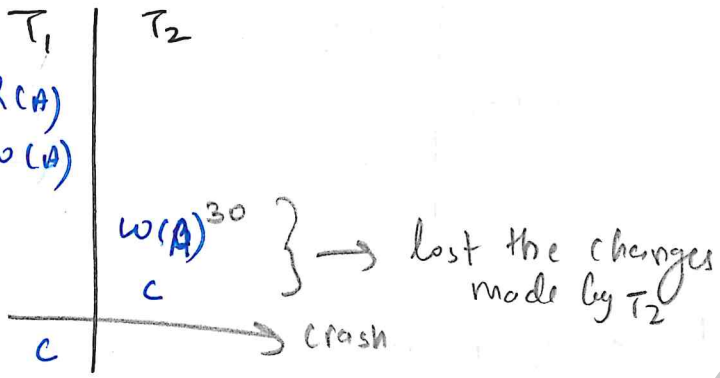
final value of A: 20
B: 40

So, here $S_1 \approx S_2$. So WW may or may not create inconsistency.

Note: If a schedule say S_1 is consistent then there is no problem. Problem arises when we have concurrent schedule which is inconsistent or schedule crashes.

Case 1: When there is crash in w-w conflict. This problem is called as lost-update (ww-rollback)

S_1:



Logs: T_1: A: 10 → 20

T_2: A: 20 → 30

T_2: C

~~T~~ crash (∵ T_1 has not committed yet ∴ we will rollback T_1)

If T_1 is rolled back then all the updates made by T_2 will get lost (~~this~~ lost the changes made by T_2 even though T_2 didn't crashed)

This problem is called lost update problem.

Note - ww - issue + inconsistent → lost update in case of crash + rollback.

18.4 Recoverability of schedules:

For the given concurrent schedules, we have 2 important properties that they should satisfy.

- Recoverability
- Serializability

Recoverability states that: If a transaction is Aborted (ended unexpectedly) or some crash happened and the transaction is rolled back that we should be able to reach to the initial (sensible) state. ie the schedule should be recoverable. (We want our Schedule to be crash proof)

Serializability is concerned to isolation. When a Concurrent schedule is equivalent to serial schedule then we say that concurrent schedule is serializable.

↔ Irrecoverable (or) schedules:

S_1:

T_1	T_2
10 R(A)	
20 W(A)	
	R(A) 20 → uncommitted / dirty read (WR Problem)
	W(A) 30
	commit
abort	
Rollback or Abort	

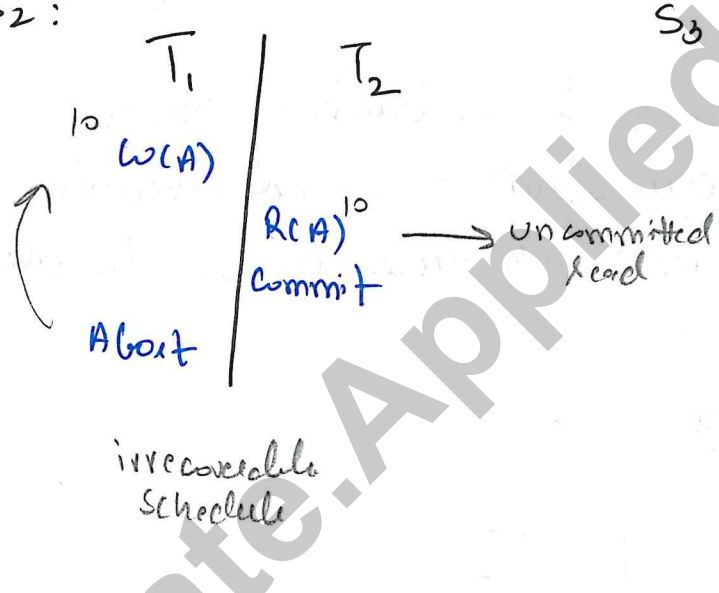
t_2 is already committed before the abort of T_1

This schedule S_1 is not recoverable because T_2 has committed the values and now it cannot be undone. So the decisions made here are ~~can~~ cannot be undone as it has been committed. If the commit of T_2 is placed after the abort of T_1 then there would be no issues.

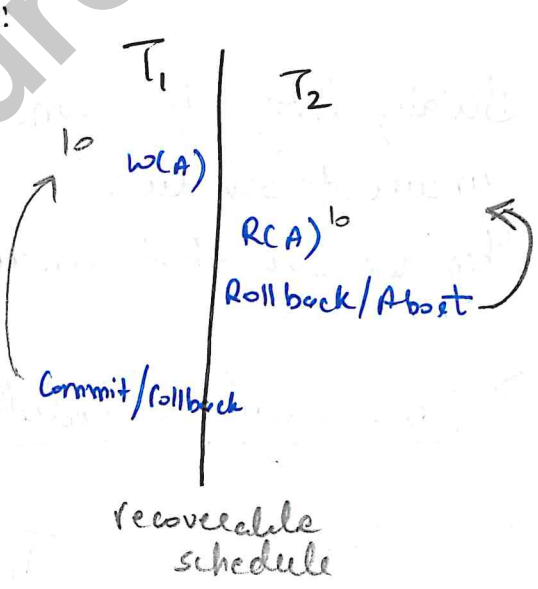
So, Rolling back a committed transaction is irrecoverable.

Example –

S_2:



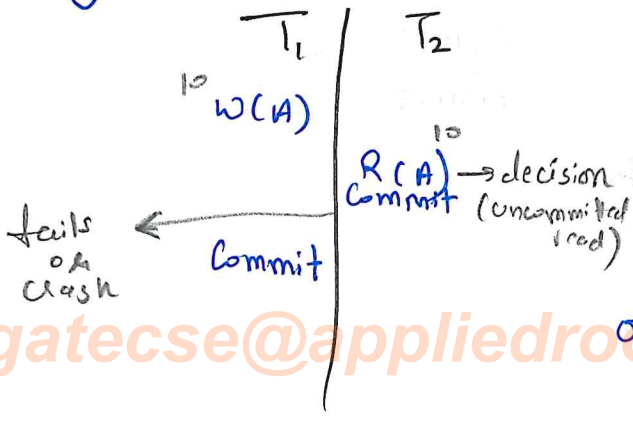
S_3:



→ Defendence between Transaction:

Case 1:

S_1: initially A = 5

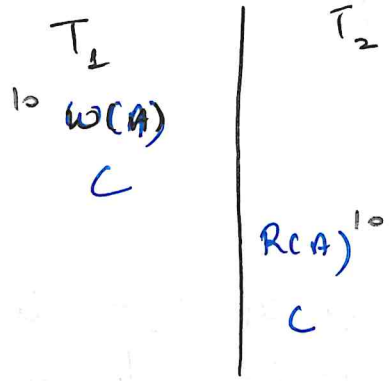


In this case T_2 depends on T_1 because decision made by T_2 is either valid or invalid based on whether T_1 will rollback or not.

So, in S_1 validity of transaction 2 (T_2) decision is based on T_1 success and failure. That is why T_2 depends on T_1.

Case 2:

S_2:



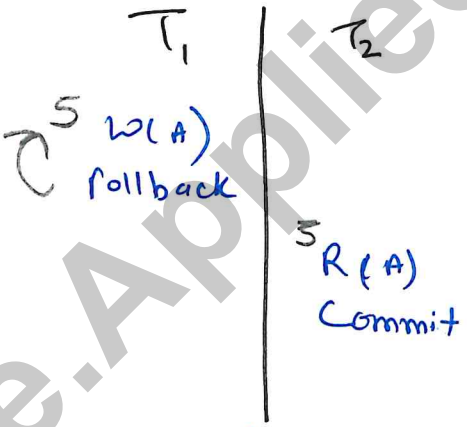
T_2 has read the committed value.

The failure or validity of value in T_2 doesn't depends on T_1 in S_2 because T_1 has already committed

In this case T_2 doesnot depend on T_1.

Case 3:

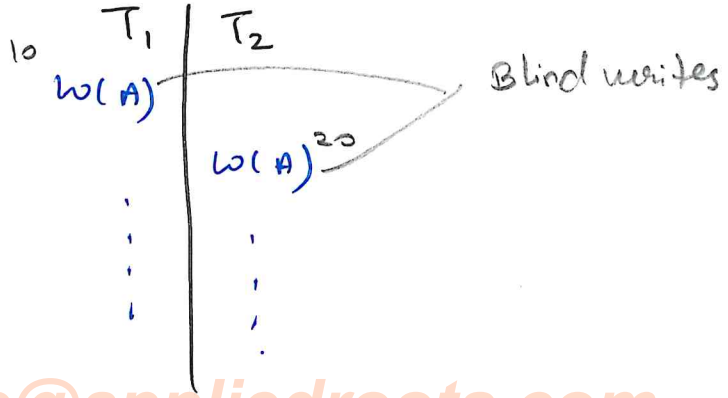
S_3:



here also T_2 doesnot depend on T_1

Case 4:

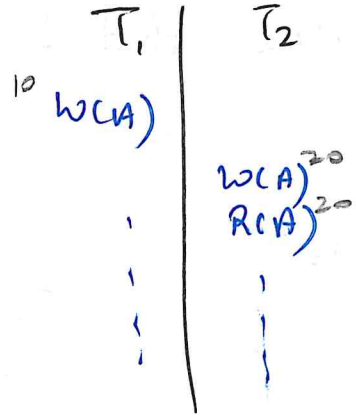
S_4:



here, T_2 is not dependent on T_1 as T_2 is not reading any value written by T_1 hence no [illegible] decision is made by T_2.

Case 5 :

S_5 :

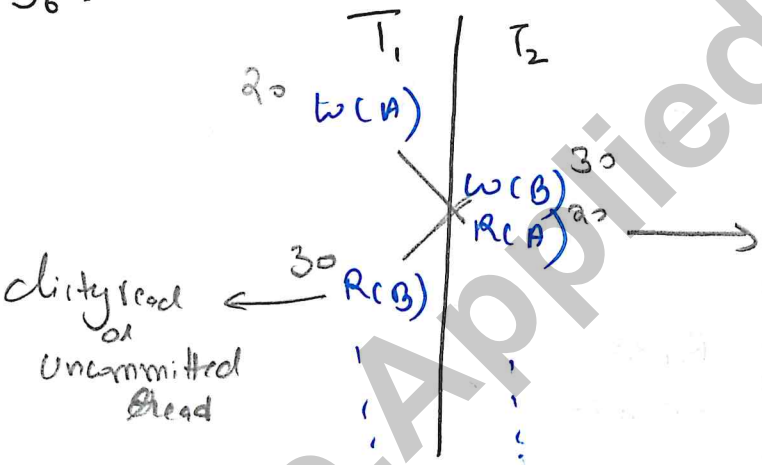


→ This is not an uncommitted read because what we are reading here is written in the same transaction.

T_2 doesn't depend on T_1 as $R_2(A)$ is not a uncommitted read, it has read the value written in T_2 itself

Case 6 :

S_6 :



→ dirty read / uncommitted read

T_2 is dependent on T_1 and T_1 is also dependent on T_2.

→ How to ensure recoverability of schedules?

If T_2 depends on T_1 then T_2 should commit only after T_1 commits or rollback. If this condition is satisfied that it is guaranteed to have a recoverable schedule.

S_1:

T_1	T_2
W(A)	
	R(A)
Commit	
	Commit

Here, T_2 is dependent on T_1 and T_2 has committed after the T_1 has committed ∴ commits are ordered. Therefore, S_1 is recoverable schedule

NOTE- A recoverable schedule may have WR (dirty read) problem.

Example:

S_2:

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

Is S_2 recoverable?

⇒ here T_2 depends on T_1

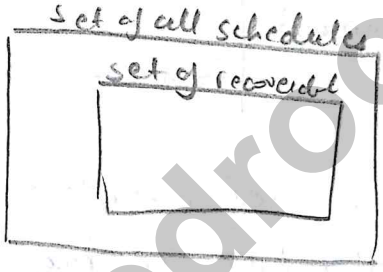
⇓

S_2 is recoverable iff C_2 (commit by T_2) happens after C_1

∵ T_2 committed after T_1 ∴ S_2 is recoverable.

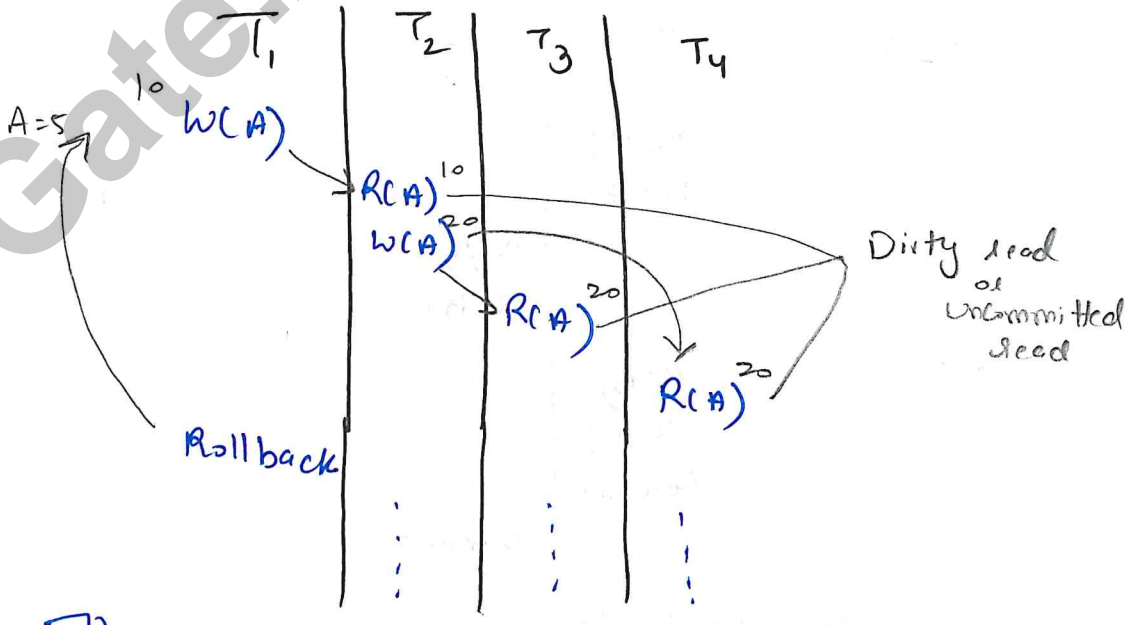
NOTE -

Set of all schedules ⊇ set of recoverable schedules



→ Cascading Rollbacks

S_1 : initially A = 5



The schedule S_1 is recoverable all rollbacks are ordered as no schedule has committed

But, here rollback of T_1 will result in rollback of T_2. Rollback of T_2 will result in rollback of T_3 and so on. (cascading rollbacks). It is recoverable but it causes other problems like lower throughput and longer waittime.

T_2, T_3, T_4 doesn't any mistake but they are dependent on T_1 ∴ they have to rollback.

We should avoid the cascading rollbacks.

→ How to avoid Cascading Aborts (ACA) / Cascadeless:

Cascading aborts are happening primarily because of dirty reads. So, in order to avoid that these should not ~~dirt~~ be presence of uncommitted reads. ie disallow a transaction from reading uncommitted changes from another transaction.

S_2:

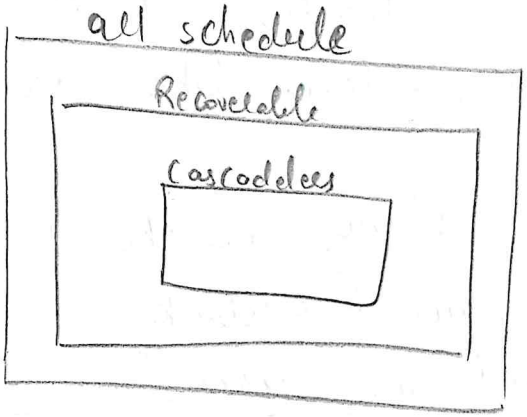
T_1	T_2
R(A)	
	R(A)
	W(A)
R(B)	
	W(B)
	C/R
R(B) ← Committed read	
C/R	

} no uncommitted or dirty read

S_2 is both cascadeless and recoverable schedule.

NOTE -

Set of recoverable schedule ⊃ Set of cascadeless schedule



S_3:

T_1	T_2
W(A)	
	R(B)
	W(A) ⟶ WW problem
	C
C	

S_3 is cascadeless and recoverable but it still have WW problem.

Cascadeless guarantees that there will be no WR problem or dirty problems, other problems may exist.

⟶ Strict schedules:

If T_1 performs W(A) then T_2 is not allowed to R(A) or W(A) till T_1 commits or rollbacks. (ie No WR and No WW)

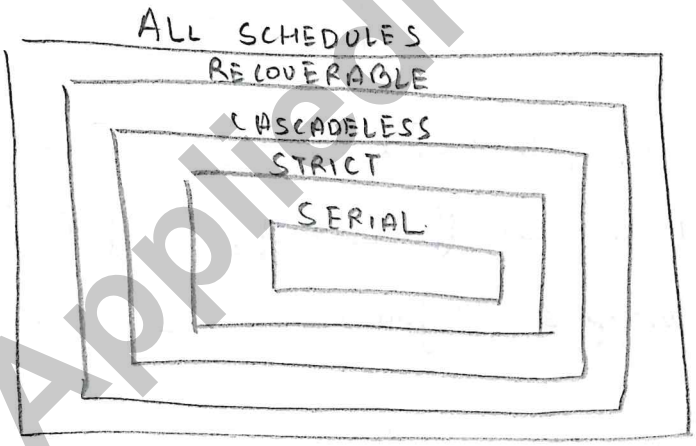
S_1:

T_1	T_2
	R(A)
W(A)	
C	
	R(A)
	W(A)
	C

S_1 is strict schedule (it is both Cascadeless as well as recoverable)

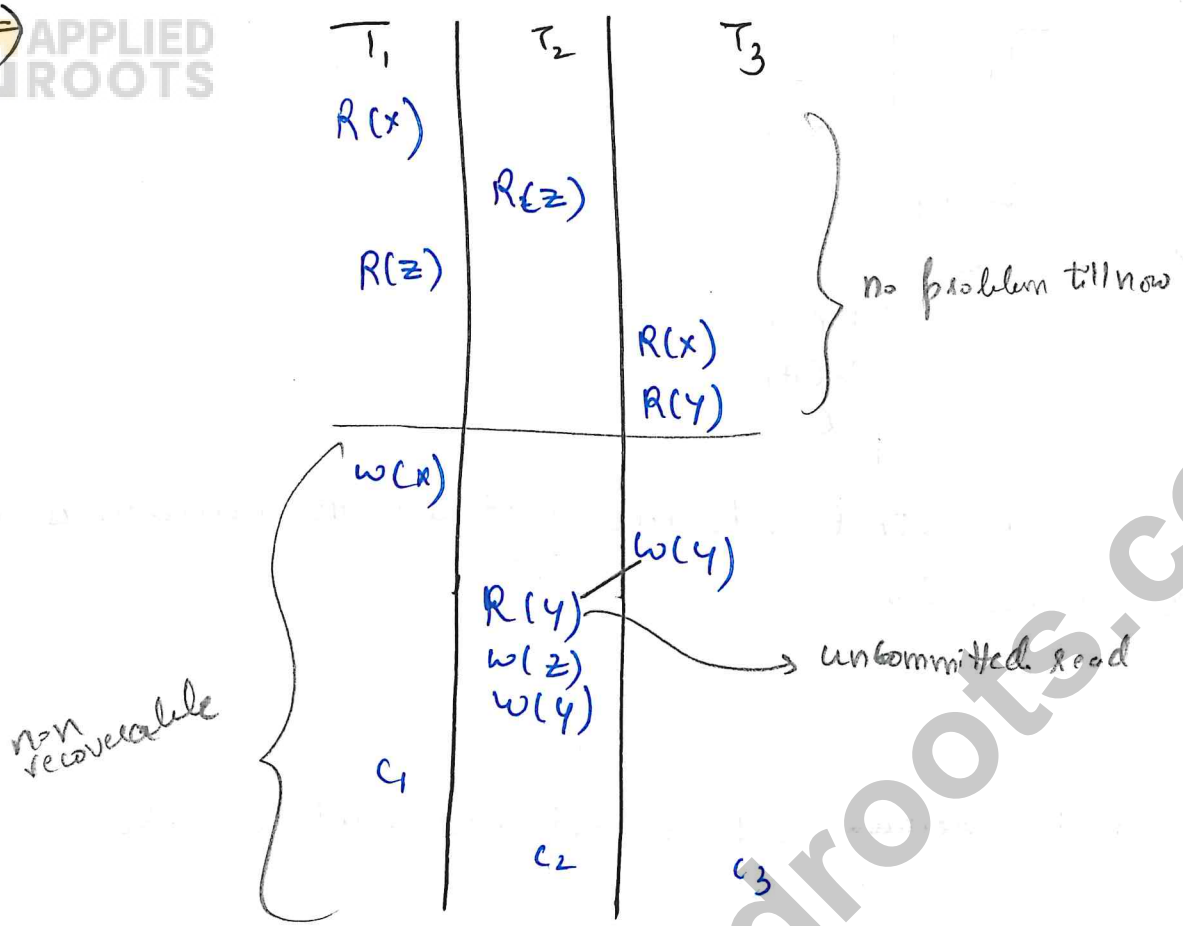
NOTE -

Complete ~~critical~~ set of schedule will be like:



(Q1) S_1 : $R_1(x)$, $R_2(z)$, $R_1(z)$, $R_3(x)$, $R_3(y)$, $W_1(x)$, $W_3(y)$, $R_2(y)$, $W_2(z)$, $W_2(y)$, C_1, C_2, C_3.

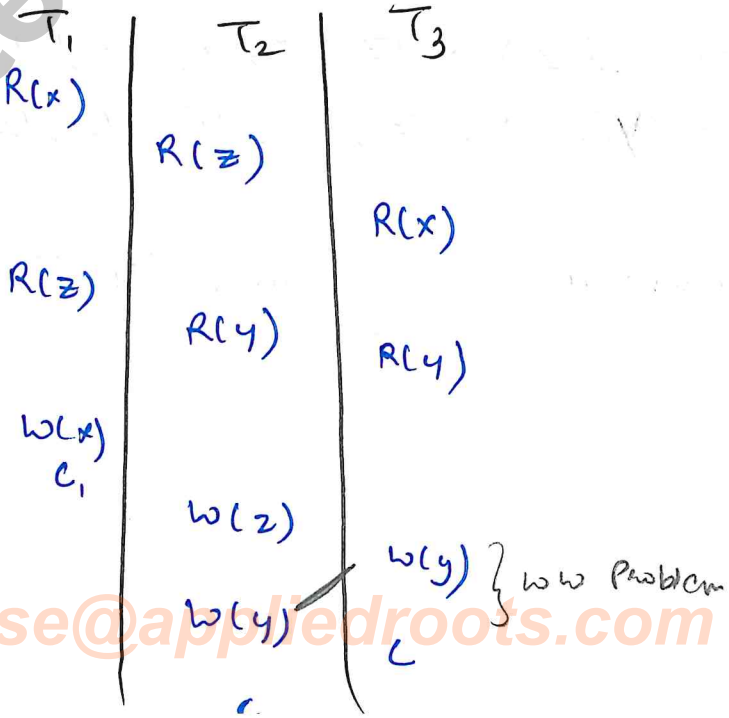
S_1 is the schedule along with its time order.



Due to uncommitted read in T_2, T_2 depends T_3. So S_1 to be recoverable if C_2 happens after C_3. But since C_2 happens before C_3 in S_1 i.e. S_1 is not recoverable

(Q2) Is S_1 is recoverable, cascadeless, strict or not)

S_1:



⇒ Since, there are no dependency ∴ S_2 is recoverable (no dirty read) but there are low problem.

∵ No dependency → S_2 is recoverable

No dirty reads → S_2 is cascadeless

∵ we have low issue → NOT STRICT.

18.5 Serializability of schedules – Conflict serializability:

A concurrent schedule to be serializable should be equivalent to any of the serial schedule.

→ How to determine serializability?

- → Conflict serializability
- → View serializability

Before getting into conflict serializability we need to get to know some few terminologies.

* Non-conflicting pair of operations:

On swapping a pair of operations, if the resulting schedule is equivalent to the original schedule than those pair of operation are called non-conflicting pair of operations.

Ex- initially A=5

S_1:

T_1	T_2
R(A) 5	
	R(A) 5
:	:

swap →

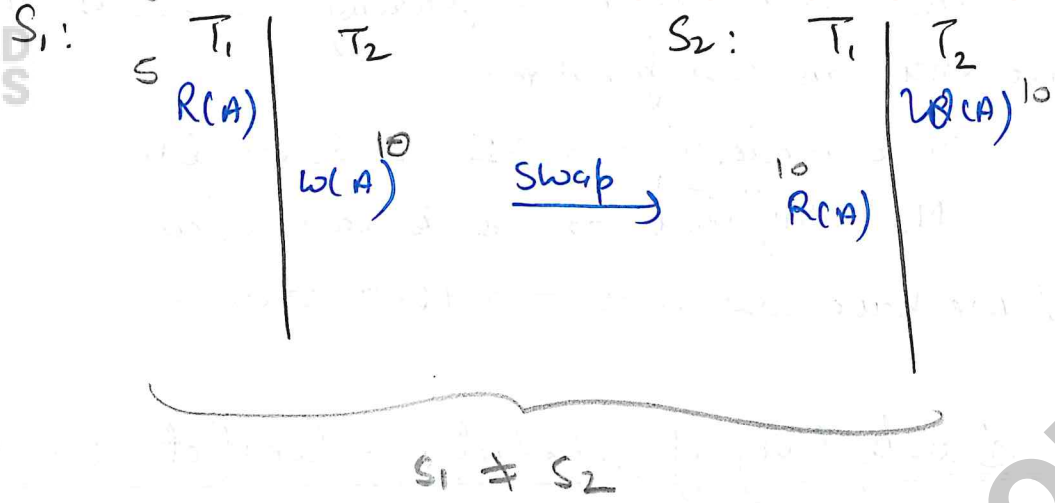
S_2:

T_1	T_2
	R(A) 5
R(A) 5	
:	:

non-conflicting pair of ops

$S_1 \approx S_2$ such pair of ops are called non-conflicting operations

Ex-2 initially A = 5



$S_1 \neq S_2$

$\because S_1 \neq S_2$ $\therefore$ the pair of ops given is a conflicting pair of operations.

→ Conflicting pair of operation:

A pair of operation is conflicting, if:

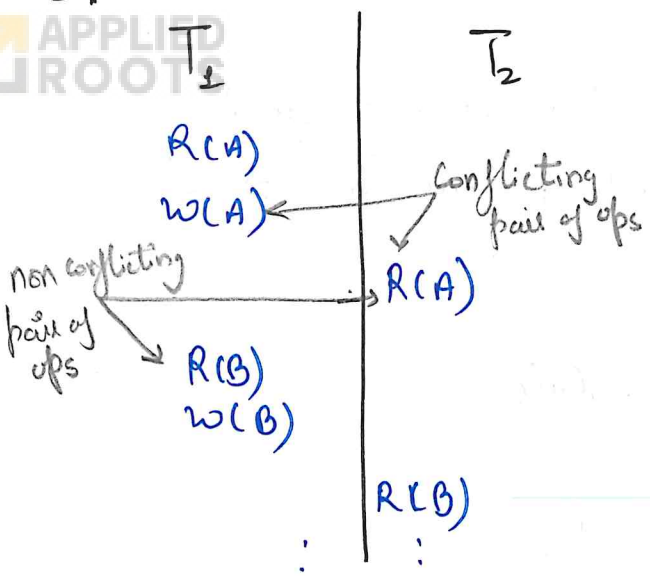
* Both operations should not belong to the same transaction
* Both operations must be on the same data
* Atleast one of the operation should be "write"

If ~~any~~ these 3 conditions are satisfied then the resultant pair of operation is a conflicting pair of operation.

→ Conflict- equivalent schedules:

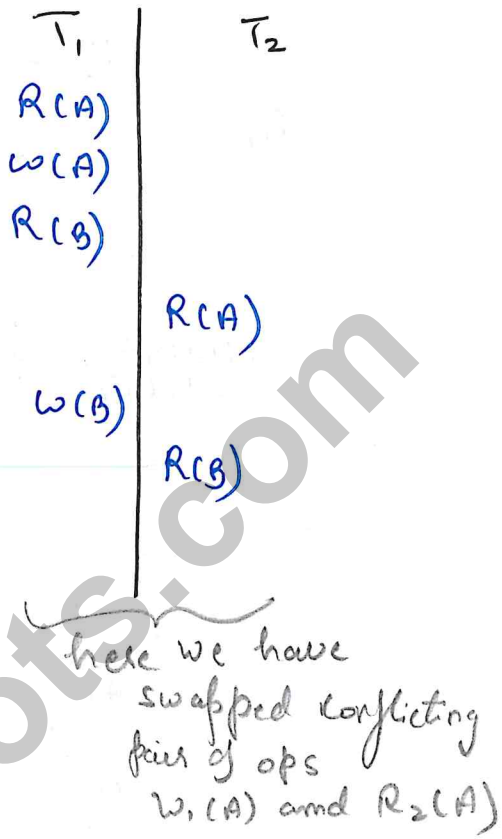
A schedule S_1 is conflict equivalent to S_2 if S_1 results in S_2 from swapping non conflicting pairs. Let S_1 and S_2 be:

S_1:



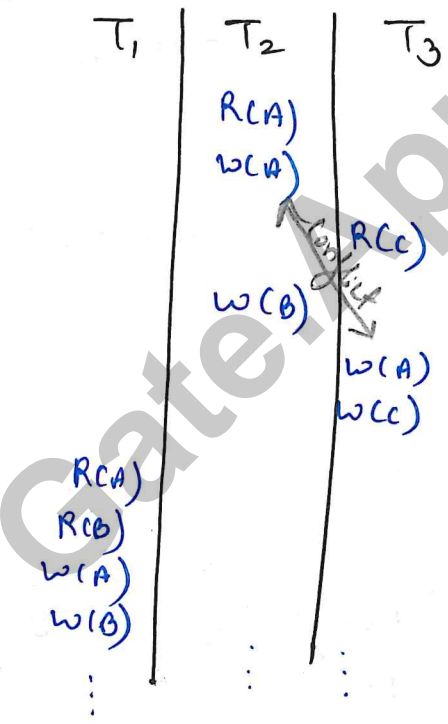
$\approx_c$

S_2:

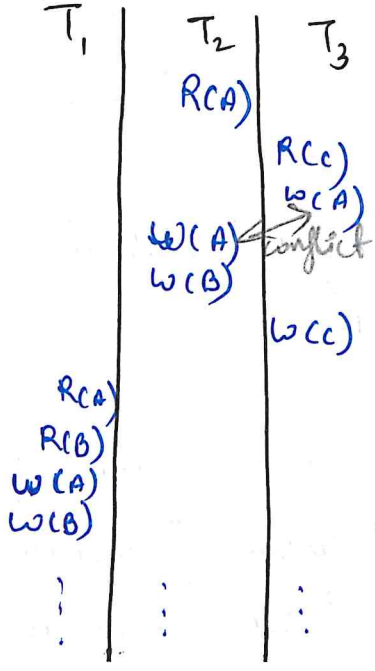


(Q1) Determine whether S_1 and S_2 are conflict equivalent or not.

⇒ S_1:



S_2:



S_1 and S_2 are not conflict equivalent because of $W_2(A)$ and $W_3(A)$

→ Conflict Serializable schedule: (CSS)

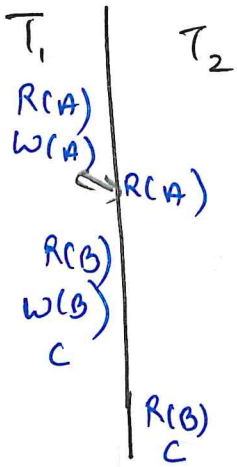
A schedule S is CSS if there exist a serial schedule S' such that S is conflict equivalent to S'

ie

$S \approx_c S'$

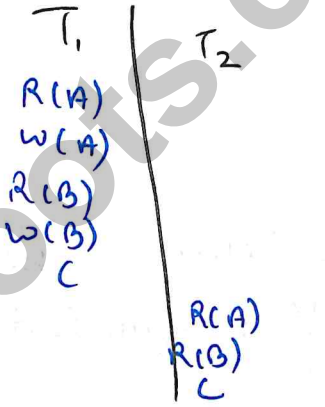
Concurrent schedule ← S ; S' → serial schedule

S :



$\approx_c$

S' :



CSS

$T_1 \rightarrow T_2$

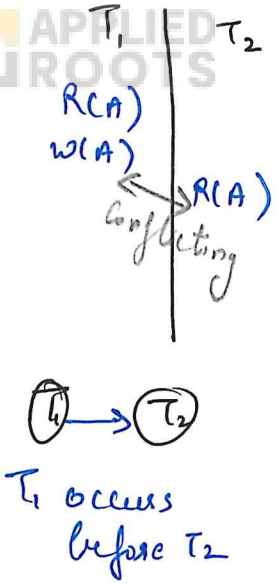
Simpler method to determine CSS is Precedence Graph.

The steps to draw precedence Graph is:

[1] Every transaction is a vertex

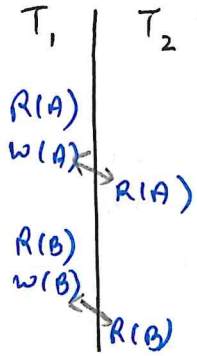
[2] Insert / create a directed edge between $T_i \rightarrow T_j$ if ∃ a conflicting pair and T_i proceeds T_j

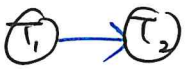
S_1:



T_1 occurs before T_2

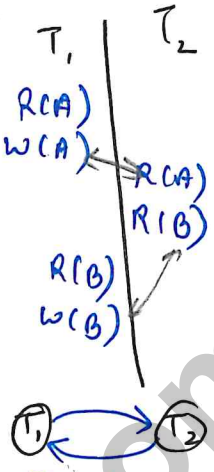
S_2:





T_1 occurs before T_2

S_3:



It forms a cycle because T_1 and T_2 are dependent on each other.

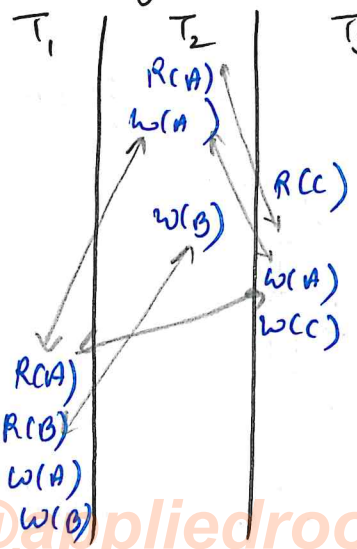
With the help of the precedence graph we can check whether a schedule is CSS or not:

If there is a cycle in precedence graph than that schedule is not CSS while vice versa is true. ie if there is no cycle for a given schedule than it is CSS.

∴ S_1 and S_2 are CSS while S_3 is not CSS.

Using precedence graph we can easily determine whether given schedule is CSS or not.

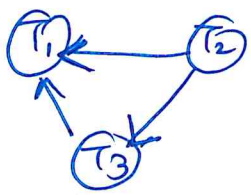
(Q 1) Determine whether given schedule is CSS or not?



⇒

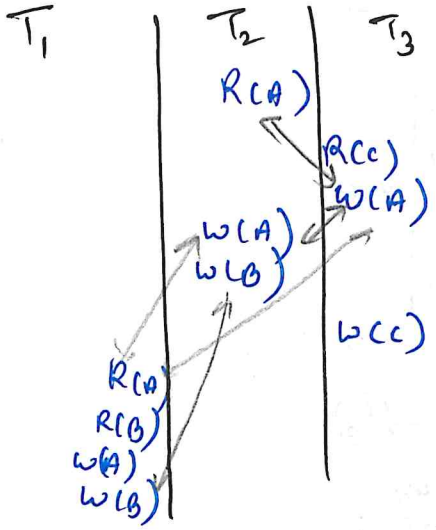
So, T_3 and T_1 depends on T_2 and T_1 depends on T_3 as well

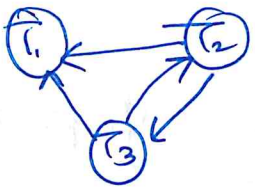
∴



Hence, it is CSS as there is no cycle

(Q2) Is it CSS or not?





∵ there is cycle ∴ it is not CSS.

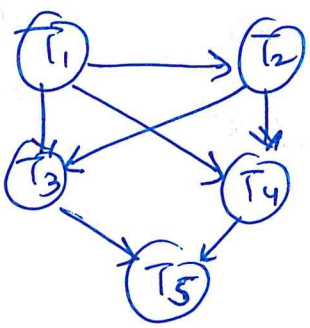
~~In previous question~~
Trace

→ How to get equivalent serial schedule: (Topological sort)

↳ it gives the serial schedule

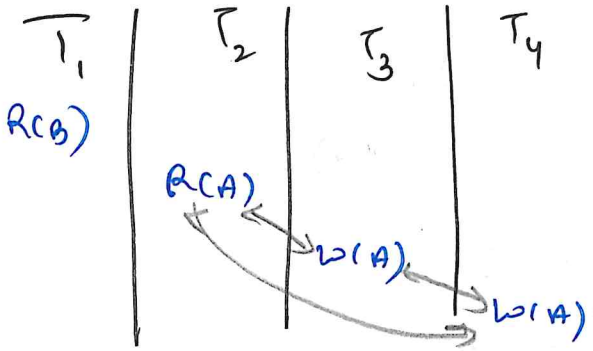
(1) Start with T_i whose indegree is zero.
(2) Remove T_i and all edges connecting from T_i
(3) Repeat 2 and 1

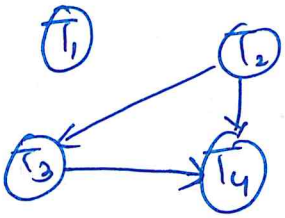
Ex- If the Graph (Precedence graph) is given as



On apply topological sorting 2 serial schedules we will obtain are:- $T_1 \to T_2 \to T_3 \to T_4 \to T_5$, $T_1 \to T_2 \to T_4 \to T_3 \to T_5$

(Q3) Is it CSS or not?





∴ there is no cycle

∴ it is CSS

It is equivalent to serial schedule $T_2 \rightarrow T_3 \rightarrow T_4$.

T_1 could be placed anywhere

∴ We have 4 serial schedule to which this schedule is equivalent to.

$T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4$

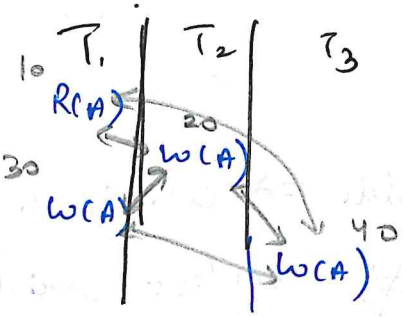
$T_2 \rightarrow T_1 \rightarrow T_3 \rightarrow T_4$

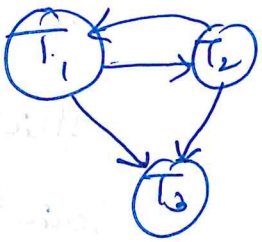
$T_2 \rightarrow T_3 \rightarrow T_1 \rightarrow T_4$

$T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_1$

(Q4) Is it CSS ?

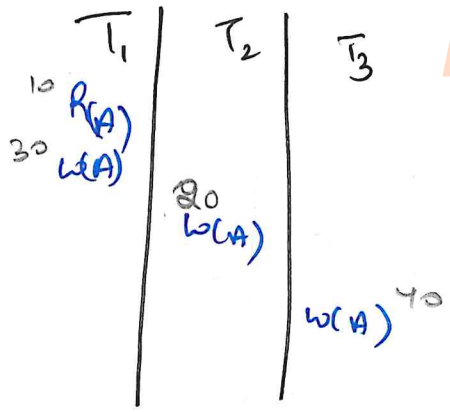
Sol:





∴ there is a cycle ∴ it is CSS and hence, it is not equivalent to any serial schedule.

S_2 :



$S_1 \not\equiv_c S_2$

but $S_1 \approx S_2$ because for every read operation value read is the same and final value written is also same. So, S_1 is $\not\equiv_c$ equivalent to S_2 (S_2 is serial schedule) but S_1 is not conflict equivalent to S_2.

NOTE -

If schedule is conflict serializable than it is serializable (ie it is having an equivalent serial schedule) but every serial schedule / serializable schedule may not be conflict serializable.

The schedule S_1 and S_2 are not conflict serializable though S_2 is serial schedule.

So, CSS is a sufficient condition for serializability NOT a necessary condition.

(If precedence graph is acyclic then CSS $\Rightarrow$ serializable else, NOT CSS but may / may not be serializable.

Conflict serializable $\Rightarrow$ Serializable

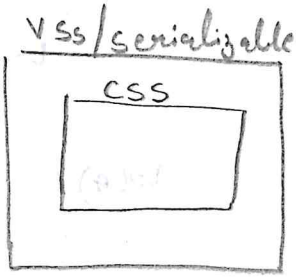
Serializable $\not\Rightarrow$ conflict serializable

18.6 View Serializability : (VSS)

If a schedule is view serializable then the schedule is serializable else it is not serializable.

View serializable $\Leftrightarrow$ Serializable

Vss is both neccessary and sufficient condition for serializability.



→ View equivalence :

(Represented as $\approx_v$)

$S_1 \approx_v S_2$ only if,

[1] Initial reads must match.

ex- A=10 (initially)

S_1:

T_1	T_2
10 R(A)	
	R(A) 10

$\approx_v$

S_2:

T_1	T_2
	R(A) 10
10 R(A)	

(Initial reads are same)

ex - initially A=10, B=20

S_1:

T_1	T_2	T_3
10 R(A)		
20 R(B)		
	30 W(B)	
		40 W(B)

$\approx_v$

S_2:

T_1	T_2	T_3
	30 W(B)	
10 R(A)		
30 R(B)		
		40 W(B)

here, ~~the~~ initial reads do not match, as R(B) in S_2 read diff from R(B) in S_1

[2] Write-Read (WR) must match.

Ex- initially A = 10

S_1:

T_1	T_2	T_3
20 W(A)		
	30 W(A)	
		R(A) 30
		W(A) 40

S_2:

T_1	T_2	T_3
20 W(A)		
	30 W(A)	
		20 R(A)
		40 W(A)

S_3:

T_1	T_2	T_3
	30 W(A)	
		30 R(A)
20 W(A)		
		40 W(A)

On S_1, $R_3(A)$ read 30 while in S_2, $R_3(A)$ read 20

$\therefore S_1 \not\equiv_V S_2$

On S_1 and S_3, the W-R operation ($W_2(A) - R_3(A)$) is not changed $\because$ both the $R_3(A)$ has read same value 30

$\therefore S_1 \approx_V S_3$

On S_2 and S_3, the W-R operation is changed

$\therefore S_2 \not\equiv_V S_3$

[3] Final writes must match for each variable.

Ex initially, A = 10, B = 20

S_1:

T_1	T_2	T_3
30 W(A)		
40 W(B)		
	50 W(A)	
		60 W(A)
		70 W(B)

final A = 60, B = 70

$\not\equiv_V$

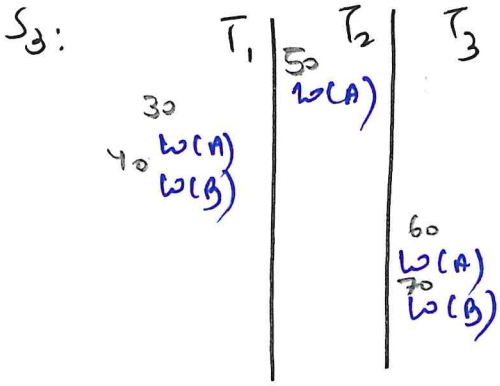
S_2:

T_1	T_2	T_3
30 W(A)		
		60 W(A)
40 W(B)		
	50 W(A)	
		70 W(B)

final A = 50, B = 70

Since, final writes of A and B is not same for S_2 and S_2 ∴ they are not view equivalent.

Suppose we have S_3 as:



here, final write is done by T_3 (same as S_1). ∴ A=70 and same for B ∴ final writes are same

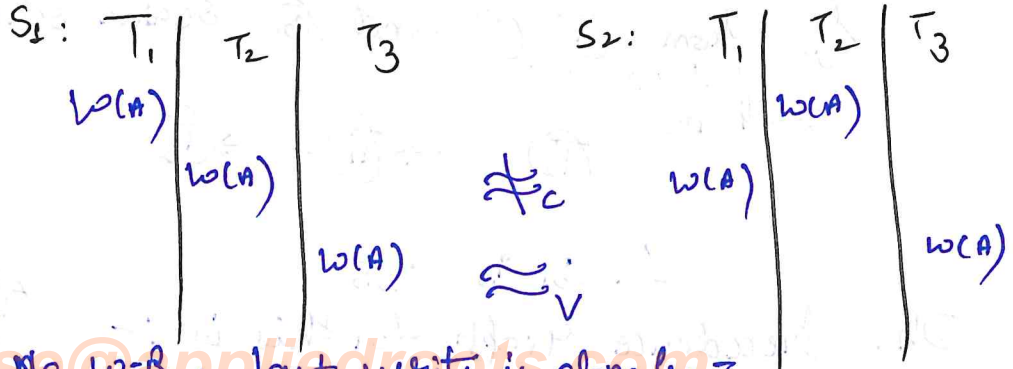
hence $S_1 \approx_v S_3$

So, a schedule S_1 is view equivalent to S_2 if all the three conditions are matched.

→ View Serializability:

Given a schedule S ~~that~~ is view equivalent to a serial schedule than it is called View serializable schedule (VSS).

Ex-

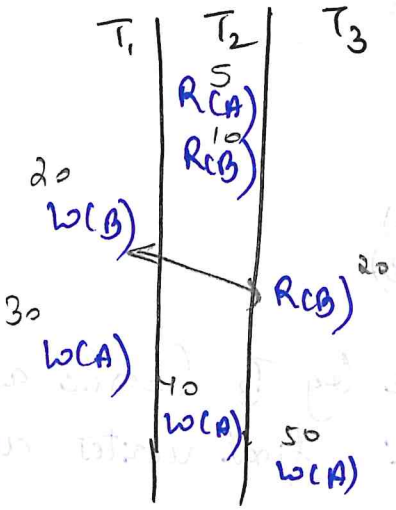


No reads, No W-R, last write is done by T_3 in both S_1 and S_2 ∴ 9t VSS but not CSS.

S_1 is serial schedule $\therefore$ S_2 is VSS but not CSS.

(Q1) Is it VSS?

S_1:



→ Let initial value of A = 5, B = 10

① initial read: T_2 should be before T_2 ($T_2 \to T_1$) and T_2 should be before T_3 ($T_2 \to T_3$)
(because in both case read will not satisfy)

② W-R seq. should not be altered : So $T_1 \to T_3$

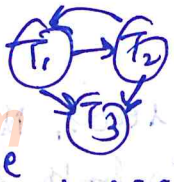
③ Final writes: $T_2 \to T_3$
$T_1 \to T_3$
So T_3 should write last

So from ① ② and ③ serial schedule will be

$T_2 \longrightarrow T_1 \longrightarrow T_3$

$\therefore$ S_1 is VSS

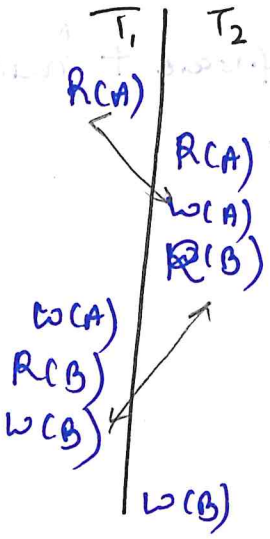
The precedence graph for this is :-



$\therefore$ there is cycle
$\therefore$ S_1 is not CSS.

(Q 2) Is it VSS?

S_1 :



⇒ ~~Let conflict serves~~ :

① Initial read ⇒ T_1 should be before T_2 as there is $R_1(A) - W_2(A)$

$\therefore T_1 \rightarrow T_2$

T_2 should be before T_1 as there is $R_2(B) - W_1(B)$

ie $T_2 \rightarrow T_1$

It is a conflict

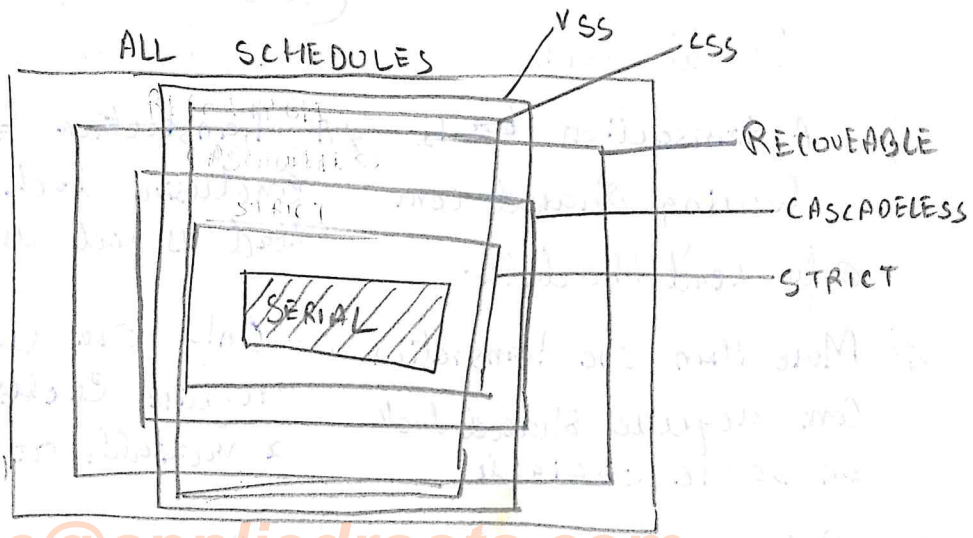
$\therefore$ S_1 is not VSS

$\therefore$ S_1 is not VSS $\therefore$ it is not CSS

NOTE -

Recoverability doesn't guarantee serializability.

Serializability doesn't guarantee recoverability.



18.7 Lock based Concurrency control:

Recoverability + Serializability ← Consistent Data

Lock is intuitively a software + hardware functionality provided (implemented) by OS.

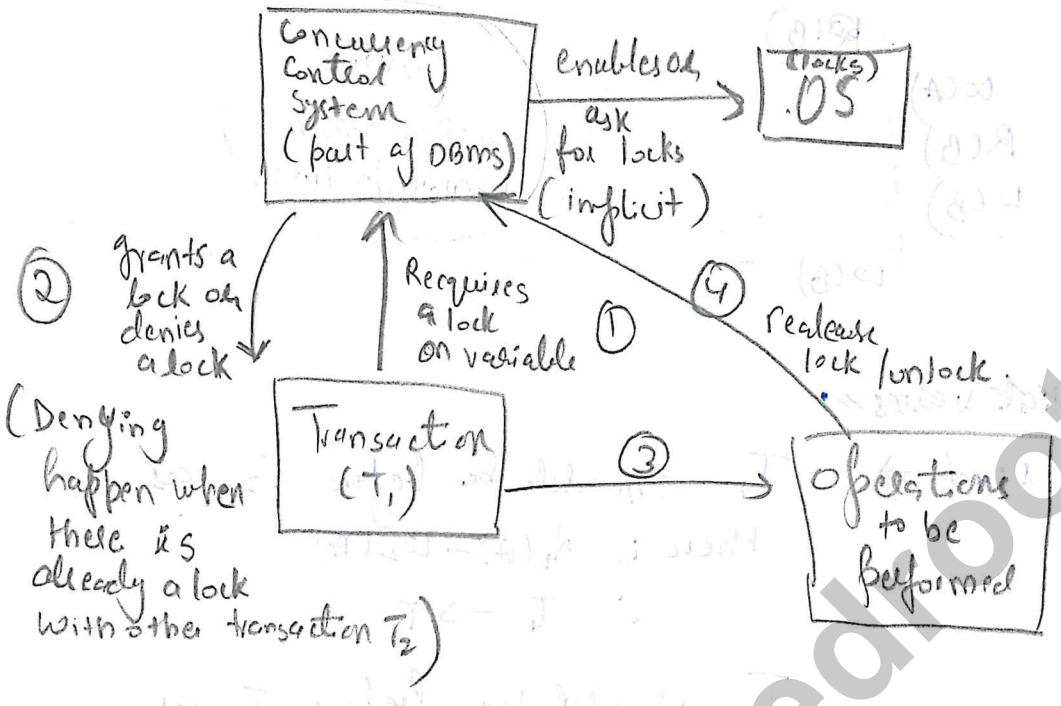
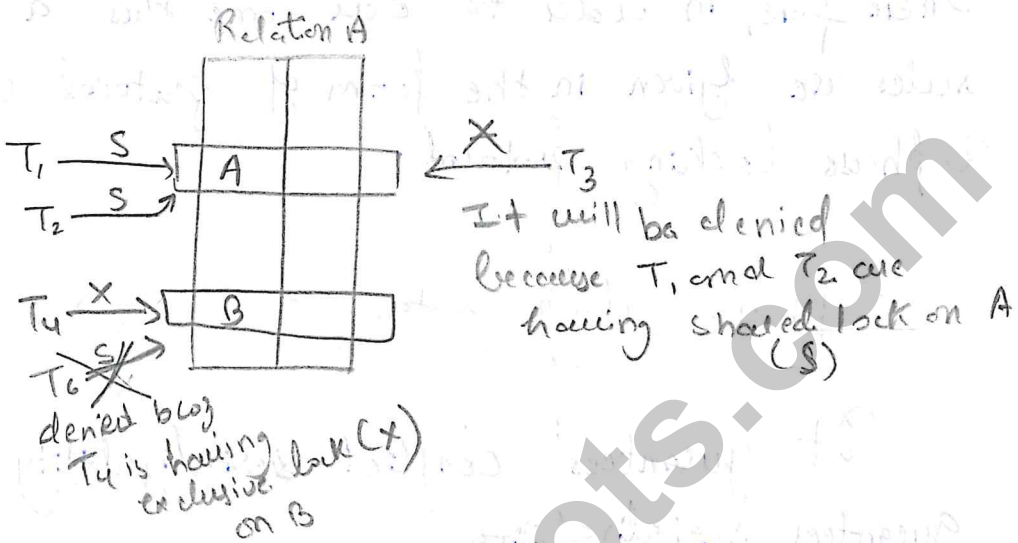


Fig: How locks work.

Locks
- Shared lock (read lock) (only read)
- Exclusive lock (write lock) (Read + write)

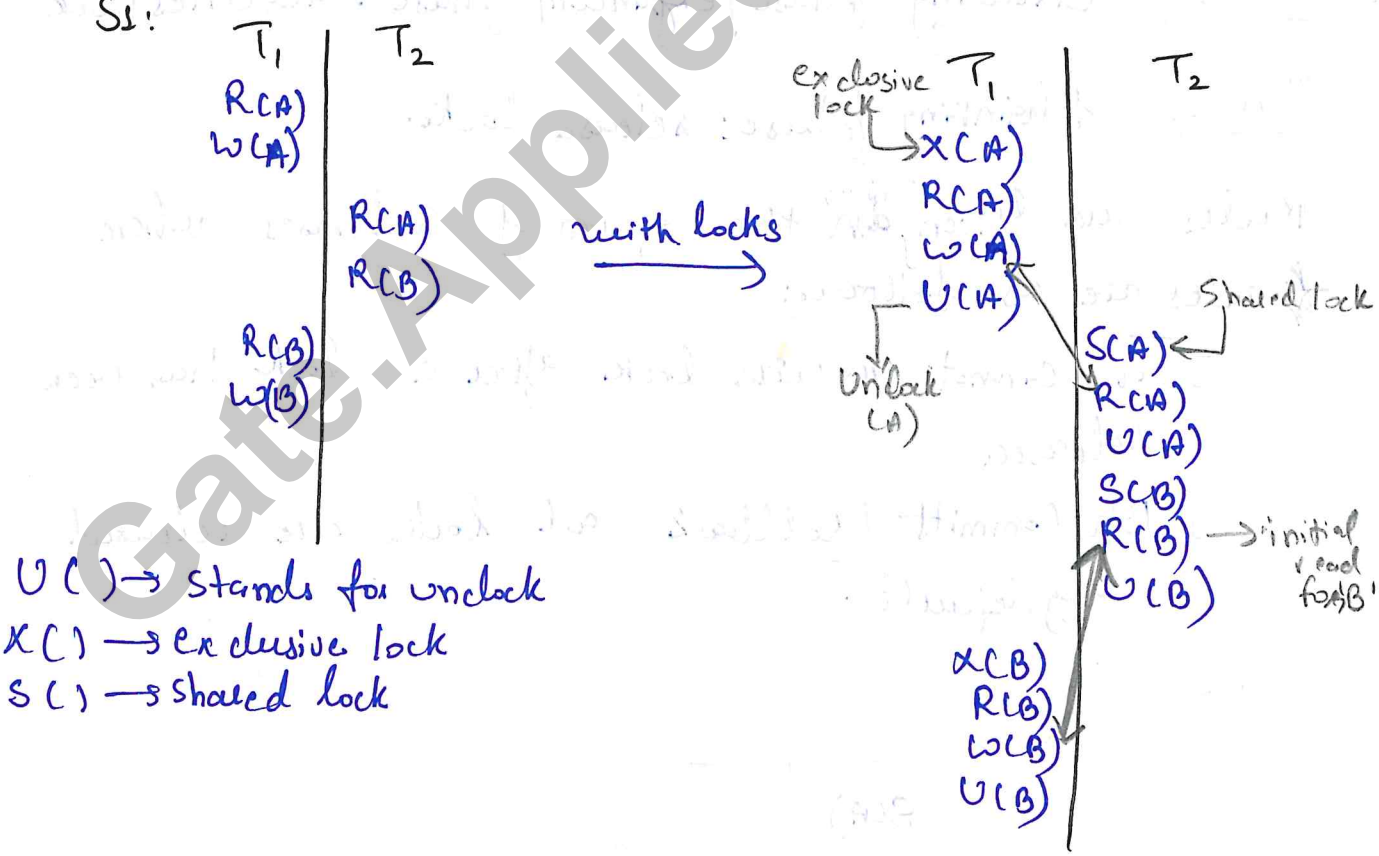
Shared lock	Exclusive lock
A transaction ~~has~~ T_1 having shared can only read the data.	A transaction ~~has~~ T_2 having exclusive lock can both read as well as write.
More than one transaction can acquire shared lock on same variable	Only one transaction can acquire exclusive lock on a variable at a time.
Represented by S	Represented by X.

NOTE - A transaction T_2, having exclusive lock on any variable will not allow any other Transaction T_2 to even have a shared lock on same variable. It will be denied.



So one way to achieve recoverability and serializability is through locks.

Example:-



U( ) → stands for unlock
X( ) → exclusive lock
S( ) → shared lock

Here, neither $T_2 \rightarrow T_1$ nor $T_1 \rightarrow T_2$ is valid. So, locks do not guarantees serializability

The Goal/aim behind using locks is to get both recoverability and serializability. Since, in the last example we neither achieved recoverability nor serializability. Therefore, in order to over come this a set of certain rules are given in the form of protocol called 2 phase locking protocol.

→ 2 phase locking protocol :- (2PL)

It guarantees conflict serializability in turn guarantees serializability

Conflict Serializability ⇒ Serializability

This locking protocol has 2 phases:

Phase 1:- Growing phase / expanding phase : acquires lock

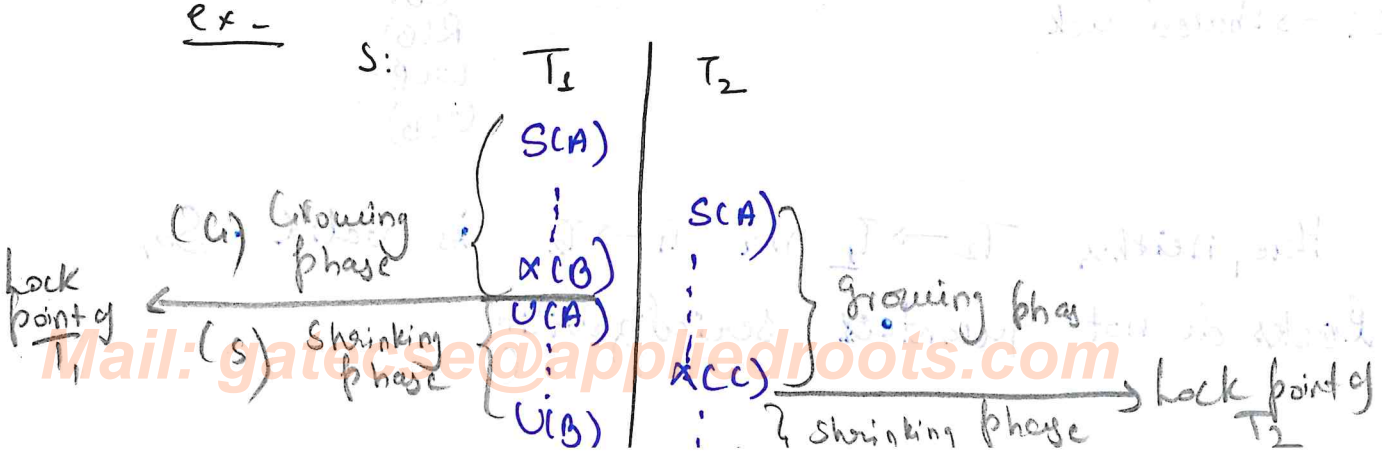
Phase 2:- Shrinking phase: release locks.

Rules are given in the form of 2 phases where rules are as follows:

[1] T_i cannot acquire lock after a lock has been released

[2] On Commit / rollback all locks are released by default.

Ex -



NOTE- lock point is the point in a Transaction where transaction has successfully acquired all the locks and has started releasing the locks.

NOTE- 2PL guarantees CSS ⇒ Serializable

Proof:- It is proved based on induction on # transactions

Again CSS ⇏ 2PL (every CSS do not follow 2PL)

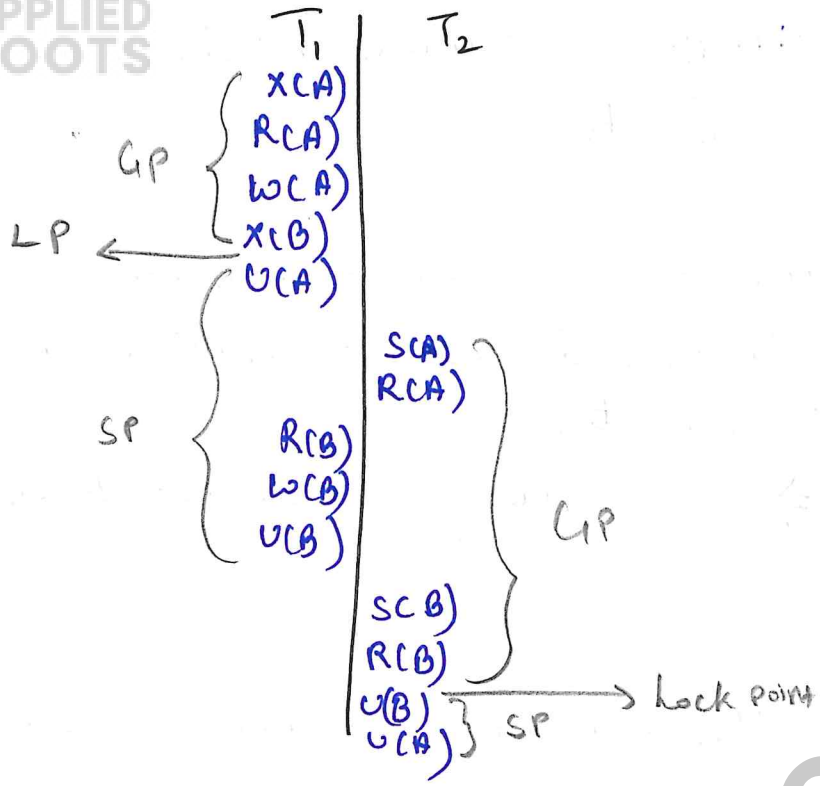
Q) How to obtain the equivalent serial schedule?

⇒ lock points are use to obtain the equivalent serial schedule. In previous example, lock point of T_1 is followed by T_2 ∴ its equivalent schedule (serial) is $T_1 \longrightarrow T_2$

Example

T_1	T_2	
X(A)		
R(A)		
W(A)		
X(B)		
U(A)		
	S(A)	
	R(A)	
	S(B)	← This will be denied beg T_1 already have exclusive lock on B.
	R(B)	
R(B)	⋮	
W(B)		
U(B)		

Q) g_s is 2PL?



⟹ To check for 2PL, we need to discover growing phase, shrinking phase and Lock points.

As we can see, it is following properties of 2PL ∴ it is CSS and it is equivalent to serial schedule $T_1 \longrightarrow T_2$ (based on the occurance of these LP)

→ Lock upgradation:-

We can upgrade the shared lock to exclusive lock ~~with~~ without unlocking it before upgrading.

Shared lock $\xrightarrow{\text{upgrade}}$ exclusive lock

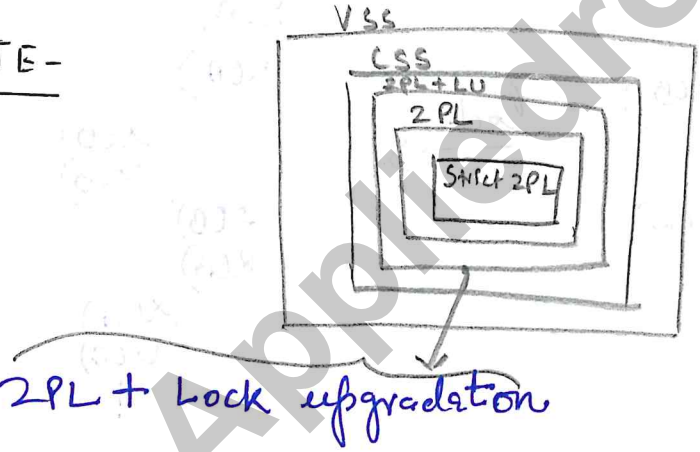
(This is possible only when T_i is the only transaction having shared lock on it)

Ex -

T_i	T_j
S(A)	
:	
X(A)	
:	
U(A)	

Initially T_i has shared lock on A, we can upgrade it to exclusive lock as no other transaction T_j have shared lock on it. (while upgrading there is no need to unlock the shared lock)

NOTE-



Example:-

S:

T_1	T_2
R(A)	
W(A)	
	R(A)
	C
C	

S_1: with locks →

T_1	T_2
GP { X(A)	
R(A)	
W(A) }	
U(A)	
	S(A) } GP
	R(A)
SP {	U(A) } SP
	C
C	

here, S_1 is 2PL ⇒ CSS ⇒ serializable

But there is no recoverability because T_2 committed before T_1.

We can't rollback transaction T_1 in S_1 ∴ S_1 is not recoverable

2PL doesn't guarantees recoverability but it guarantees serializability.

Hence, 2PL ⇏ Recoverability

→ Deadlocks :-

S:

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

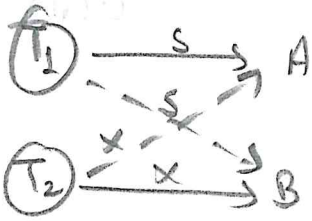
S_1 :

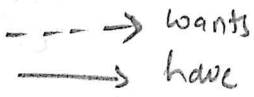
T_1	T_2
S(A)	
R(A)	
	X(B)
	W(B)
S(B)	
R(B)	
	X(A)
	W(A)
	⋮

locks → GP → stuck

GP → stuck

here T_1 has shared lock on A
T_2 has exclusive lock on B
T_1 wants shared lock on B
T_2 wants exclusive lock on A





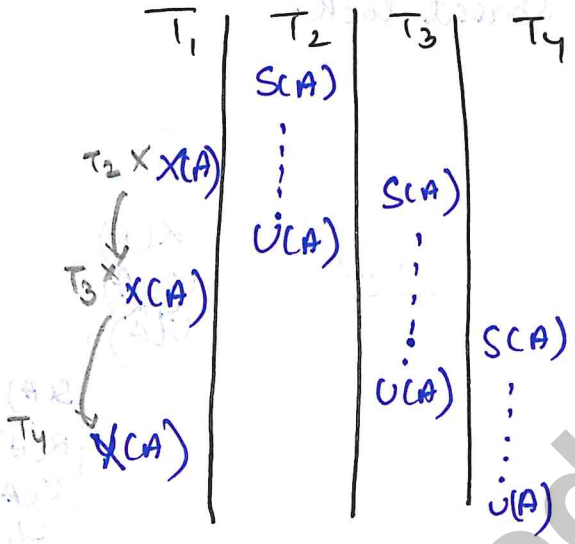
Both T_1 and T_2 are waiting for each other to release lock on B and A and B respectively.

Neither T_1 nor T_2 can proceed anywhere. This is called deadlock.

This schedule S_1 follows 2PL but it is having deadlock

So, 2PL can suffer with deadlocks. (Stuck in infinite loop)

→ Starvation :-



Here T_1 is being starved for getting lock on A because T_2, T_3, T_4 are requesting for shared lock (S(A)) before unlocking. This problem is called starvation problem as T_1 is being starved by other transaction. Though it is following 2PL but still it faces the starvation problem.

NOTE- 2PL may suffers from deadlock and starvation.

Q) Can we have system which can guarantee both serializability and recoverability?

⟹ Strict 2PL guarantees CSS (serializability) and strict recoverable schedules.

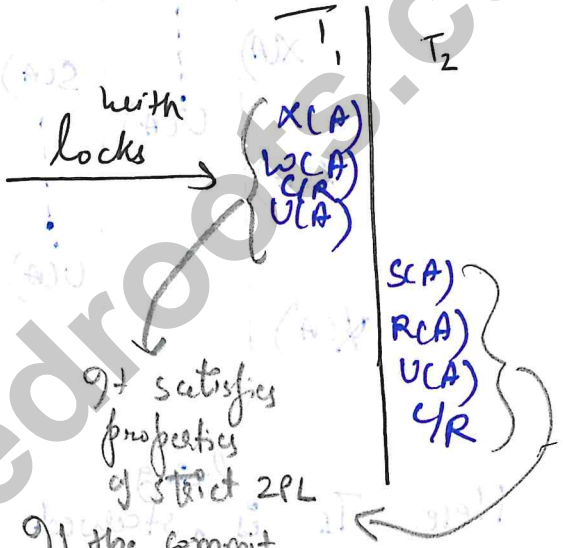
→ Strict 2PL:

Strict 2PL guarantees both Serializability (CSS) and Recoverability (Strict Recoverable).
It follows all rules of 2PL + all exclusive locks must be held till commit/rollback but there is no such restrictions with shared lock.

ex-

S_1 :

T_1	T_2
W(A)	
C/R	
	R(A)
	C/R

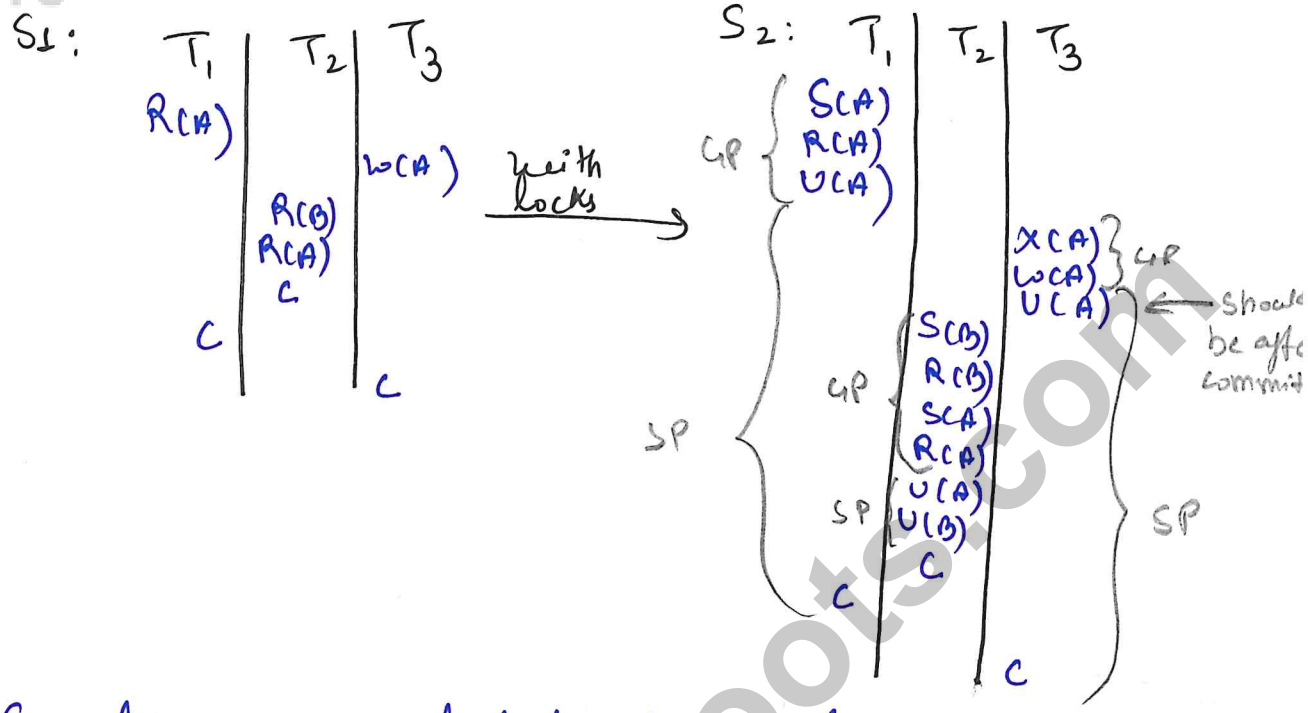


Strict 2PL ⟹ CSS + Strict Recoverable.

(Q) How to obtain equivalent serial schedule given that schedule is having Strict 2PL?

→ The sequence of lockpoints will be the order of transaction. The resultant order will be the equivalent serial schedule.

(Q1) Is it strict 2PL?



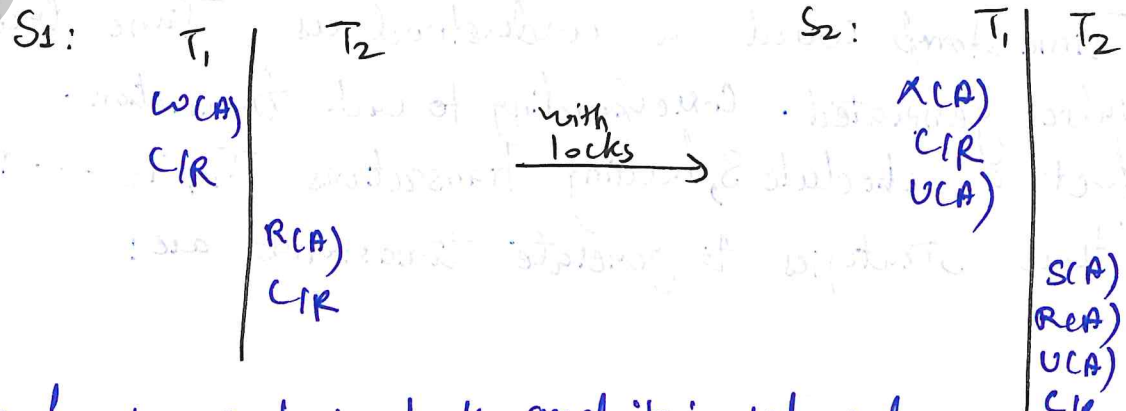
⇒ S_2 follows all properties of 2PL but it is not strict 2PL because in T_3 before commiting after $W_3(A)$ we are releasing lock.

∴ S_2 is not in strict 2PL.

→ Strong strict 2PL / Rigorous 2PL:

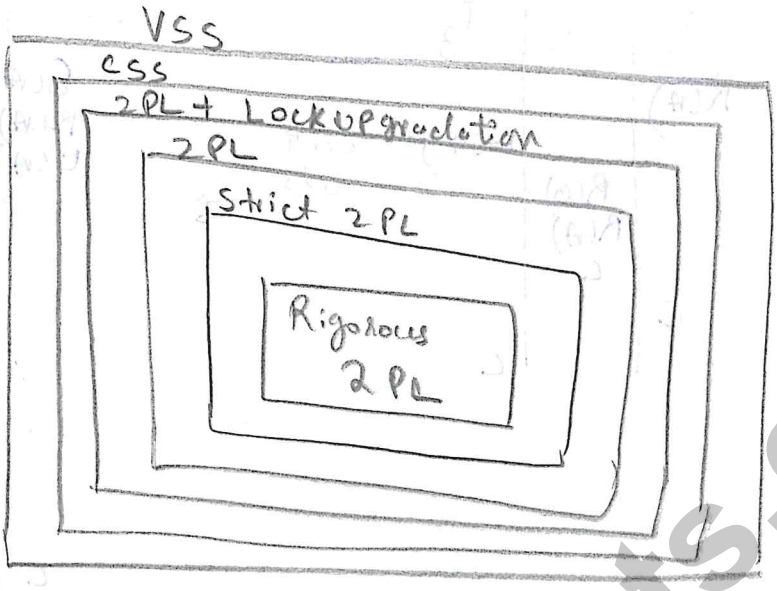
2PL + all locks (shared or exclusive) should be held till transactions commit or rollback.

Ex:-



S_2 is having exclusive lock and it is released after commit. ∴ S_2 is strict 2P. But it is not Rigorous 2PL

NOTE – Strict 2PL may suffer with both deadlock and starvation.



18.0 Timestamp based protocols + Deadlock and Starvation prevention:

Another way to strategy to obtain both serializability and recoverability is Time stamp based protocols.

In lock based protocol system we need to have OS enabled locks / software locks while Timestamp based protocols are alternative of lock based protocols.

Time stamp could be understood as Time based value generated. Corresponding to each transaction.

Let a schedule S, having transactions $T_1, T_2 \cdots T_n$, The two strategies to generate timestamp are:

Generate a time stamp:

① Use the system time generated by OS to generate Ts (time stamp)

② Create a shared counter and use it to assign a unique no. for each transaction.

This counter is thread safe.

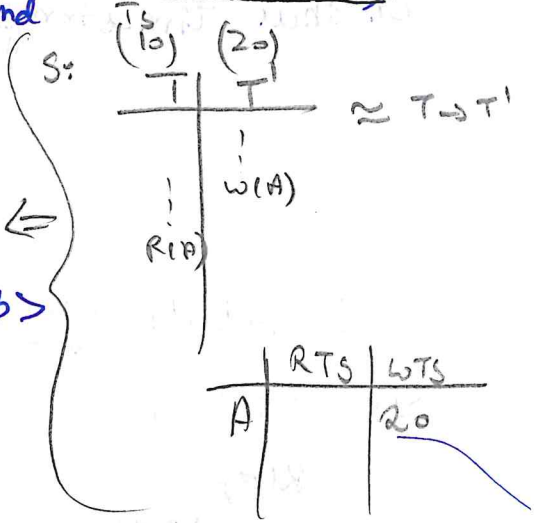
→ Time stamp based protocol: (TSP, TSOP)

↳ Timestamp ordering protocol

* Each transaction must obtain a unique timestamp @ start.
* Each data-item also has time stamp (Write time stamp (WTS), Read Time stamp (RTS)

Given the above construct the following 2 conditions should be satisfied.

Case1: let assume transaction T and T' (T wants to R(A) and T' wants to W(A))



[1] $TS(T) < WTS(A)$

→ Rollback (T) and Restart with new Timestamp (new Timestamp > T)

[2] $TS(T) > WTS(A)$

⇒ T' → T → execute R(A)

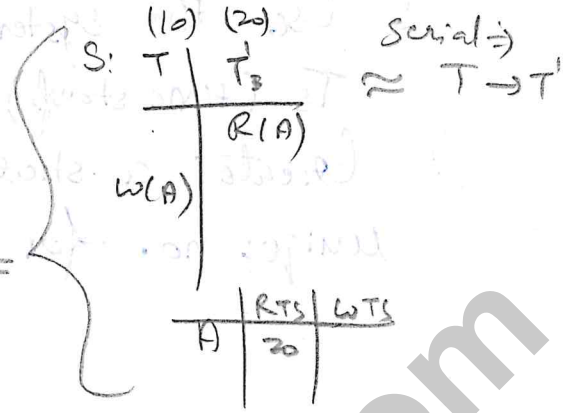
→ RTS(A) = max(RTS(A), TS(T))

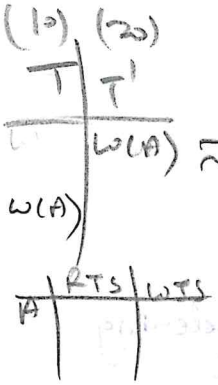
Case 2:

If T wants to $w(A)$:-

[1] $RTS(A) > TS(T)$

→ Roll back (T) and restart





[2] $WTS(A) > TS(T)$

→ Roll back (T) and restart

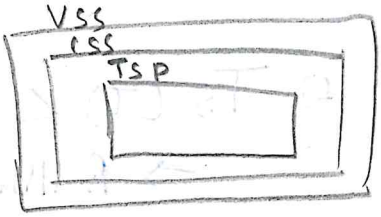
[3] otherwise,

→ execute $w(A)$

→ set $WTS(A) = TS(T)$

NOTE:-

* Time stamp based protocol guarantees CSS (~~no~~ conflicts reads). Given any TSP, we can obtain equivalent serial schedule by ~~sampling~~ simply ordering transactions based on their timestamps.



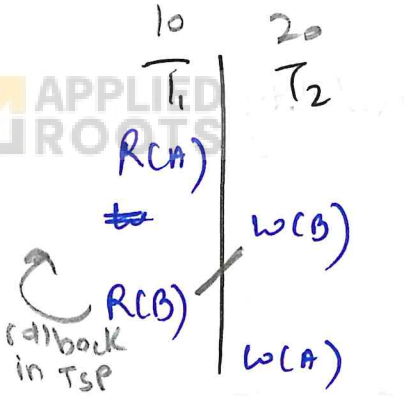
→ Deadlocks w.r.t TSP :-

S1:

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

S1 suffers with deadlock in case of 2PL, Strict 2PL.

Let's check it with respect to (w.r.t) TSP.



	R-TS	W-TS
A	10	
B		20

Since T_L has smaller TS ∴ it should be equivalent to serial schedule

$T_1 \longrightarrow T_2$

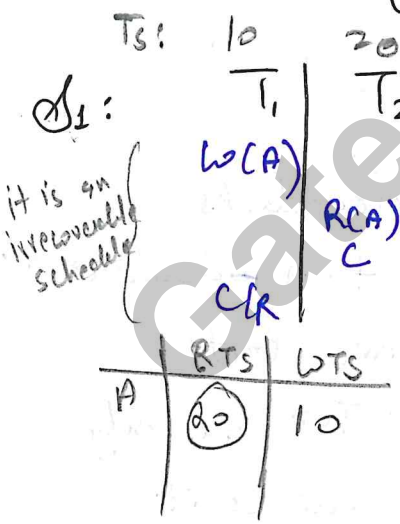
Since S_1 is not equivalent to $T_1 \longrightarrow T_2$ as TSP will not allow R1(B) to takes place.

It will say rollback R1(B) and restart the whole thing.

$T_1 \longrightarrow T_2$

Hence,* TSP guarantees no-deadlock. It doesnot allow deadlocks to take place by not executing the operation leading to deadlock.

→ Recoverability wrt TSP:



	RTS	WTS
A	20	10

TSP will try to check whether Schedule S_1 is equivalent to serial schedule $T_1 \longrightarrow T_2$ (as T_1 has smaller timestamp)

TSP will execute $R_2(A)$ as $TS(T) > WTS(A)$ ∴ $R_2(A)$ will get executed and RTS will be updated to 20.

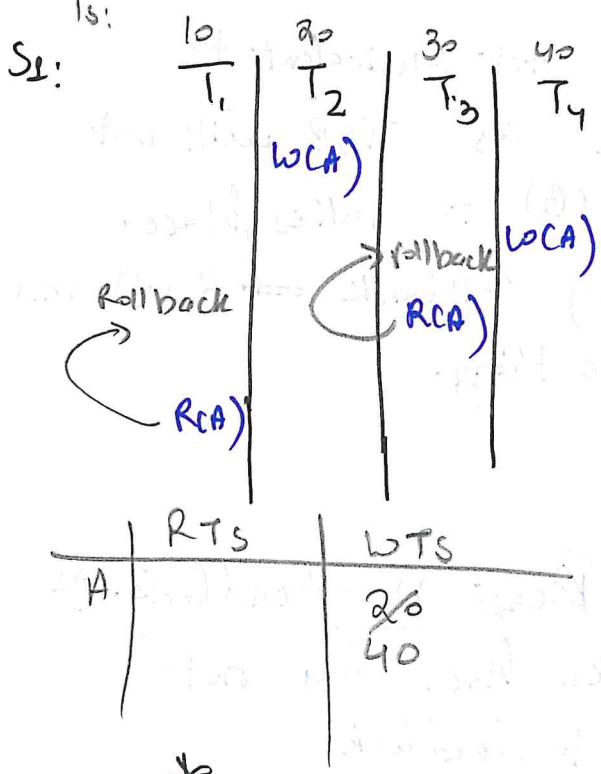
∴ TSP is allowing a irrecoverable schedule

It is not forcing it to rollback.

*

Hence, TSP doesnot guantee recoverability

→ Starvation wrt TSP:



Task of TSP, to make S1 equivalent to ≈ $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4$

Here T_2 and T_4 can starve T_1 and T_3 ∴ starvation can happen as Trasactions with larger TS can force Transaction with lower TS to rollback.

*

∴ TSP doesnot guantee starvation free schedule

→ Strict Timestamp ordering protocol (Strict TSP):

TSP ensures serializability and no-deadlocks while strict TSP ensures serializability + no-deadlocks + strict recoverability. Basic Construct is same but in place of 2 cases (as in TSP), strict TSP has only 1 case. (Because for strict recoverability we can't have WR and WW)

Role for Strict TSP

So, if any transaction T wants to read or write ie R(A) or W(A)

if, WTS(A) < TS(T)

→ Rollback (T) and restart

else,

Everything stays the same, just update the WTS and RTS accordingle

This ensure No WR and No WW and hence, ensures strict recoverability. Since, this condition is more strict than TSP ∴ it definetly ensures no-deadlock and Serializability.

NOTE- Strict TSP ⊂ TSP ⊂ CSS ⊂ VSS

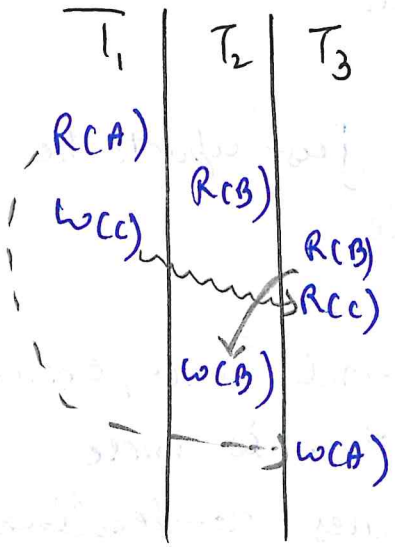
(it is also strict recoverable + no-deadlock but starvation may occur)

Q1) What are the valid TS(T_i) values to avoid Rollbacks?

S1:

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

⇒ In order to assign TS we need to find that the given schedule S is equivalent to which serial schedule.



T_1 should happen before T_3 ~~and~~ due to $R_1(A)$ and $W_1(C)$. $T_1 \rightarrow T_3$

T_3 should happen before T_2 due to $R_3(B)$ $T_3 \rightarrow T_2$

∴ Serial order is

$T_1 \rightarrow T_3 \rightarrow T_2$

So if we assign TS values as

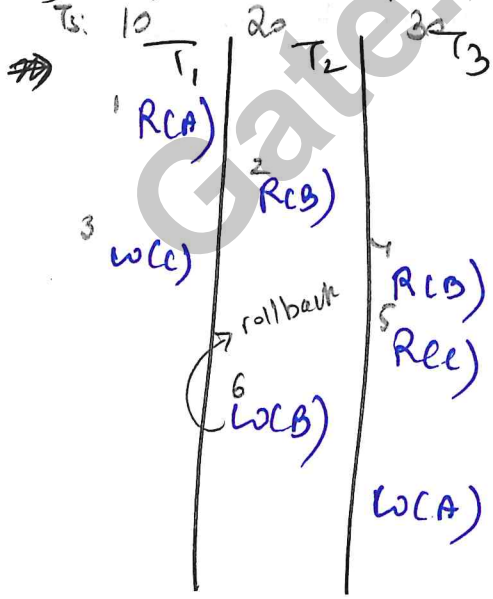
$T_1 - 10$

$T_3 - 20$

$T_2 - 30$

These, TS values will ensure that if we apply TSP then there will be no rollbacks.

(Q2) Which T_i will rollback first? (TS are already given in question)



⇒ For the give TS values

	RTS	WTS
A	10	
B	~~20~~ 30	
C	30	10

∴ the equivalent serial order TSP ~~execution~~ trying to maintains

$T_1 \rightarrow T_2 \rightarrow T_3$ (acc. to TS)

While performing 6th operation

$TS(T) = 20$, $RTS(B) = 30$, $WTS(B) = 20$

and equivalent schedule is $T_1 \rightarrow T_2 \rightarrow T_3$

$\therefore TS(T_2) < TS(T_3)$ $\left(RTS(B) \geq WTS(B)\right)$

$\therefore W_2(B)$ will not be allowed and it will be rolled back.

$\therefore$ First transaction which will roll back is T_2.

NOTE-

2PL, Strict 2PL } Cannot avoid deadlock + starvation

TSP, Strict TSP } Can avoid deadlock but not starvation.

In real world combination of 2PL and TS are used, as TS can effectively avoids deadlocks.

$\therefore$ Strict 2PL + TS are used in practice to ensure Serializability + recoverability + no-deadlock.

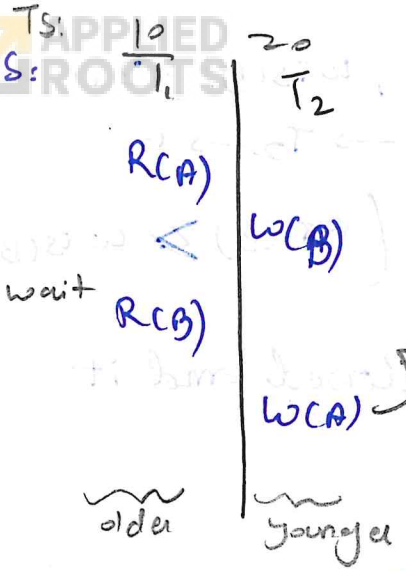
(8) Additional strategies to be introduced in lock based protocol to ensure deadlock and starvation?

$\Rightarrow$ There are 2 strategies

(1) wait-die

(2) wound-wait.

(1) Wait-die: Use strict 2PL + TS with 2 conditions cases



Case 1: If older transaction wait → younger transaction than, Continue waiting (wait)

Case 2: If younger transaction wait → older transaction then, Roll back (die)

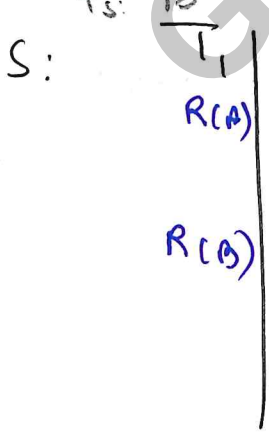
T_1 is having Shared lock on A while T_2 have exclusive lock on B.

∴ T_1 is older transaction waiting for T_2 (younger transaction) to release lock on B. ∴ acc. to rules, it will keep waiting.

T_2 is the younger transaction which is waiting for T_1 (older transaction) to release lock on A ∴ Based on rules, it will get rolled back (Die)

Hence, S2PL + TS will overcome the deadlock problem.

(2) Wound - wait: It also uses Strict 2PL + TS with 2 cases



Case 1: Younger transaction wait → older transaction then, Continue waiting (wait)

Case 2: older transaction wait → younger transaction then, Rollback (Die) younger Process

NOTE

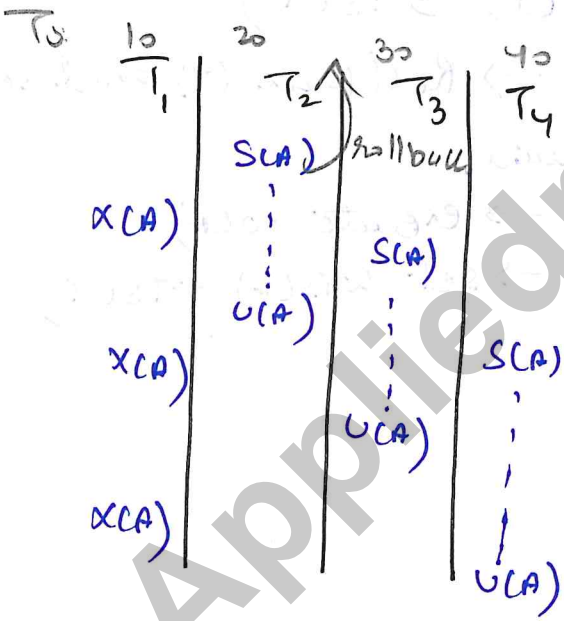
In both the the cases wait-die and wound-wait we rollback the younger transaction.

→ Starvation prevention:

* Wounded-wait:

① Older $\xrightarrow{\text{wait}}$ younger $\xrightarrow{\text{rollback}}$

② younger $\xrightarrow{\text{wait}}$ older (waiting)



With the help of wound wait we can prevent starvation

Hence, Strict 2PL + wound-wait (TS) will ensure

Strict 2P guarantees this → Serializability, Strict Recoverability

Wound wait(TS) ensure this → no-deadlock, no-starvation.

} Strict TSP guarantees this

18.9: Thomas write Rule (TWR):

It is based on Timestamp based protocol. A simple TSP have 2 cases. The 2nd case of TSP has 3 conditions

Case 2: – T wants to write(A) / W(A):

[1] RTS(A) > TS(A)
→ Rollback (T) and restart

[2] WTS(A) > TS(T)
→ Rollback (T) and restart

[3] Otherwise,
→ execute W(A)
→ set WTS(A) = TS(T)

S_1:

10	20
T_1	T_2
	W(A)
W(A)	

This rule is changed in Thomas write rule

It often refers to as Thomas write rule

Ex-

10	20
T_1	T_2
	① R(B)
② R(A)	
	③ W(C) → it will always overwrite $W_1(A)$
④ W(C) ← don't execute	
C	
	C

	RTS	WTS
A	① 10	
B	② 20	
C		② 20

⑨ WTS(C) = 20 > TS(T_1)

The serial schedule corresponding to it: $T_1 \rightarrow T_2$

Ace to serial schedule, anyway T_2 write will overwrite the T_1 write. (anyhow $w_1(A)$ will be overwritten). So TWR say if Transaction T_2 has larger TS and has written a value of variable and we have an older transaction T_1 who is writing after it (in serial order anyway T_2 will overwrite T_1), than don't execute $w_1(A)$.

So, it says don't rollback if such condition occurs, just ignore that operation.

TWR is used in context of TSP, it is not used in isolation.

FILE STRUCTURES (Sequential files, indexing, B and B^+ trees)

19.1 Files and Indexing: Introduction:

DB-tables storedin → Hard Disk (Secondary memory)

Objective: Access data on the disk fast
→ Search, Read, write, modify, delete

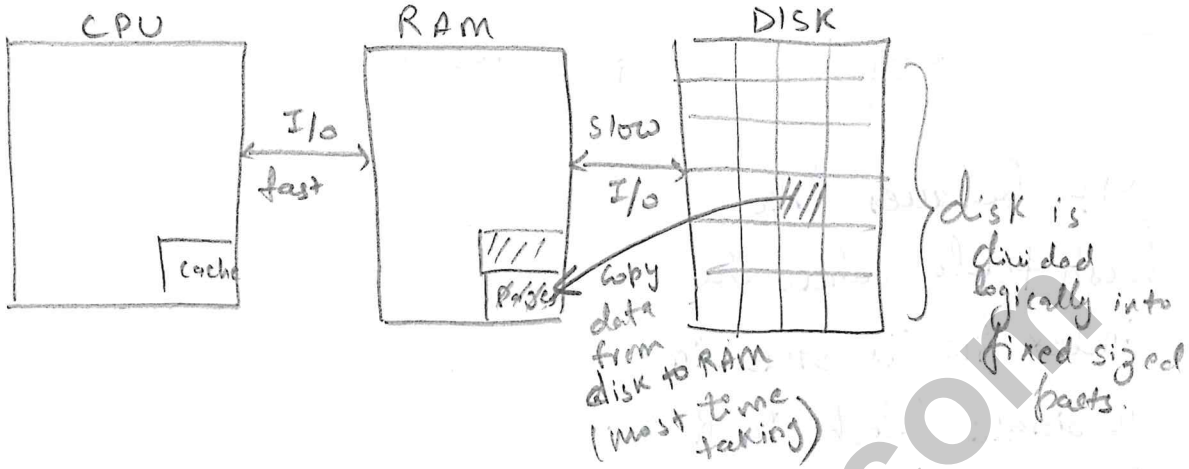
→ why do we need index?

Index makes searching of an attribute / column or a tuple more faster.
Index is a data structure such that access based on that attribute is faster. (It is suggested to have indicies on primary key)
In the given example id is the index + search by name.

id	name	addr
1	n_1	a_1
4	n_2	a_2
6	n_3	a_3
⋮	⋮	⋮
num (index)	str	str

We can have multiple indices in a given table.

~ Accessing data on disk:

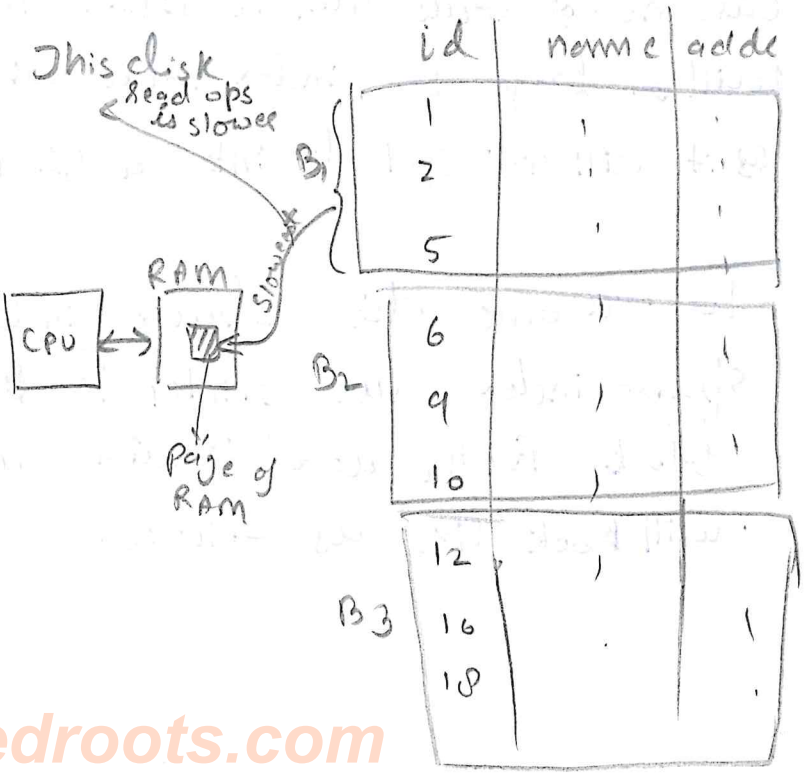


So, how much time it takes us to execute an SQL depends on how efficiently we copy data from disk to RAM and back. (Everything is stored in blocks in a disk, a file, table is also divided into blocks and # of records stored in each block depends on size of each record)

If size of each record is S and Block size is 512B then no. of records it can store is $512/S$.

~ Cost of accessing data:

The DISK-READ operation is very costly, but once the block is loaded in RAM then it is very cheap to read data and perform required operations.



So, our aim is to minimize the number of disk reads.

→ Indexing: (dense + sparse)

It behaves like hash table where the index file is stored in a single block in RAM.
It is dense because for every row we have a pointer.
Sparse indexing is when we have pointers for a subset of rows.

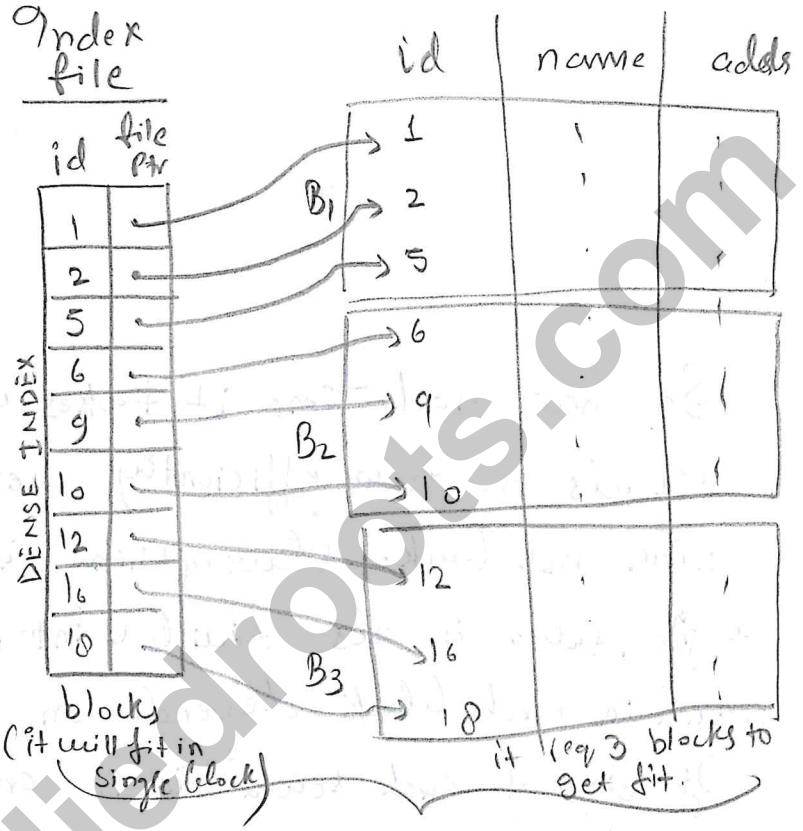


Fig: Dense indexing

Disadvantage of dense index is when we have large tables, in this case no. of rows will be high ∴ index table will be large so, index table will also get divided in blocks as it will not get fit into a block.

To overcome this sparse index come into picture. Sparse index have pointer to the first row of each block. For the above relation customer. Sparse index will look like as follows.

Our goal is to minimize disk-reads.
The index file will be smaller in case of sparse index and
∴ could easily fit in a block in RAM or disk
So with just 2 disk reads we can get the required tuple.

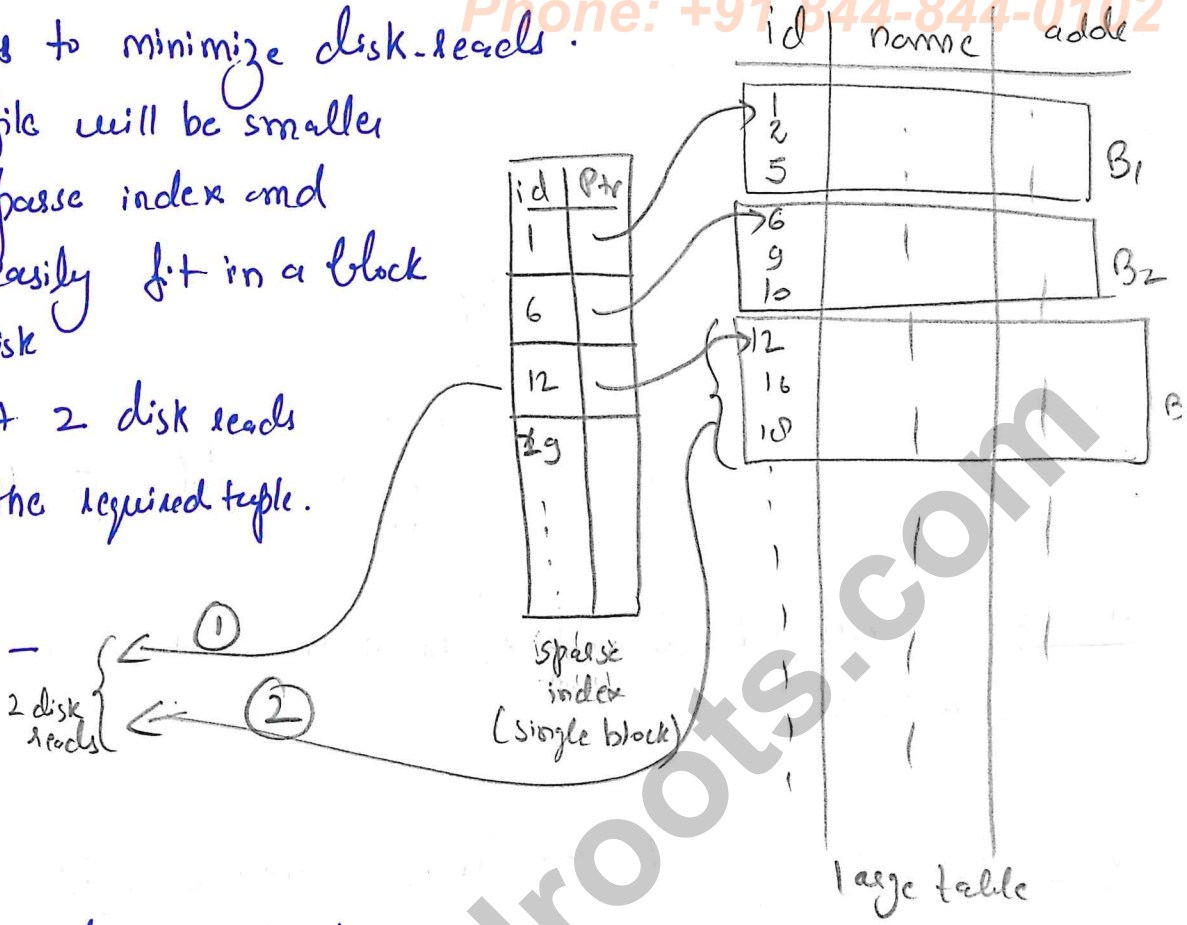


Fig: Sparse indexing

Suppose we have super large table and even sparse index could not fit in a single block of Disk.
In this case we will have.
(n+1) disk reads in worst case

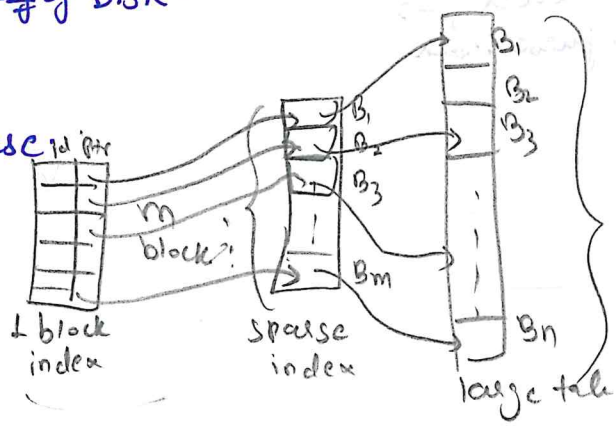


Fig: multilevel index

This leads to the idea of multilevel indexing

In multilevel indexing, we build an index out of index so that we can fit the index into a single block of disk or RAM. The multilevel index seems to be like a big tree. We can arrange the index in the form of m-way search trees. An m-way search tree will have m keys and m+1 pointers in a node (Connect to the concept of BST)

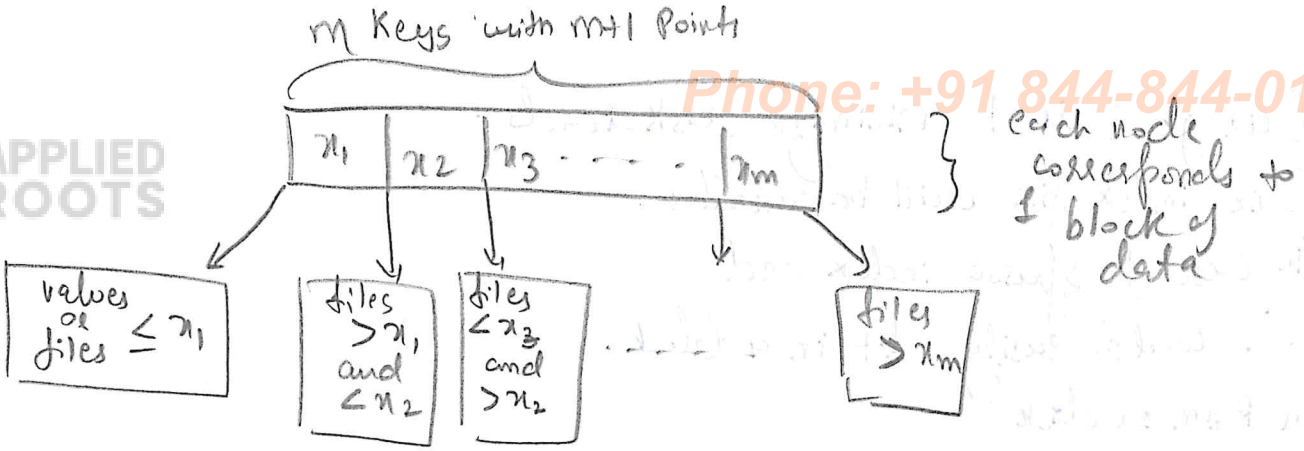


Fig: m-way tree node structure.

NOTE- multilevel index is nothing but m-ary Search trees.

19.2 B- Trees and B+ Trees with examples:

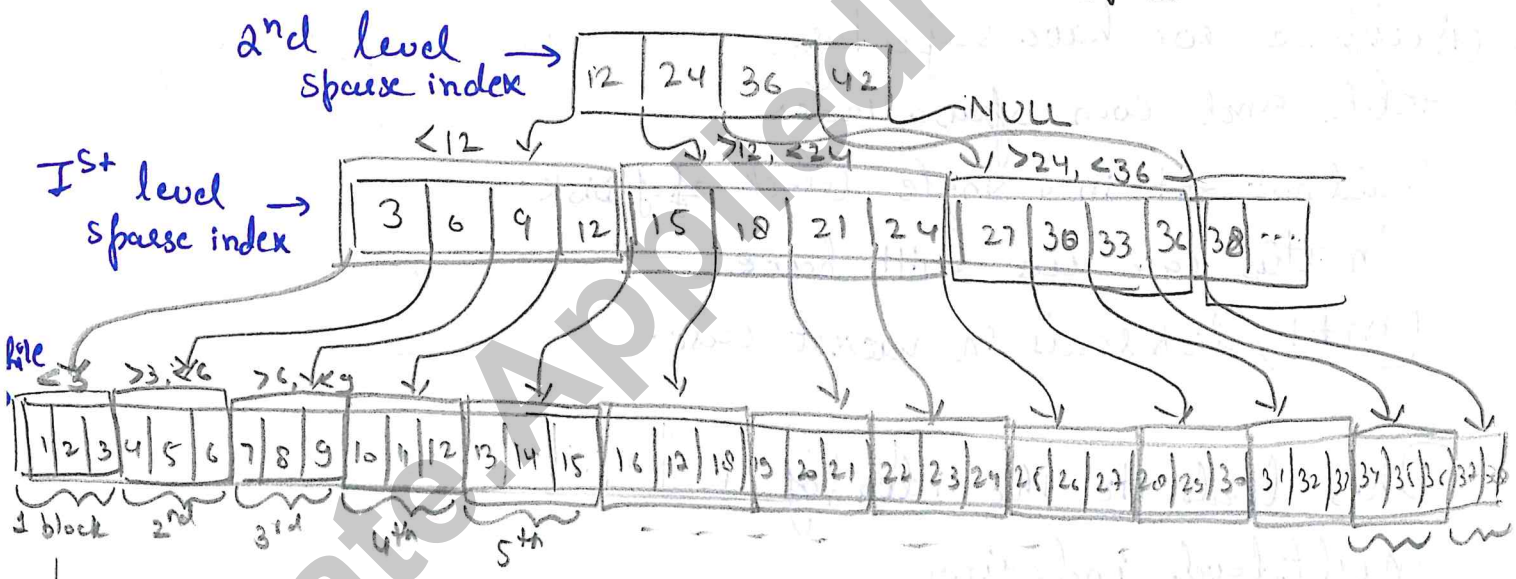
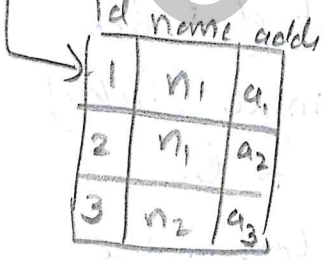


Fig: multilevel index as an m-ary search tree



On searching 11#, we will have 1st read on 2nd level index, 2nd read on 1st level index and then final read of record ∴ total 3 reads are required in this case.

→ How to construct ~~an~~ optimal m-ary tree?

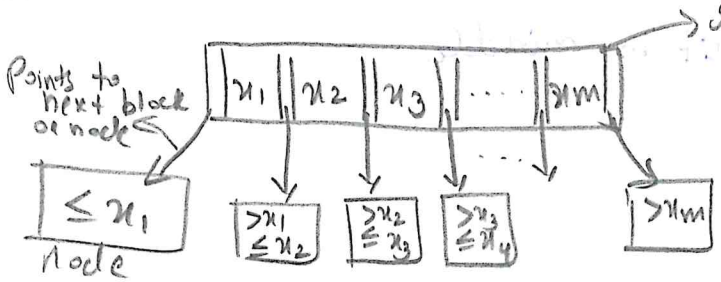
Using m-ary trees we have reduced the no. of disk reads. Now the next task is to store them minimimeum no. of blocks so that height of the tree doesnot exceed. If each of the node/block is totally filled than we can achieve minimal depth of tree (pack the data), this is the only way to get optimal m-ary tree. (We want height of the tree as small as possible as height of the tree = no. of disk reads in approx).

For this, the concept of B-tree and B+ trees come into picture.

* B-trees:

* Inspired by/from m-ary trees.
* It is a rooted tree
* Each node in B-tree is a block

Typical structure of a node is as follows:



$x_1 < x_2 < x_3 \cdots < x_m$

$x_1, x_2, \cdots x_m$ are called Keys

So each node have m keys and m+1 pointers.

These $x_1, x_2 \cdots x_m$ are pointer to a tuple with id $x_1, x_2 \cdots x_m$ respectively.

* All leaf nodes are at same depth.
* All internal nodes (non-leaf node) except root must have $\lceil m/2 \rceil$ children. (each of nodes must be half filled)
* Root has atleast 2 children

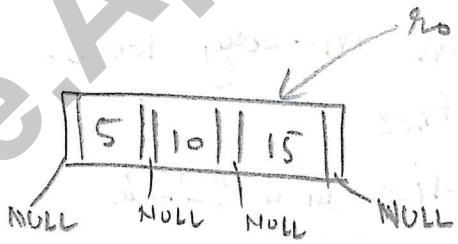
→ These 3 rules ensure that depth of tree is as small as possible.

* Order is maximum no. of pointers of a node in a tree.

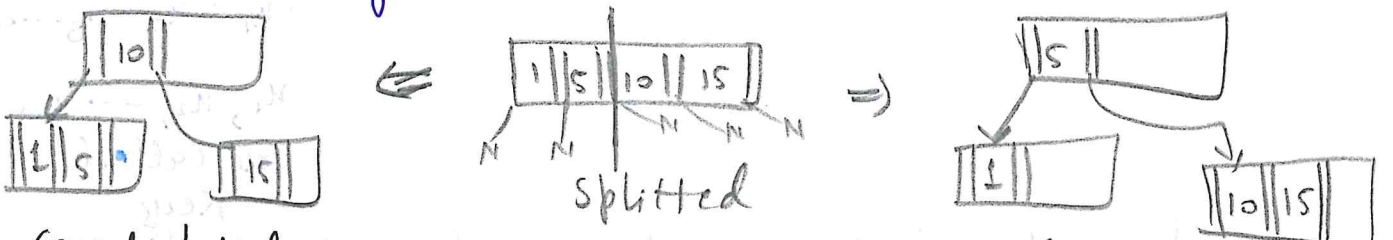
~ Build a B-tree (insert):

(A) ∵ Order of tree = m+1 = ## Ptr in a tree at max ∴ If order = 4, than no. of keys possible in a node = 3

(1) Insert 5, 10, 15:



(2) Insert 1: We cannot insert 1 in the root node.
∴ we split in middle

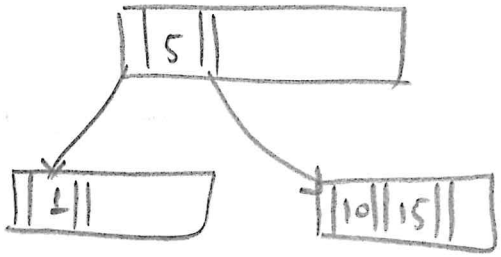


Case 1: Left biasing (splitting st more elements are in left sub tree)

Case 2: Right biasing (splitting such that more elements are on right tree)

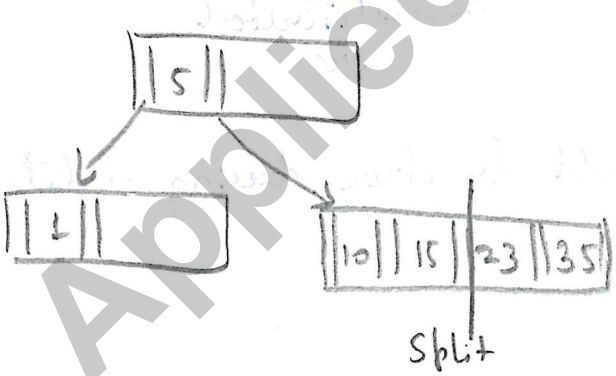
Note that both right biasing and left biasing is valid. Let us continue with right biased tree (ie case 2a)

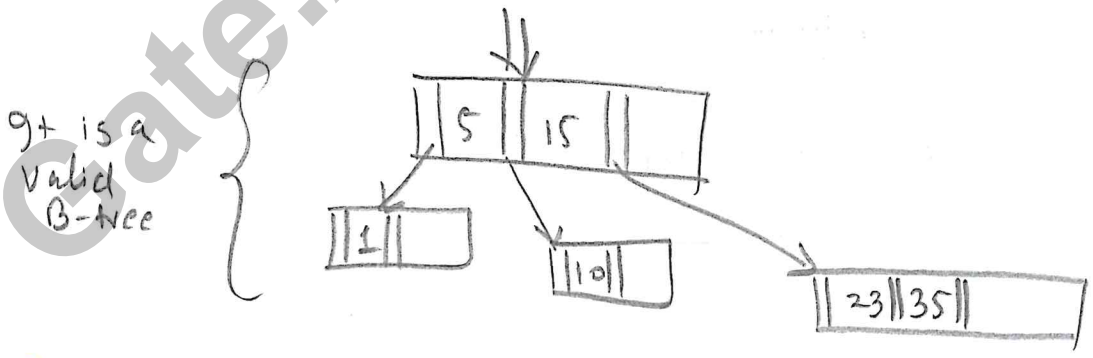
③ Add 23 :



We always fill tree in bottom up method. ∴ we have left in the node where we have some space.

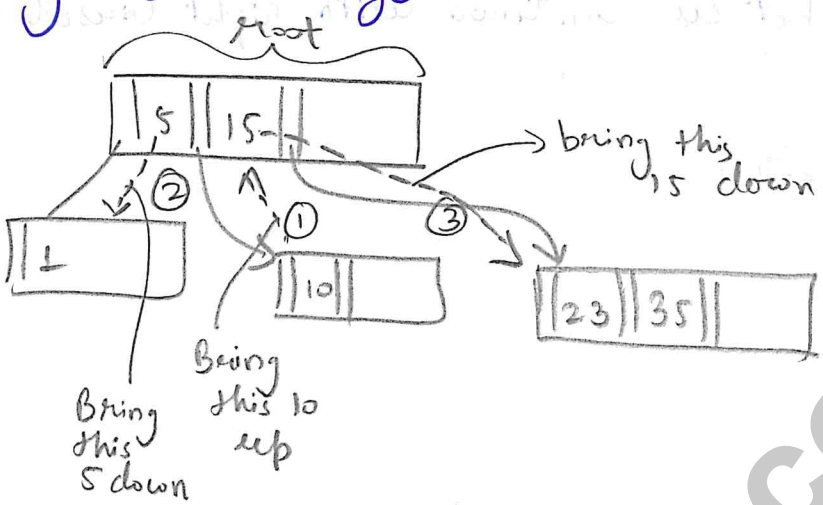
④ Add 35 :



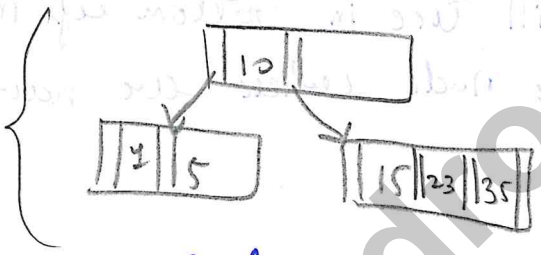


But here, there is some problem as there is lot of empty space in each node. So, we should somehow adjust them so that we can get utilize those empty spaces. ie Redistribute the data/keys.

On redistributing we will get



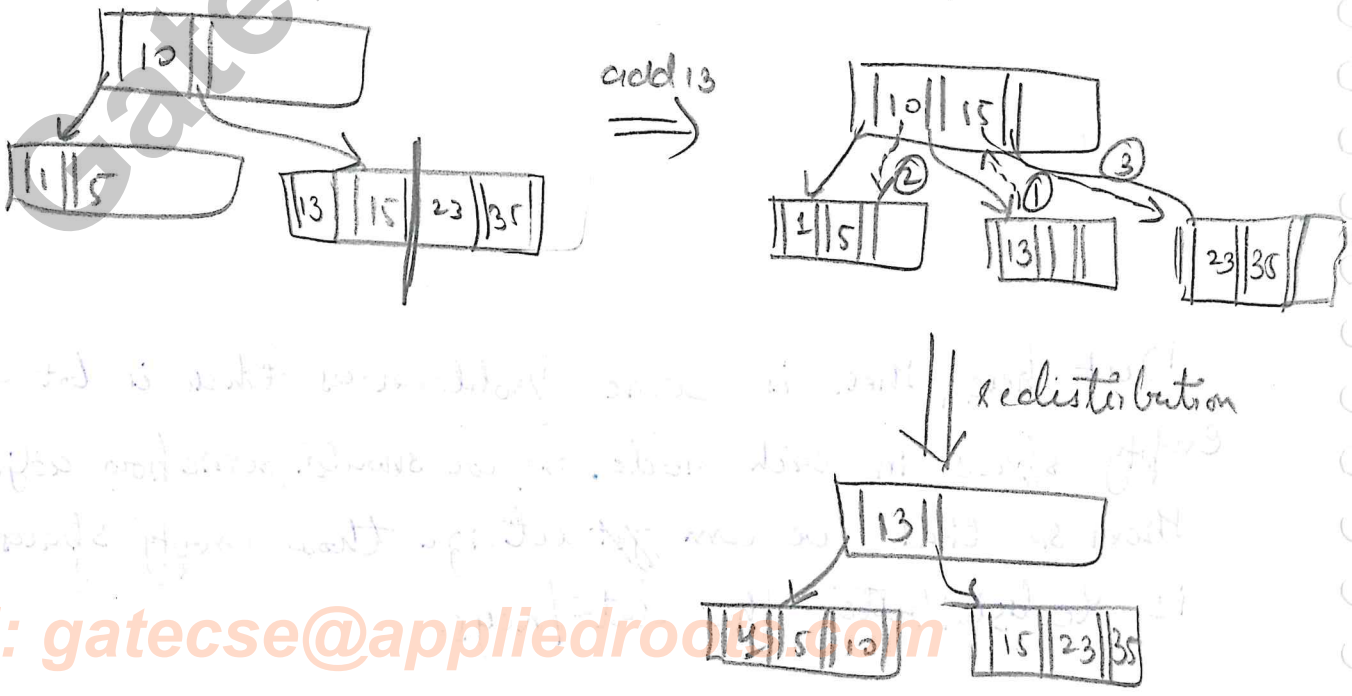
We have saved spaces by making it into 3 blocks in place of 4 blocks.



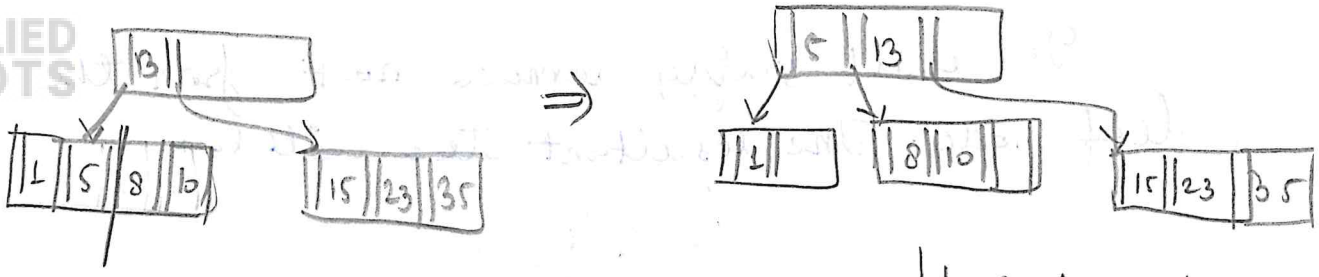
Redistributed tree

Redistribution could be done during deletion as well as insertion.

⑤ Add 13:



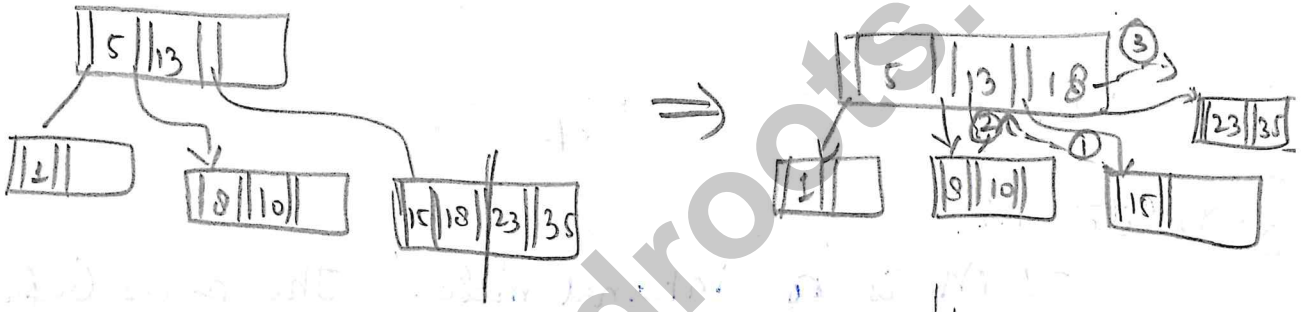
(6) Add 8 :



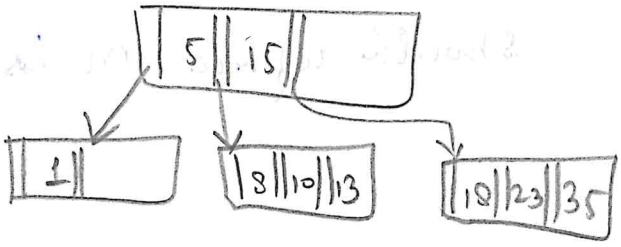
Redistribution

X not possible in this case

(7) Add 18 :



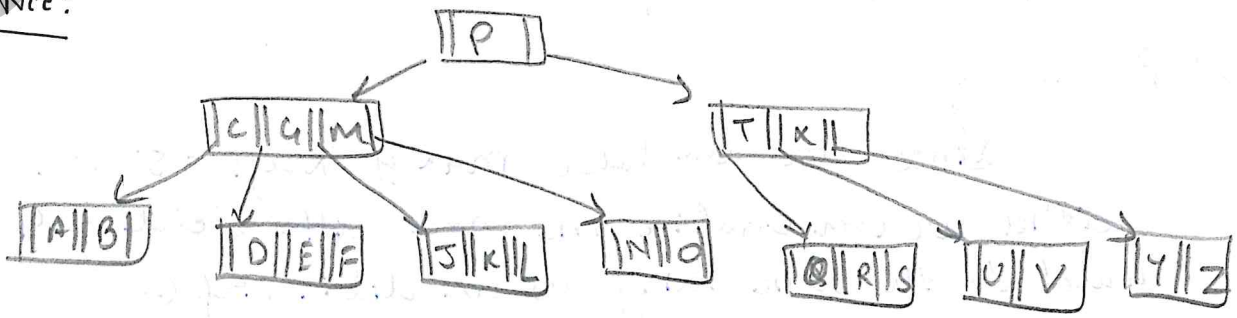
redistribution



~ Deletion from B-tree (Delete) :

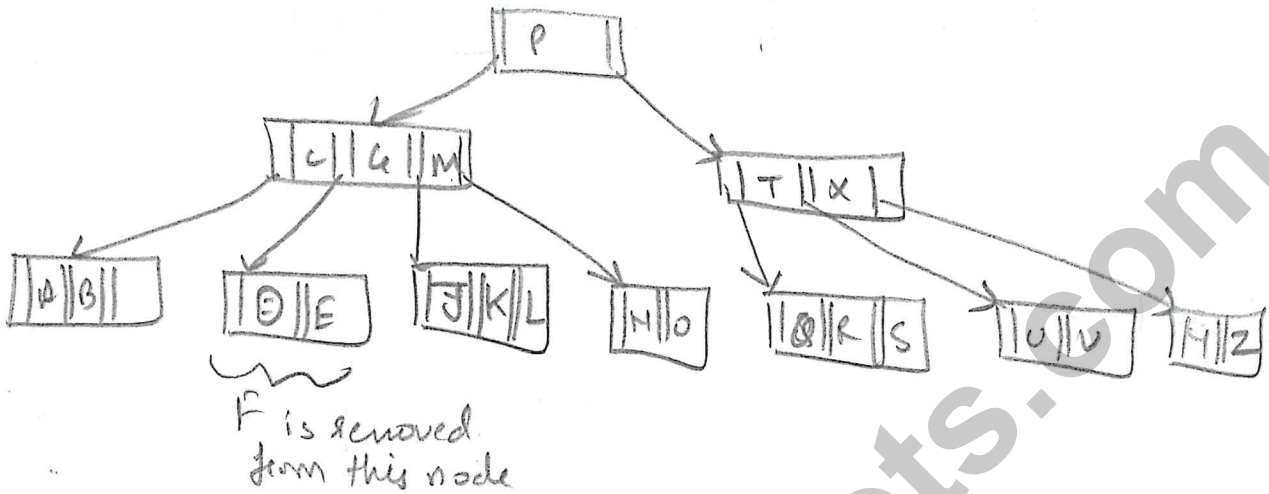
Order = 6 ∴ max # Keys = 5

initial tree :



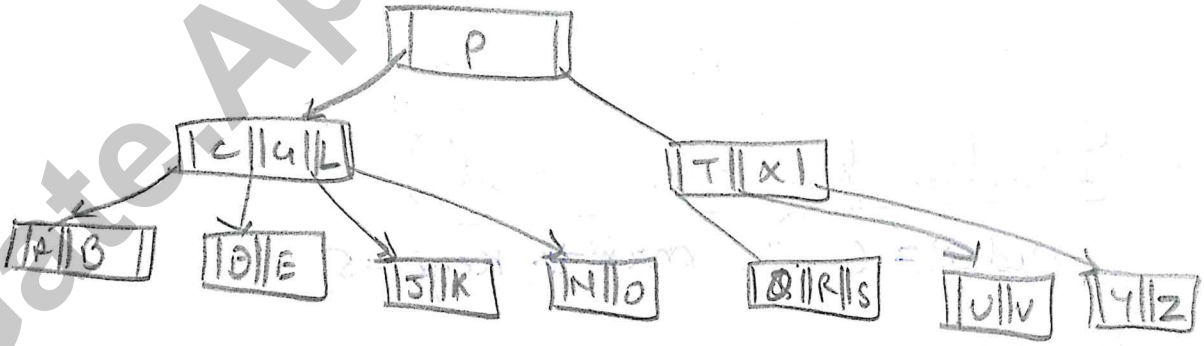
① Delete F:

It will simply remove ~~no~~ F from the leaf node. The resultant tree will be:



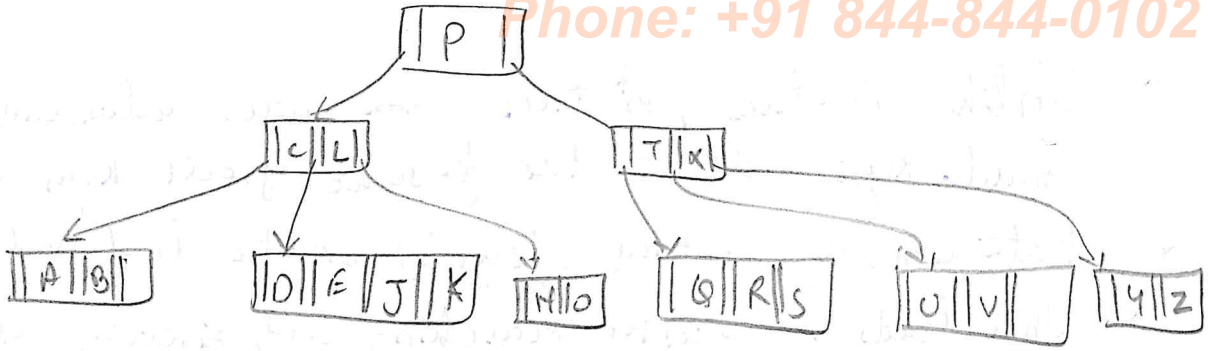
② Delete M:

∵ M is an internal node. The node before M has all values < M and node after M has all values > M. ∴ most logical value which should replace M is L.

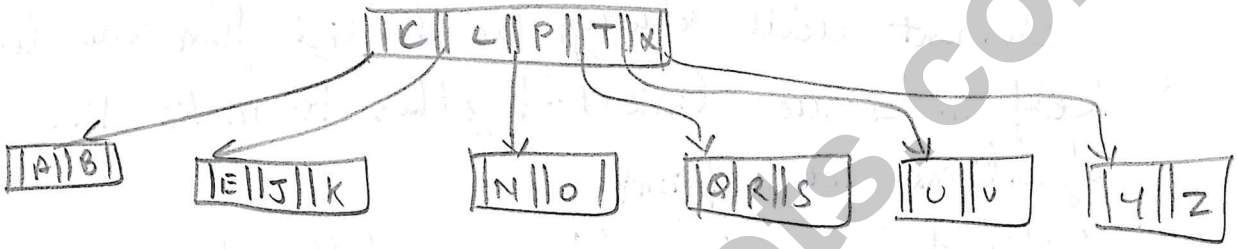


③ Delete G:

Since we can have max of keys = 5 ∴ either we can combine the node after deleting G or could do the same thing as we did in step ②



(9) Delet D:



Note:-

While inseting or deleting ~~node~~ key in B-tree or B+ tree there are lots of cases and boundary cases to be handled.

→ a B+ trees

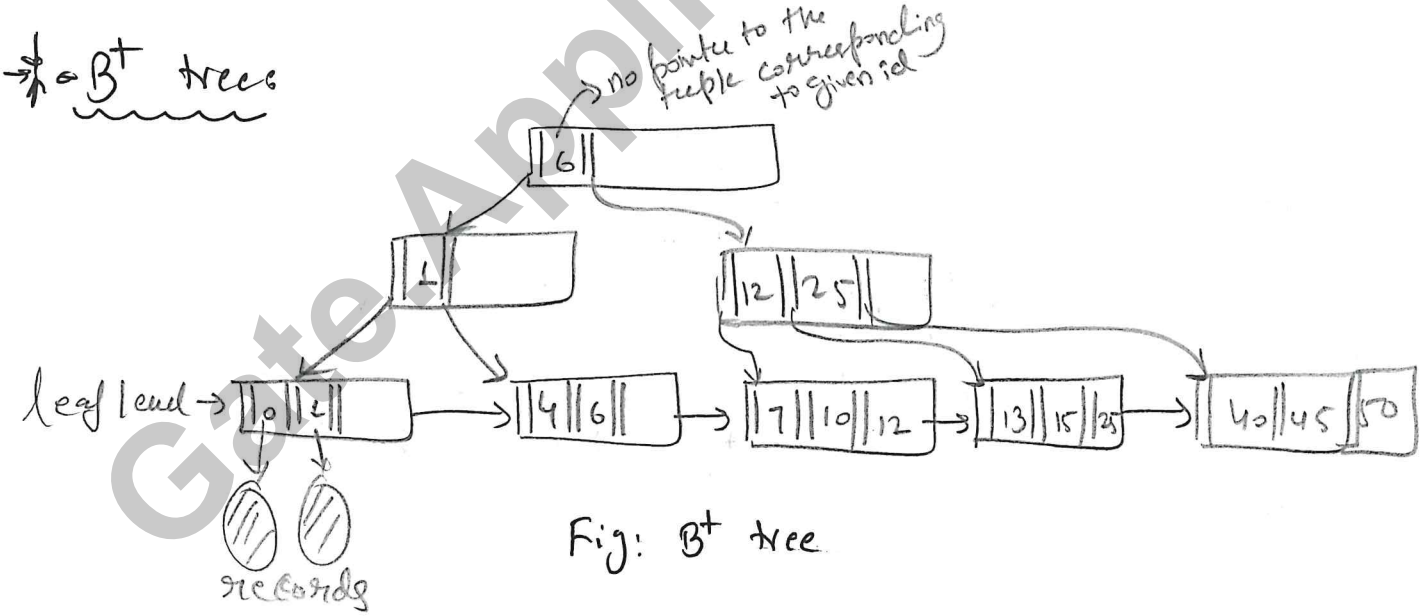


Fig: B+ tree

* Every node is a block
* Root node has ≥ 2 pointers or children
* Internal nodes have >= ⌈m/2⌉ pointers or childeens.
* All leafs must be at some level

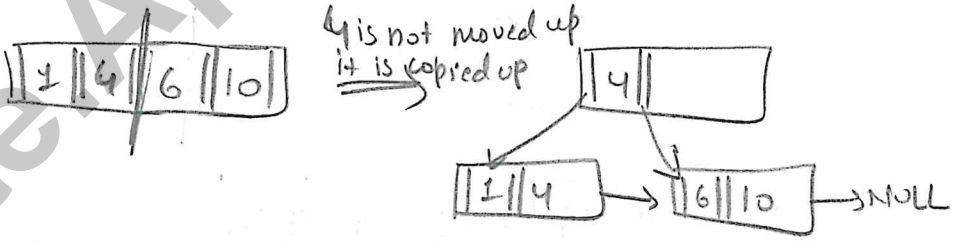
* Unlike B-trees, B^+ trees have redundant search keys (due to the presence of all keys at leaf)
* Data can only be stored (Records) on the leaf nodes.
* This leads to faster searching comparitively as data can only be found on the leaf nodes.
* Deletion will never be a complicated process since element will always be deleted from the leaf nodes.
* Leaf nodes are linked together to make the search operations more efficient.
* Order of leaf and non-leaf is different.

~ Insert into B^+ trees:

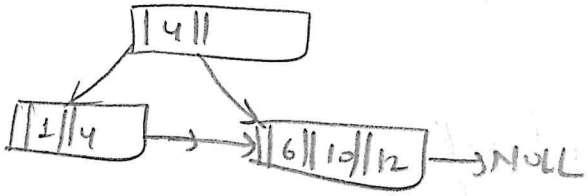
Order = 4 ⇒ # Keys = 3

① Insert 1, 4, 6:

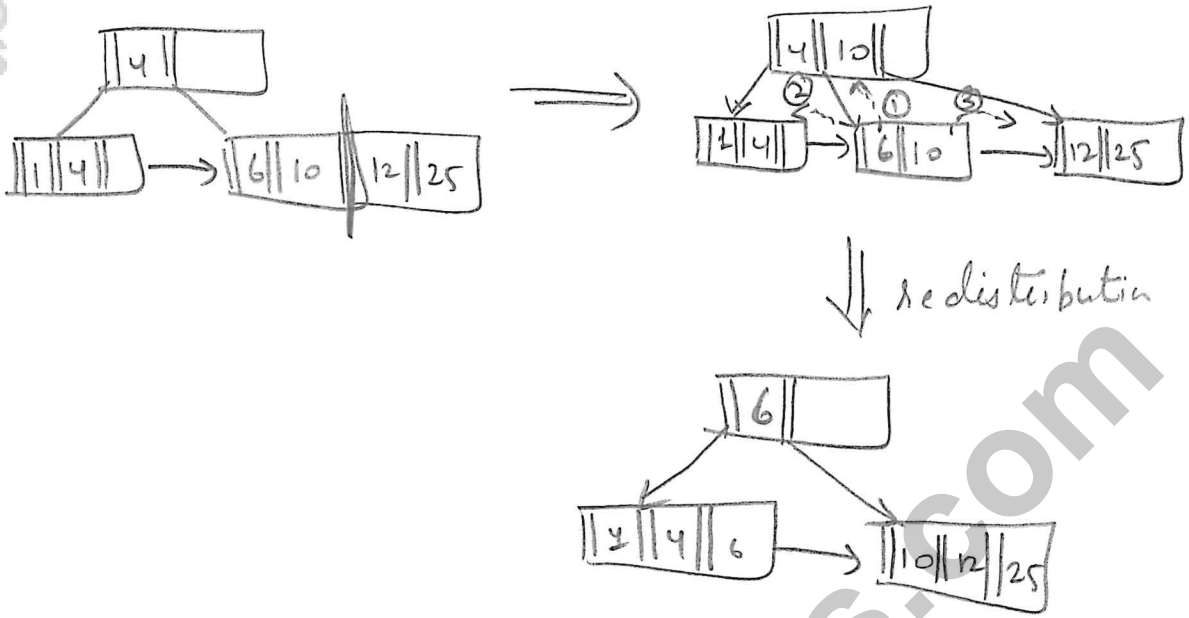
② Insert 10:



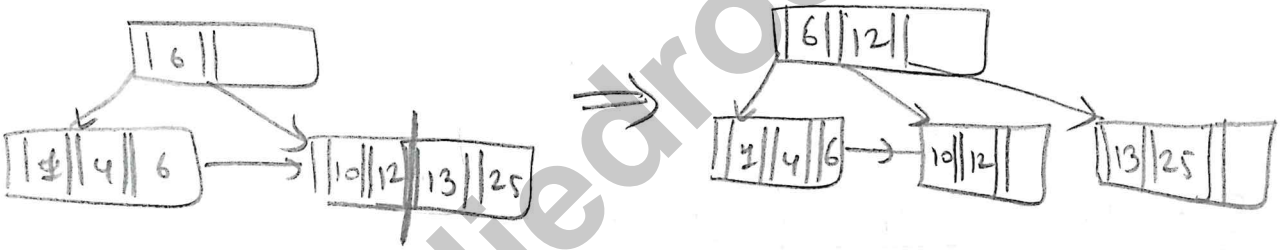
③ Insert 12:



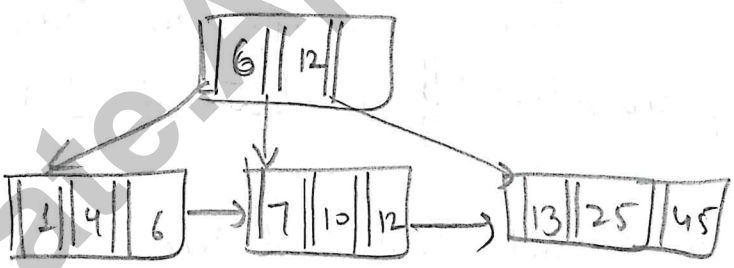
(4) Insert 25:



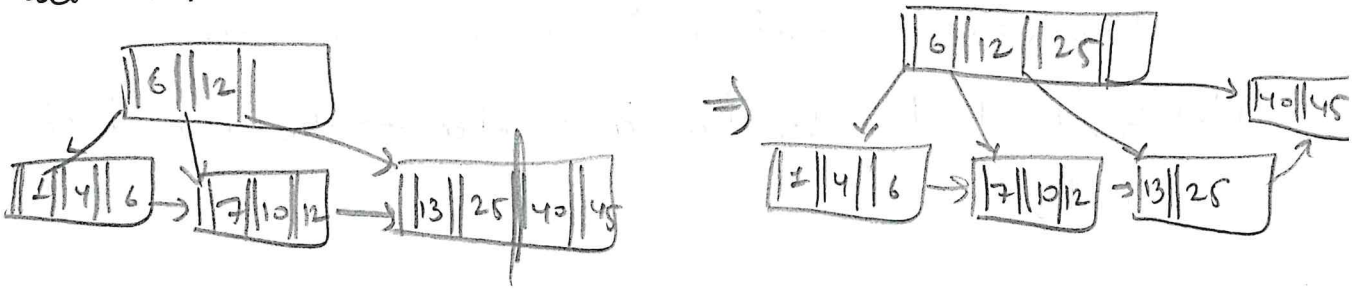
(5) Insert 13:



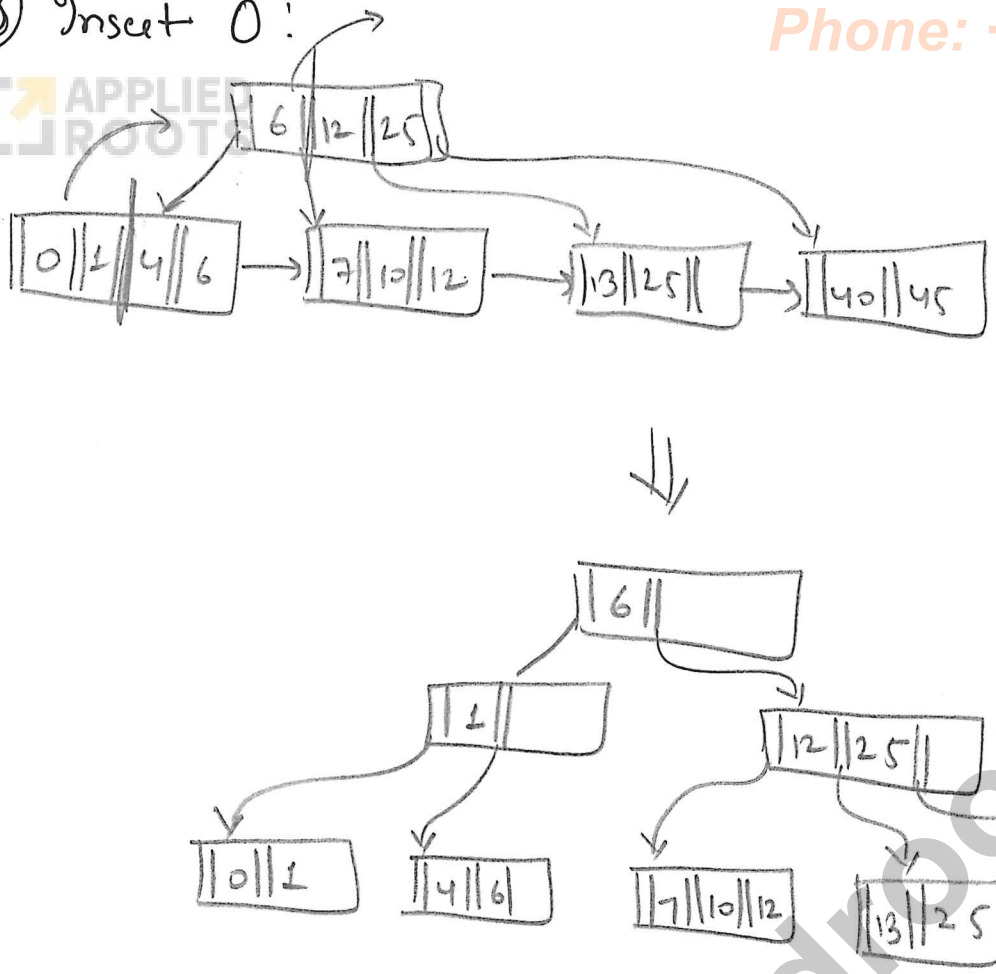
(6) Insert 7; 45:



(7) Insert 40:



(8) Insert 0 :



19.3 More solved problems:

(Q1) Which one of the following statement is NOT correct about the B^+ tree data structure used for ~~correct~~ creating an index of a relational database table?

(A) B^+ Tree is a height balanced tree

(B) Non-leaf nodes have pointers to data records

(C) Key values in each node are kept in sorted order

(D) Each leaf node has a pointer to the next leaf nodes.

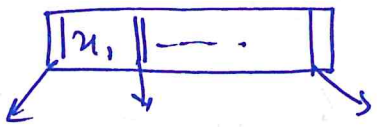
⇒ (A) 97 is correct

(B) In B+ trees we pointers at leaf node ∴ it is incorrect

Hence, correct option is (B)

(Q2) In a B+ tree, if the search key value is 8 bytes long, the block size is 512 bytes and the block pointer is 2 bytes, then the maximum order of the B+ tree is _____

⇒



Let there be m keys than order will be m+1 pointers

$BS = 512$

$P_b = 2$

$K = 8$

$\Rightarrow 8m + 2(m+1) \leq 512$

$\Rightarrow 10m + 2 \leq 512$

$\Rightarrow 10m \leq 510$

$\Rightarrow m = \lfloor 510/10 \rfloor = 51$ //

(Q3) B+ trees are considered BALANCED because

(A) The length of the paths from the root to all leaf nodes are all equal

(B) The length of the paths from the root to all leaf nodes differ from each other by at most 1.

(C) The number of children of any two non-leaf sibling nodes differ by at most 1

(D) The number of records in any two leaf nodes differ by at most 1

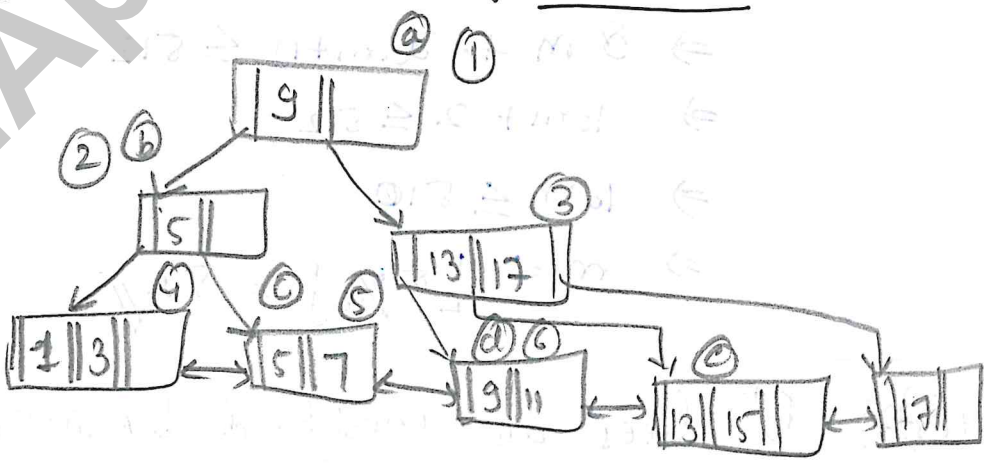
⇒ (A) is true
(B) not true
(C) not true
(D) not true

∴ Correct option is (A)

Q (Q-4) With reference to the B^+ tree index of order 1 shown below the minimum no. of nodes (including the root node) that must be fetched in order to satisfy the following query:

"Get all records with a search key greater than or equal to 7 and less than 15" is: _____

(A) 4
(B) 5
(C) 6
(D) 7



⇒ We want minimum no. of nodes fetched while searching ≥7 and <15. On following search based approach will result in total 6 disk-read.
(Here, ①, ②, ③, ④, ⑤, ⑥ are disk reads)