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 (1.2) 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 user 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:

Column 1 / field 1 / attribute ←

→ column 4

CUST_ID	CUST_NAME	CUST_ADDR	CUST_PIN	
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

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, thafor, 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)

refers to

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

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:-

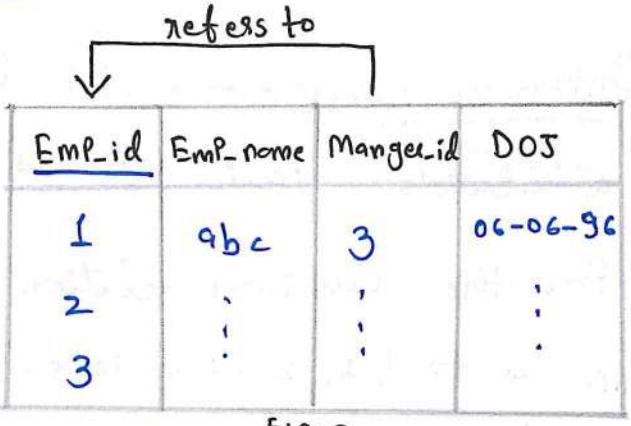


Fig: 3

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 Colemns/attributes 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 ~~relati~~ 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 conditions.

(a) On delete no action:- On deleting 1st row from ~~refere~~ CUSTOMER, no changes are made in PURCHASES. Though refesential 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 ← { a_1 b_1 } 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 refencing 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 voilation.

(ii) On deleting :- On deleting a row from refencing relation will make/brings no change in the refenced relation

(iii) On modifying/editing : We need to check for any voilations regarding foreign key constraint in refenced 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 manager set the constraint as ON DELETE CASCADE.

S.No.	A_1 (foreign key)	A_2
1	2	4
2	3	4
3	4	3
4	5	2
5	7	2
6	9	5
7	6	4

⟹ Since, the constraint is ON DELETE CASCADE.

Therefore, on deleting (2,4)

* Initially 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 refing 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 delete 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 $\cup$ ϕ

(it is candidate key given in question.)

A_2

A_3

$\vdots$

$A_2 A_3$

$\vdots$

$A_2, A_3 \cdots A_n$

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.

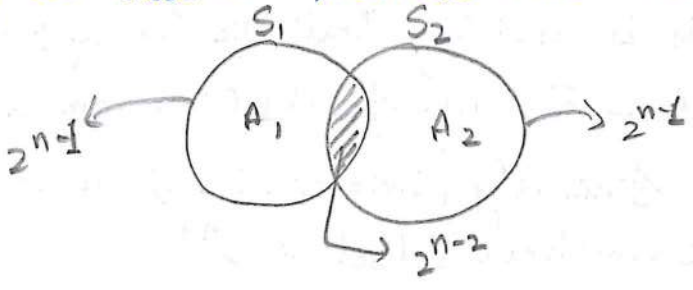
Hence, # super keys given candidate key as $\{A_1\}$ are 2^{n-1}

(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	$\cup$	ϕ	
		A_2	
		A_2, A_3	2^{n-1} Superkeys
		$A_2, A_3 \cdots A_{n-1}$	
A_2	$\cup$	ϕ	
		A_1	
		A_1, A_3	2^{n-1} Superkeys.
		$A_1, A_3, \cdots A_{n-1}$	

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

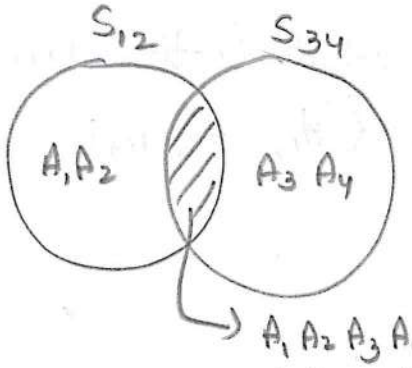
$= 2 * 2^{n-1} - 2^{n-2}$

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

(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 $= 2^{n-4}$

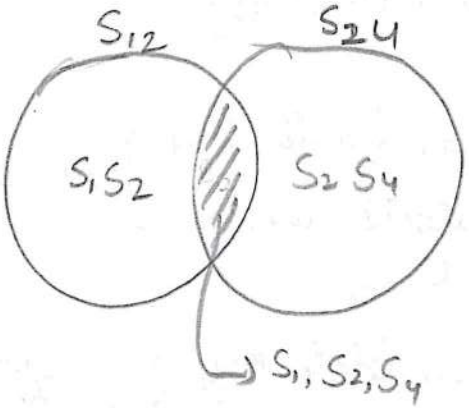
Therefore, no. of superkeys $= 2^{n-2} + 2^{n-2} - 2^{n-4}$

(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}) $\} = 2^{n-2}$

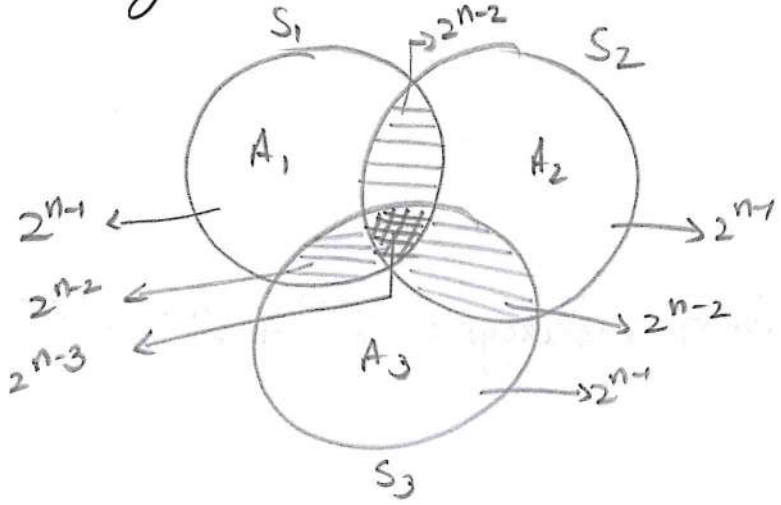
For $\{A_1, A_2, A_4\}$ no. of subsets will be $= 2^{n-3}$



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:

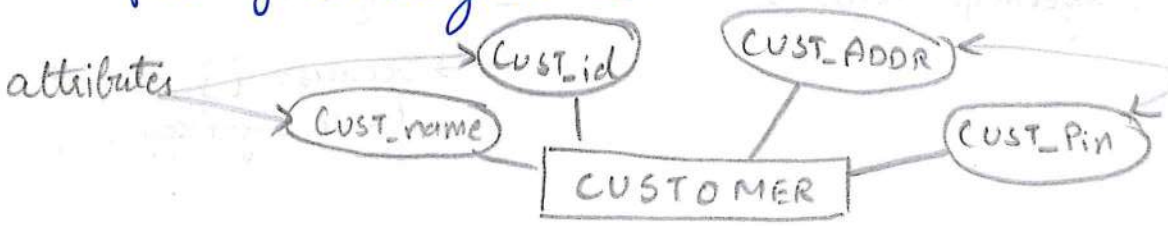


Fig: 5

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: 5, CUST_ID is the key attribute.

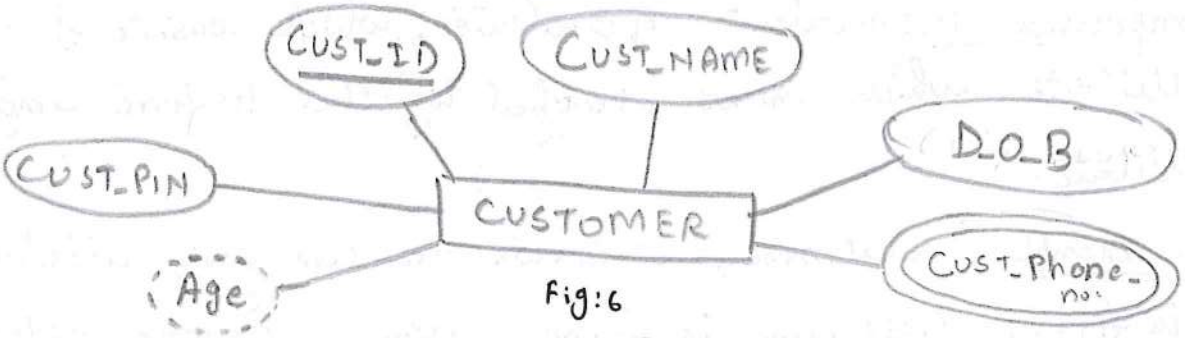


Fig:6

→ 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.

For example :- 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.

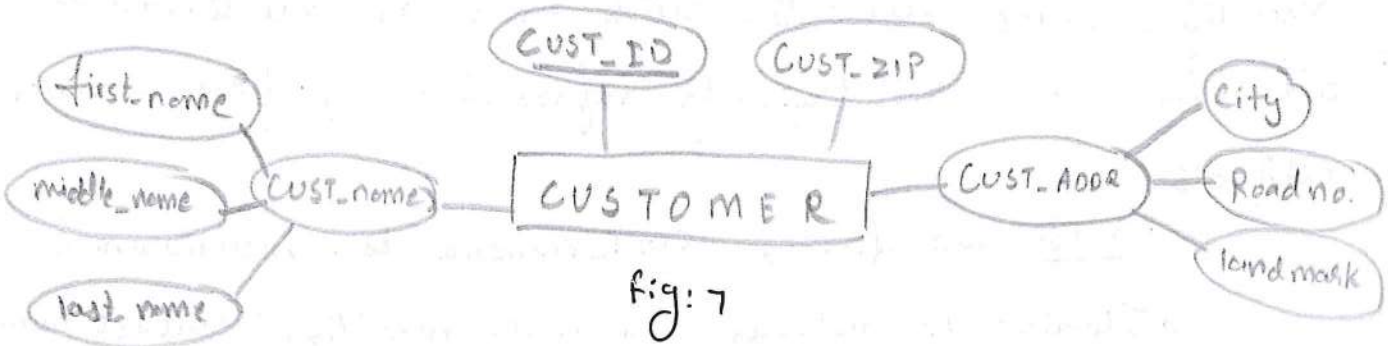


fig: 7

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~~ buy/s a product/s." 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:

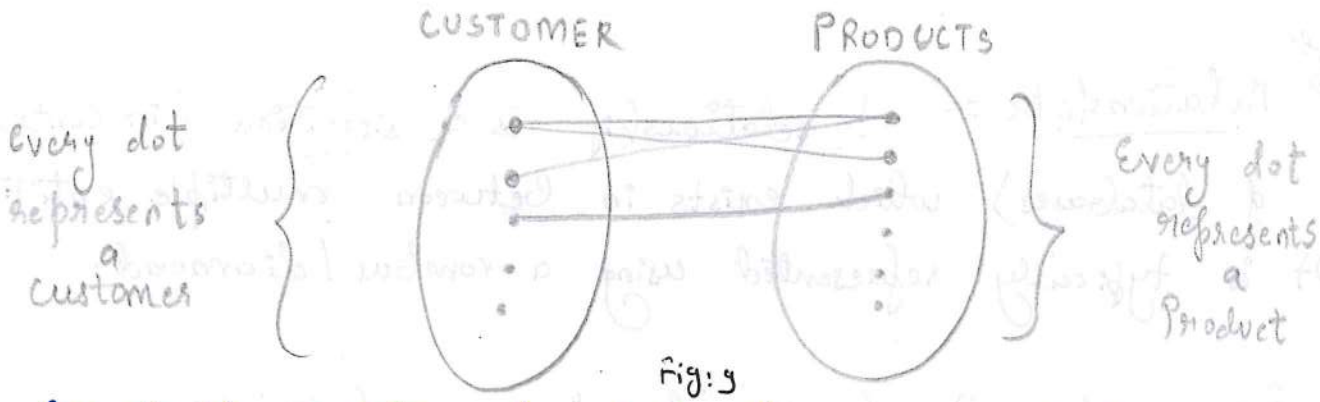


Fig: 9

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)

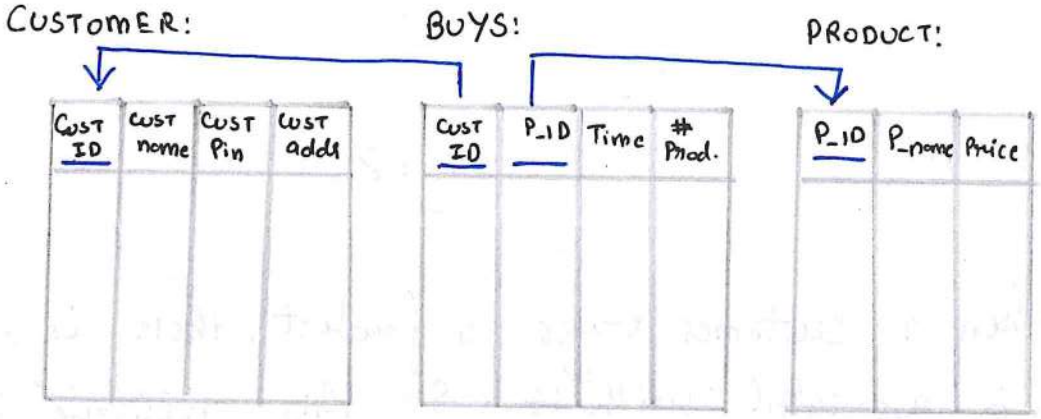


Fig: 10

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"

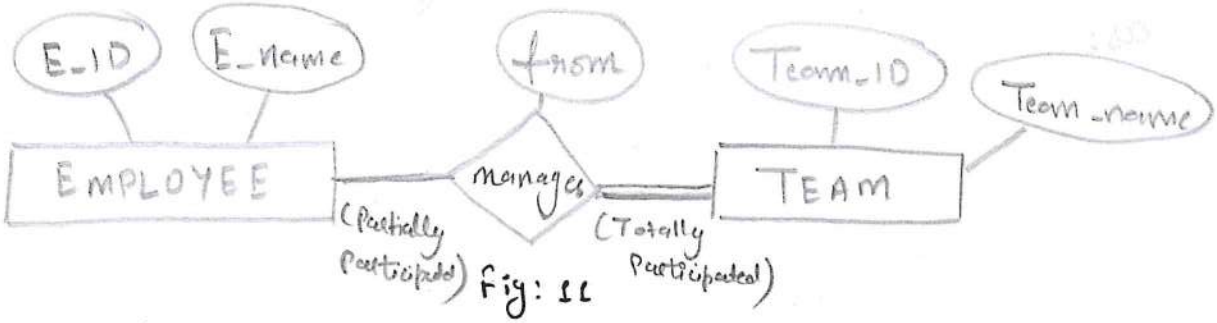


Fig: 11

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 participated 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:

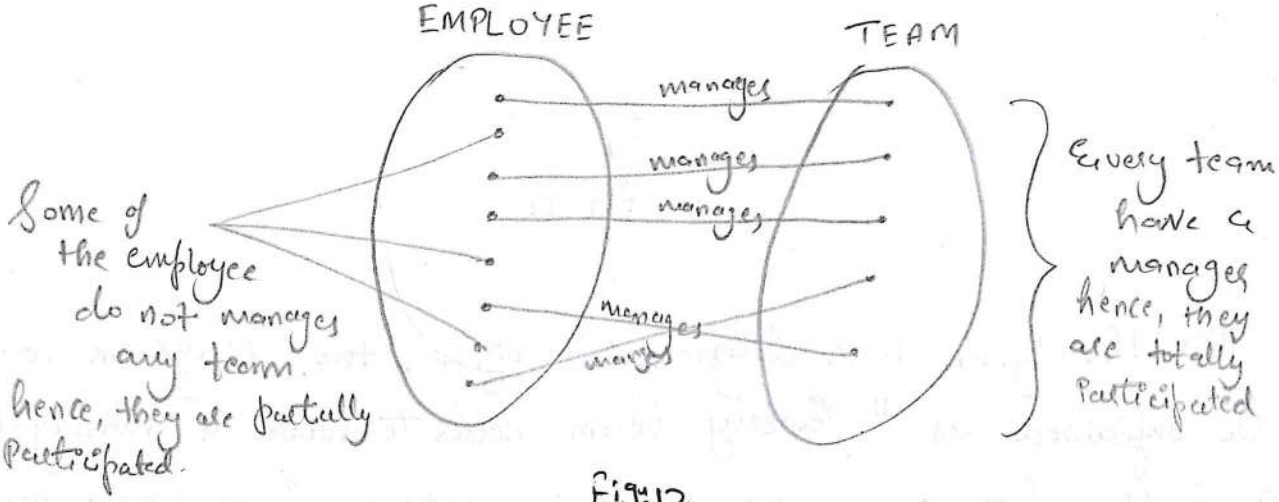


Fig:12

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

II Cardinality of Relationships and Constructing (2.2) 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)

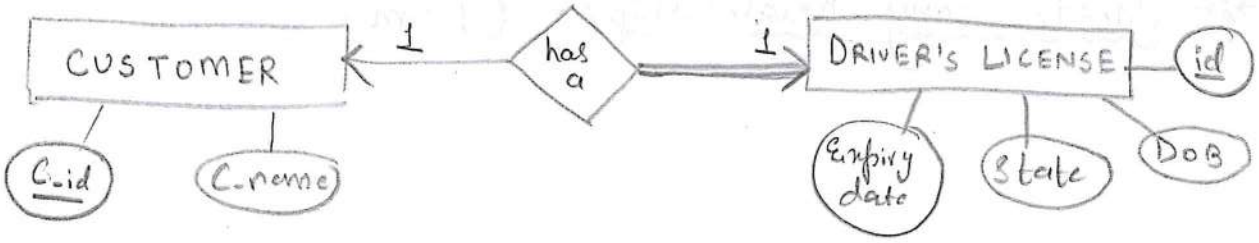


Fig: 13

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, defics the set theoretic fersfective of E-R diagram given in Fig: 13.

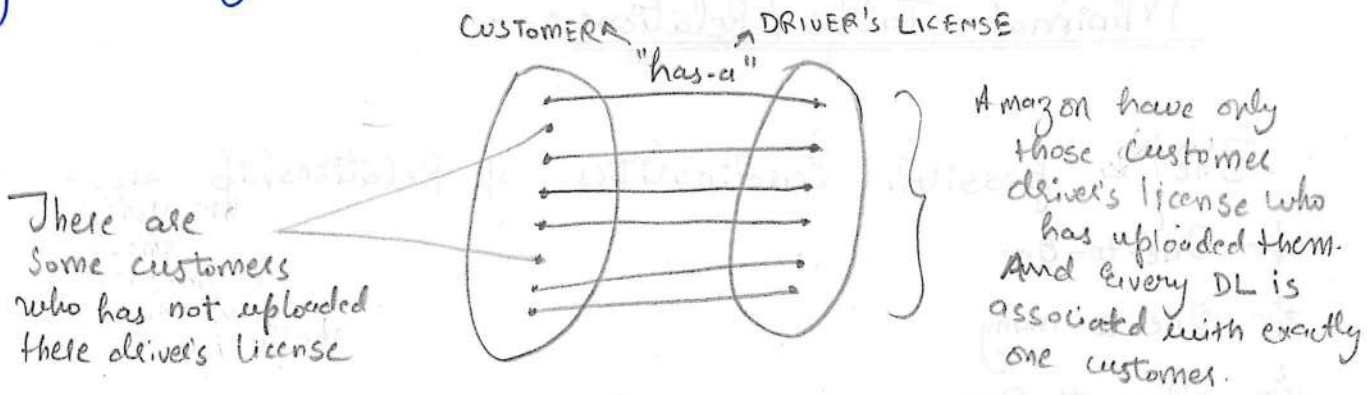


Fig: 14

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)

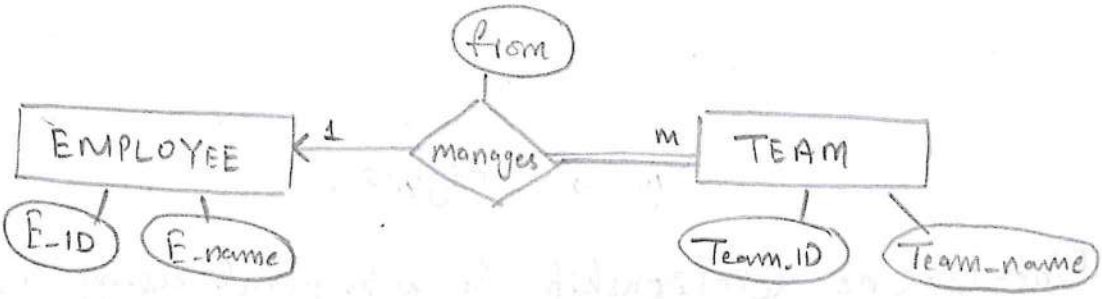
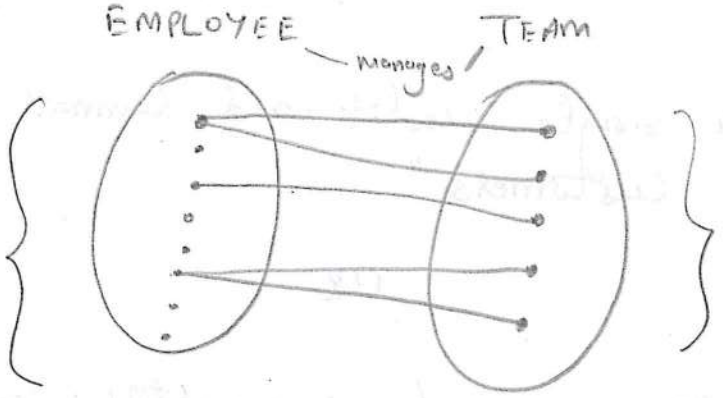


Fig: 15

Single or double line by default defics the many fart 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 defics the set theoretic fersfective 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)

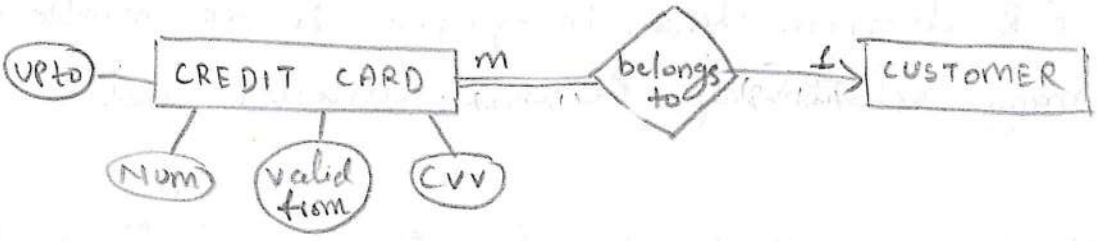


Fig:17

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 there 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 :-

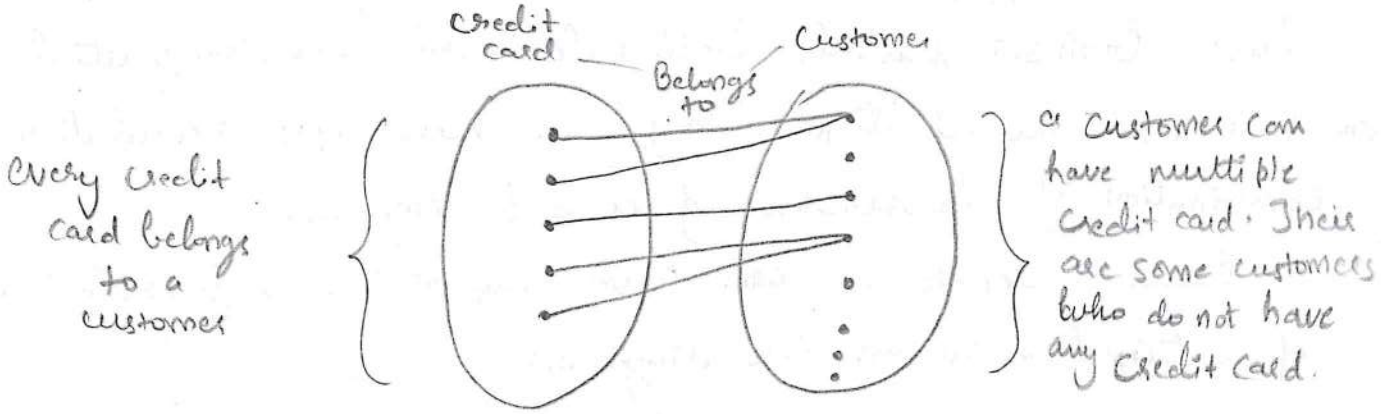


fig:18

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 :-

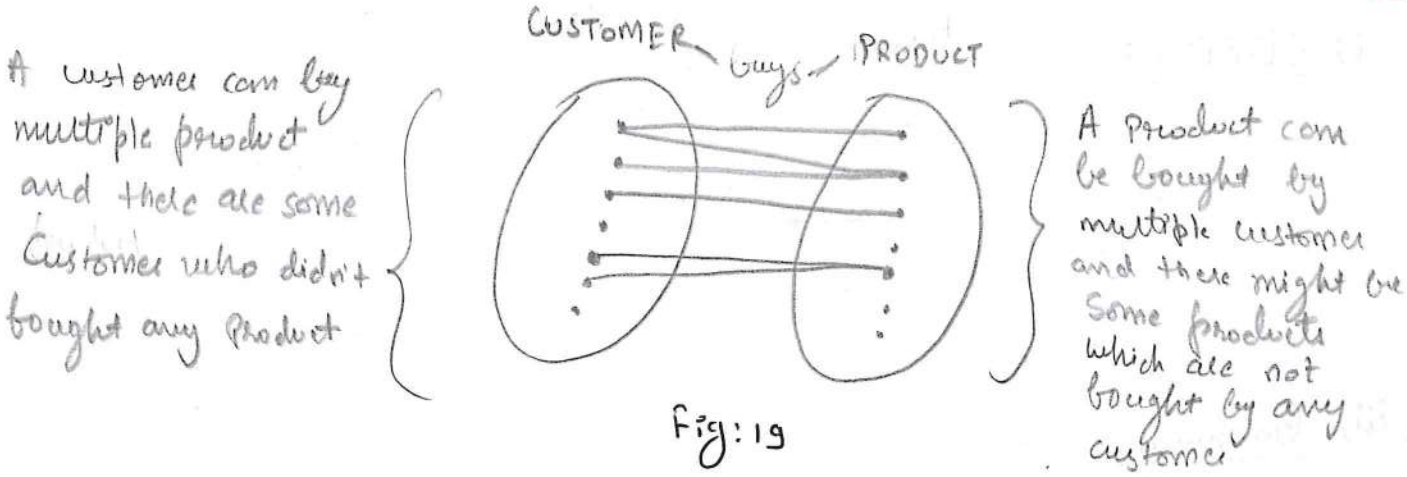


Fig: 19

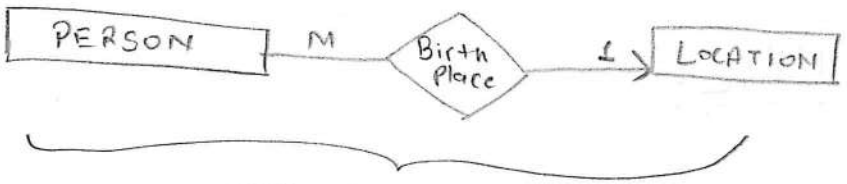
→ Alternative Conventions :-

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

→ ⇒ To depict 'one'

—— ⇒ To depict 'many'

for example: For many-to-one :-



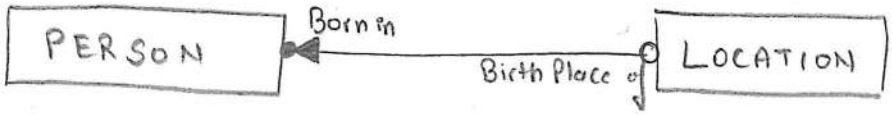
This is one of the way to represent cardinality.

Other popular alternative conventions are :-

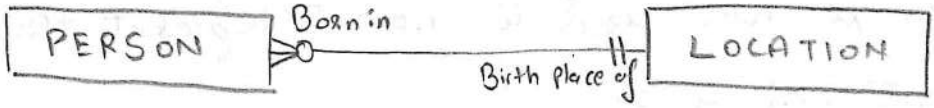
(i) Chen :

(ii) IDEFIX:

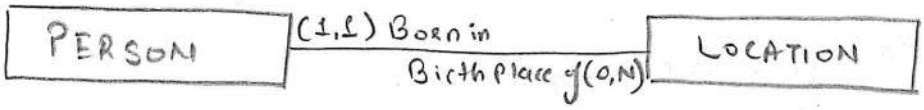
(iii) Bachman:



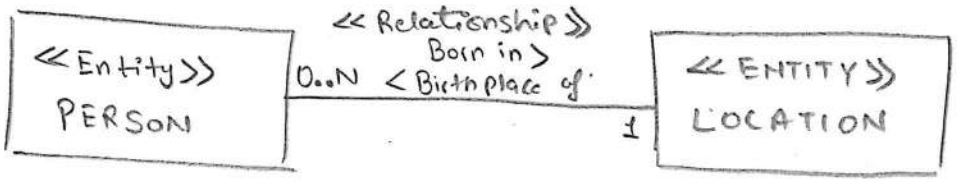
(iv) Martin / IE / Crow's Foot:



(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 ? 33

Imp

→ Constructing tables from E-R Diagrams :-

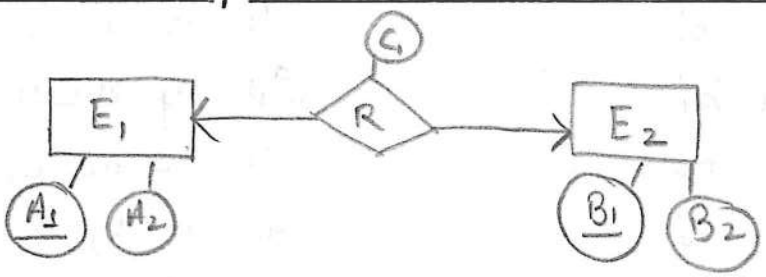
Given an E-R diagram we need to look for cardinality, participation, keys etc in order to construct tables from that. Along with this there is some set of rules defined for constructing a table / relation. These are as follows:

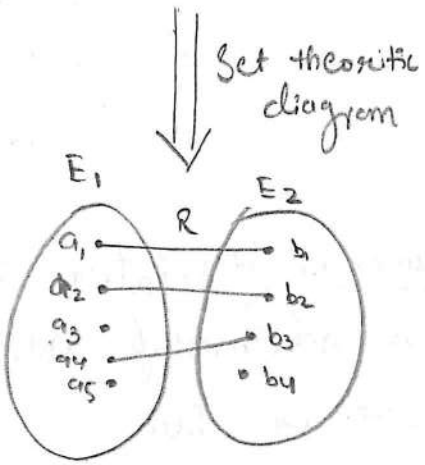
[1] We need to minimize the number of table / relations.

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 :-





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)

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'

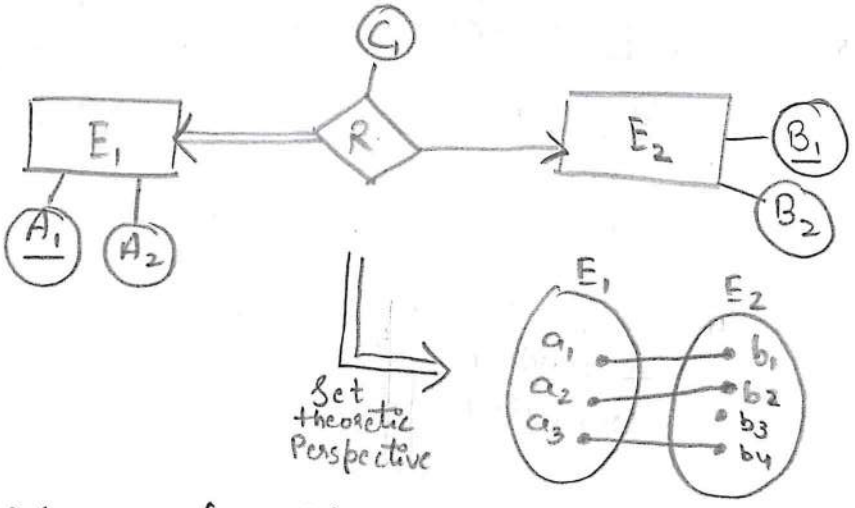
Key values are not NULL

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

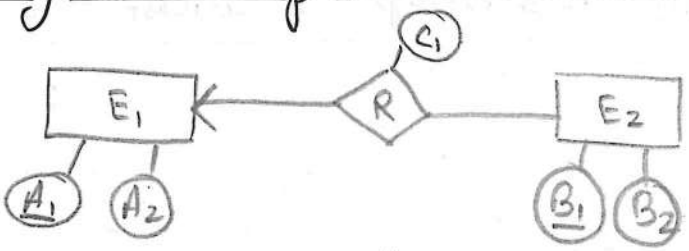
Cannot become Key

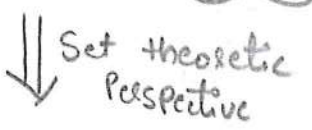
B_1 can become Key

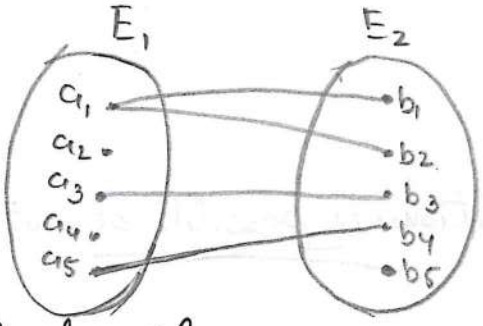
As, we can see there is no NULL value for attribute B_1, therefore, it B_1 can become the primary of this table.
Hence, minimum no. of relation for given condition is 1.

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:-







⇒ 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 :-

refers to

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:

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:

refers to

Table 1:

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

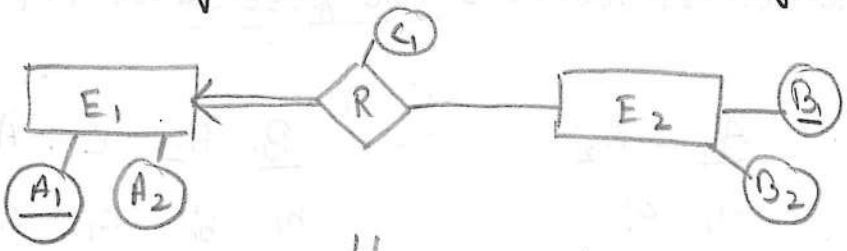
Table 2:

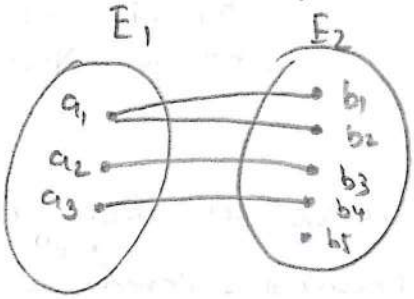
B_1	B_2
b_1	b_1'
b_2	b_2'
b_3	b_3'
b_4	b_4'
b_5	b_5'

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

× ×

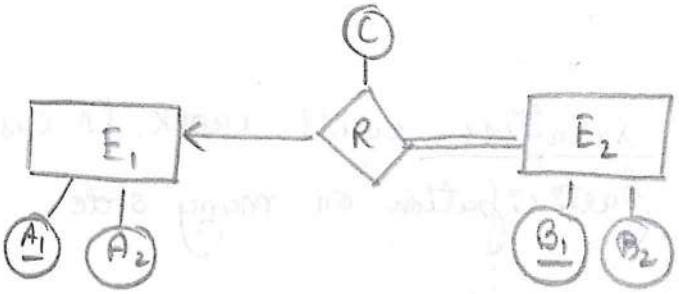
↓

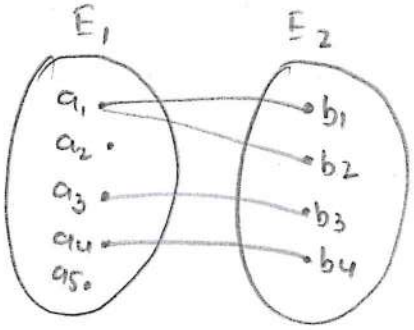
This could be the Primary key

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:-

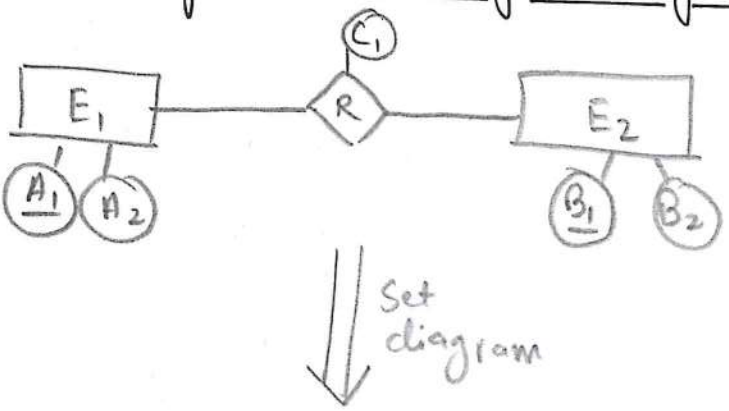
A_1	A_2
a_1	a_1'
a_2	a_2'
a_3	a_3'
a_4	a_4'
a_5	a_5'

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

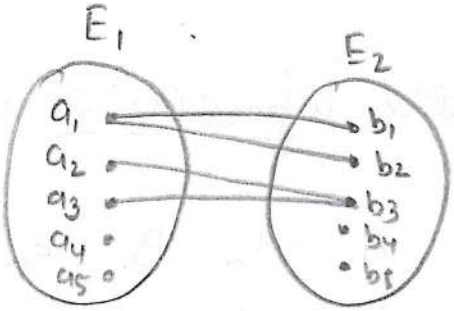
Hence, 2 relations will work in case of one-to-many and total participation on many side.

(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, ~~one~~ 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)

A_1	A_2

2)

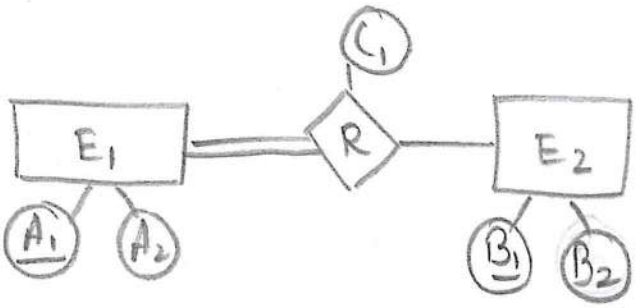
C_1	A_1	B_1

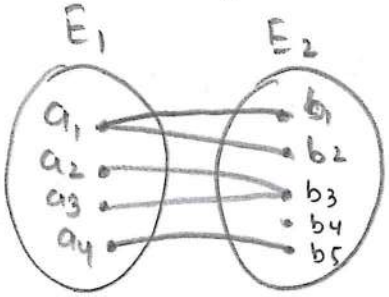
3)

B_1	B_2

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'

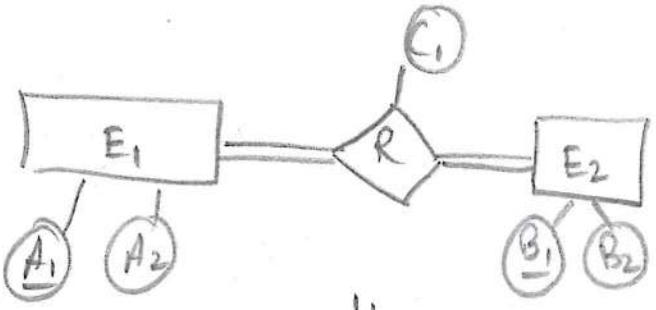
B_1	B_2	C_1	A_1
b_3	b_3'	c_3	$\{a_2, a_3\}$

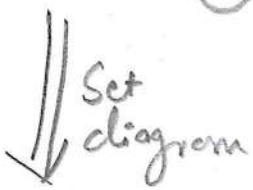
X

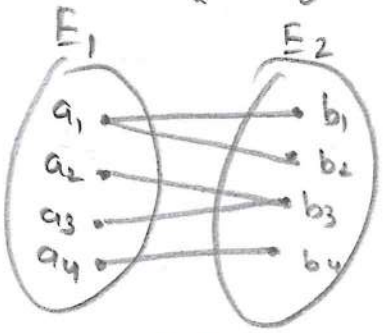
Hence, it is not possible to have 2 relations. and therefore we need 3 relations ≡ minimum for the given case

Case 3:

Many-many
+
Both side
total
Participation.







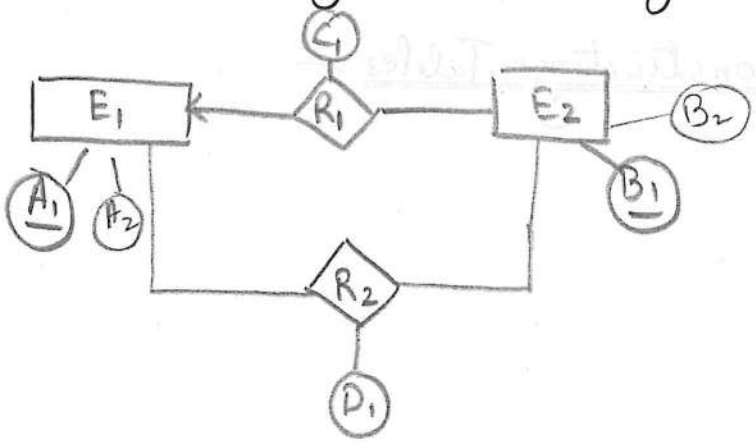
As we can see here 1 relation is possible in minimum case with $\{A_1, B_1\}$ as the primary key. But it will result in lot of redundancy of data.

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?

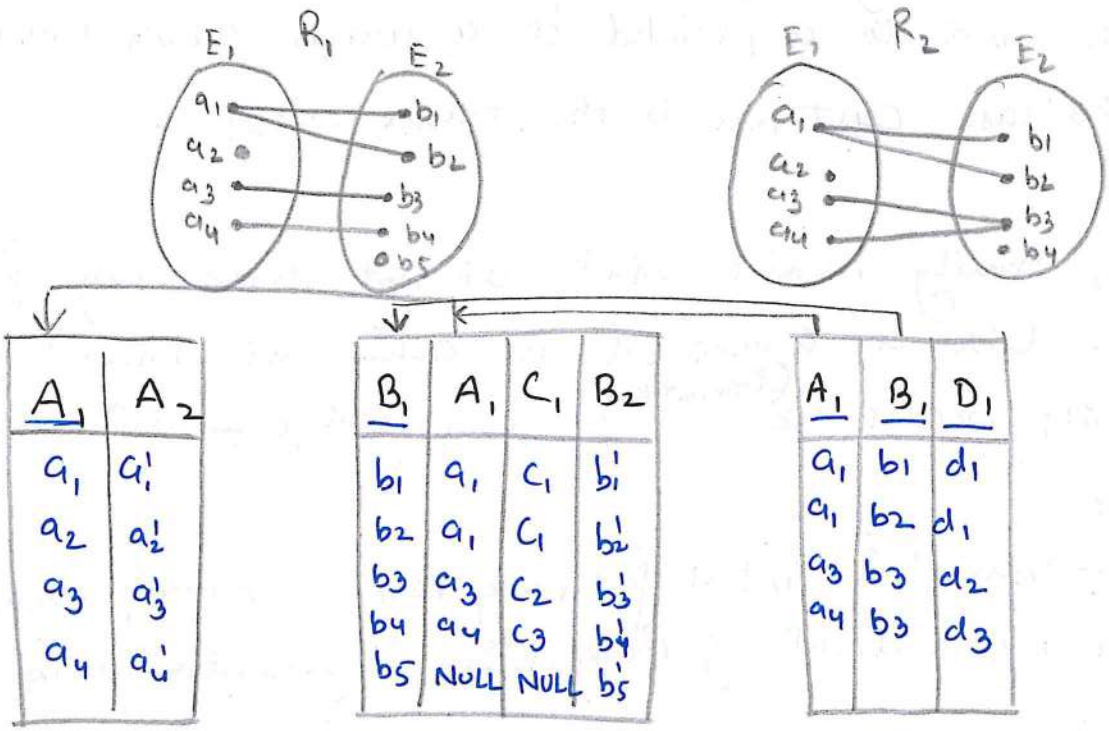


⟹ ~ We can trivially do it in 4 relations.

~ 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 there 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 :-

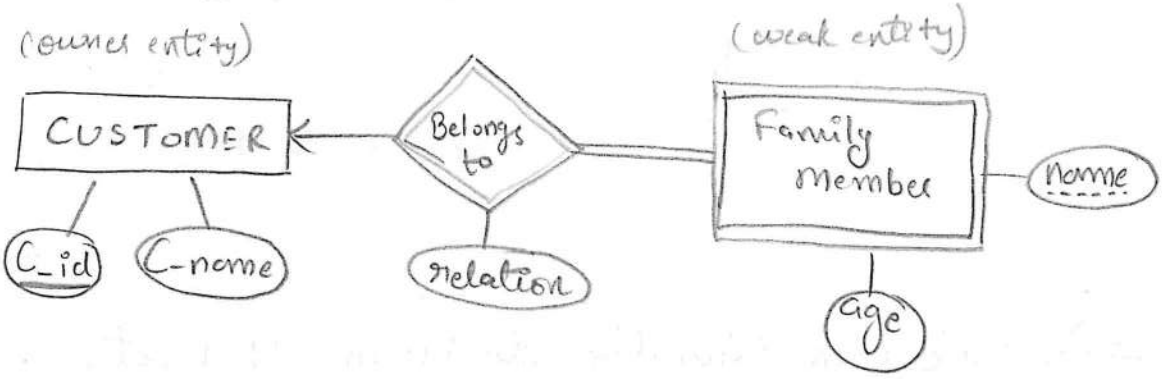


Fig: 20 Weak entity E-R diagram.

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 / rohmbus 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?

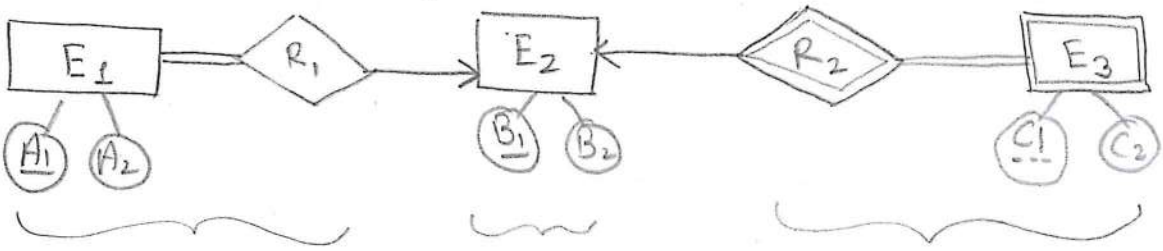


Fig: 21

⇒ 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:

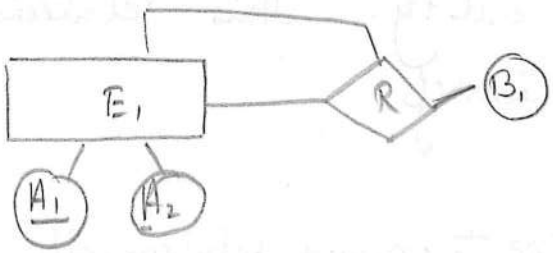
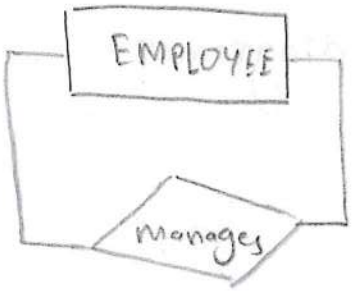


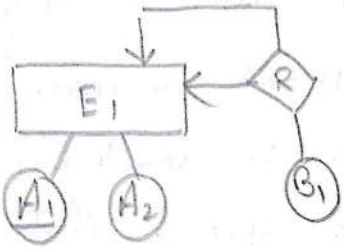
Fig: 21 Self-referential relationship

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

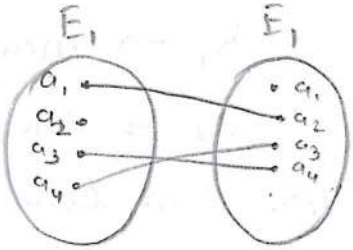
for example:- "An employee manages another employee" is a self-referential relationship.



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



Set diagram ⟹



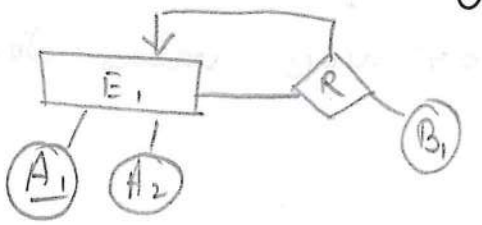
⇒ Check for 1 relation possible or not:

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

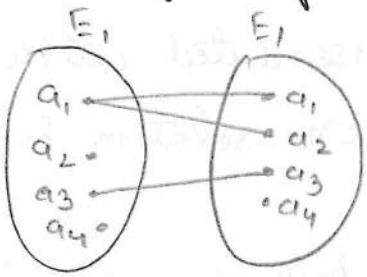
NO NULL ✓ Derivable key ✗

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

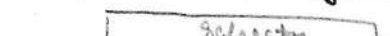
Case :2: One-to-many + Partial-partial participation:



Set diagram ⟹



⟹ Check for 1 relation is possible or not:

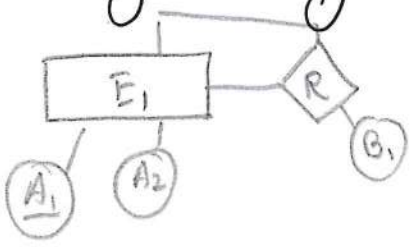


A_1	A_2	A_1'	B_1
a_1	a_1'	a_1	b_1
a_1	a_2'	a_2	b_1
a_3	a_3'	a_3	b_3
NULL	a_4'	a_4	NULL

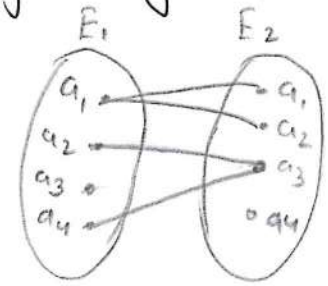
Here, we cannot make A_1 as our primary key because in set diagram we can see a_1 is related to more than one value of itself. Therefore, we will have A_1' as our primary key.

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 (2.4) Solved Problems: (PYQs)

(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?

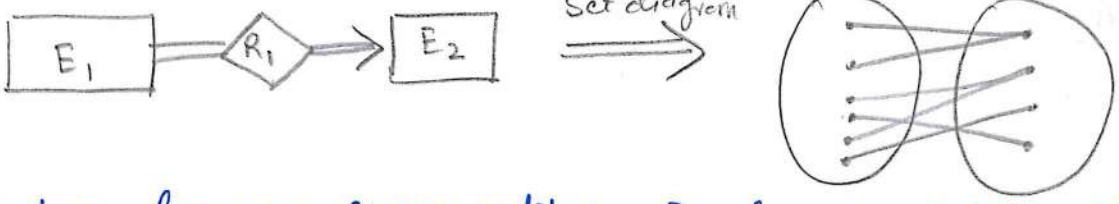
(A) Every entity in E_1 is associated with exactly one entity in E_2

(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?

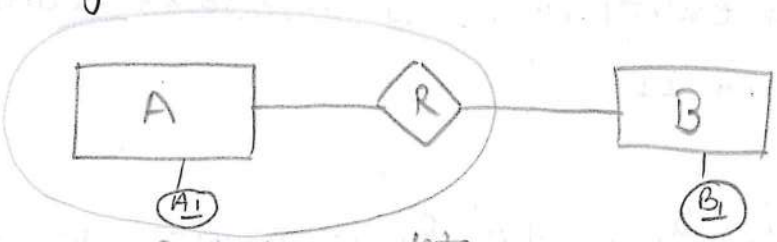
(A) Relation R is one-to-many and the participation of A in R is total.

(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.

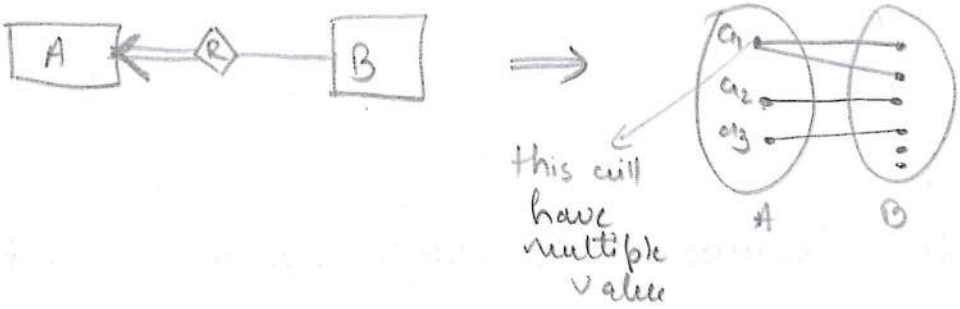
(D) Relation R is many-to-one and the participation of A in R is partial.

$\Rightarrow$



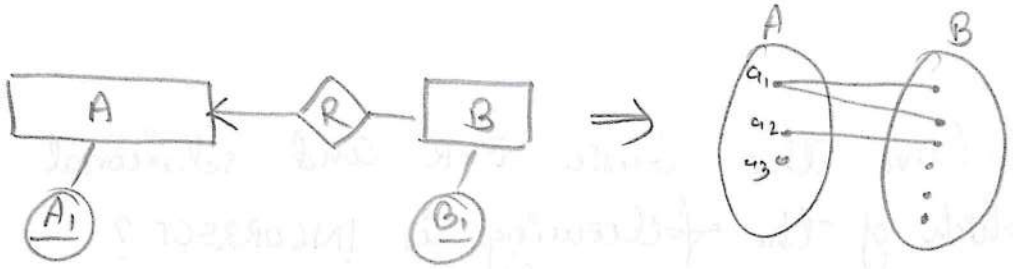
In which condition we can merge them?

Case A:



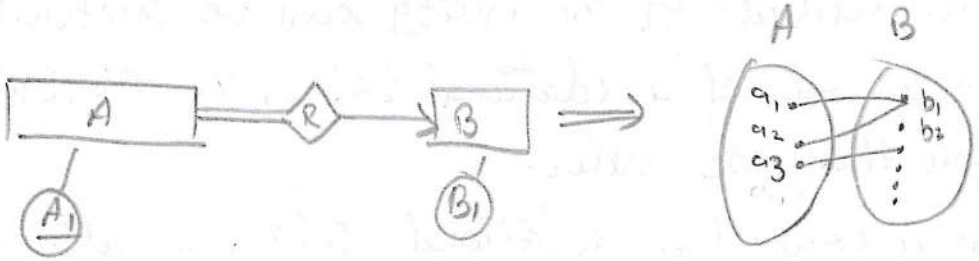
As we can see entity A will have multiple values of 'a1'. Therefore, we cannot combine R with A in this condition. Hence, this is false.

Case B:



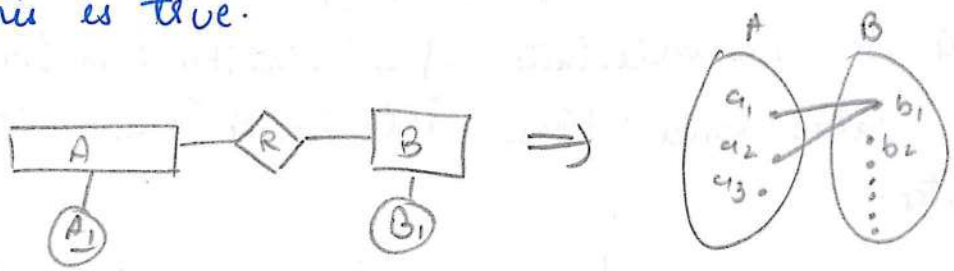
In case B also, there is the same problem. 'a1' will have multiple values. Hence, R could not be merge with A.

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) Give 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'.

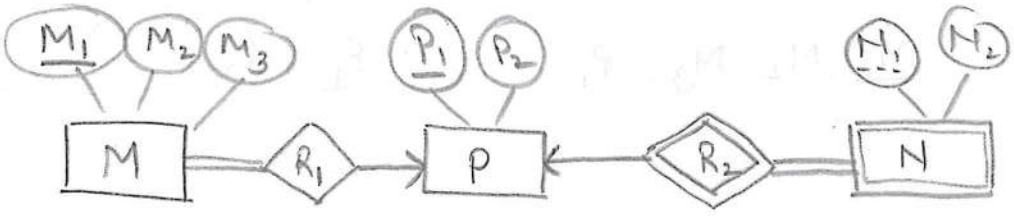
(C) It is incorrect, a tuple/row of a table cannot store more than ~~value~~ one value in a cell.

(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

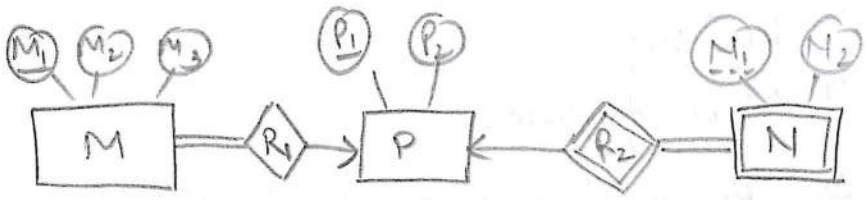


⇒ If we recall, it is same/similar to Fig: 21 problem.
- option (C) and (D) are the trivial option.
- We cannot do it in 2 relations because neither M and P could be combined together nor P and N could be combined.

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) ~~occup~~ 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 ~~is~~ 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 $A \times B$

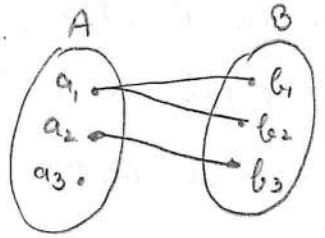
ie. Given set $A = \{a_1, a_2, a_3 \cdots a_n\}$

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

$R \subseteq A \times B$

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

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



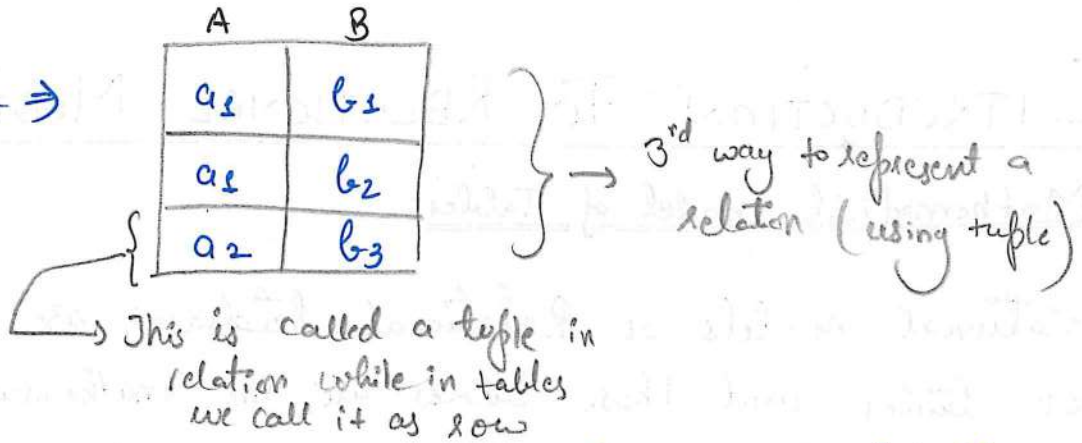
→ 1st way to represent a relation

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 $A = \{a_1, a_2, a_3 \cdots a_n\}$ 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, Cgif, 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. If one decides to delete record (3, 8) from table T_1, the number of additional records that need to be deleted from table T_1 is _____

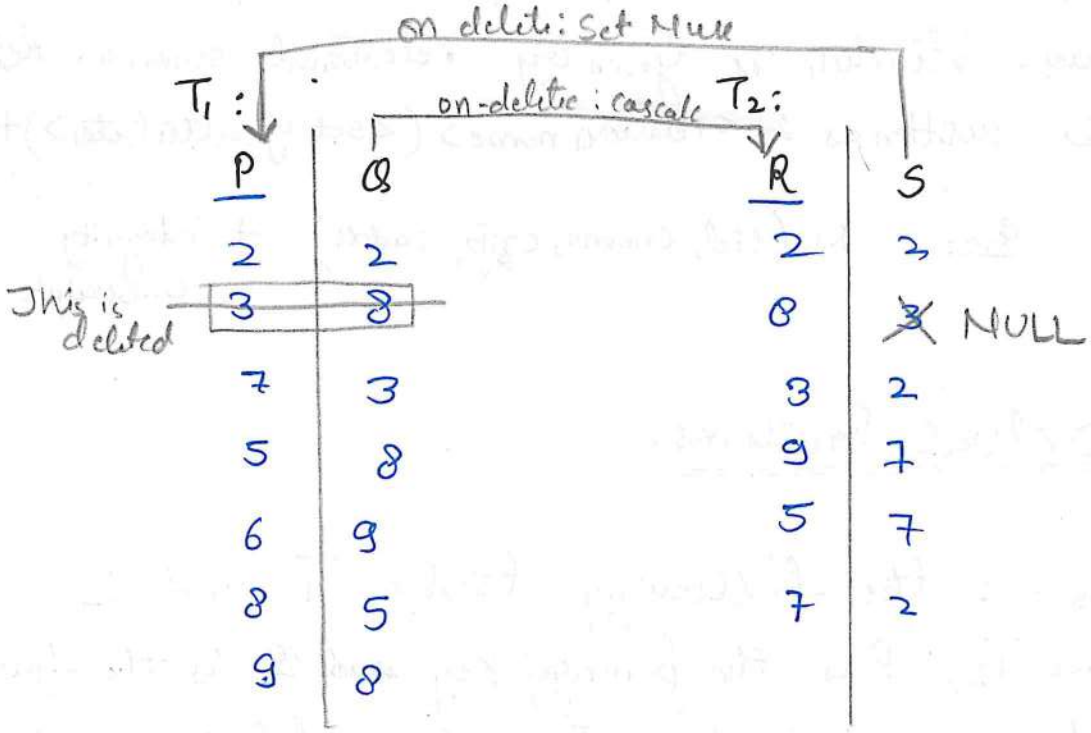
T_1:

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

On deleting (3,8) in T_2, only ~~some~~ 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:

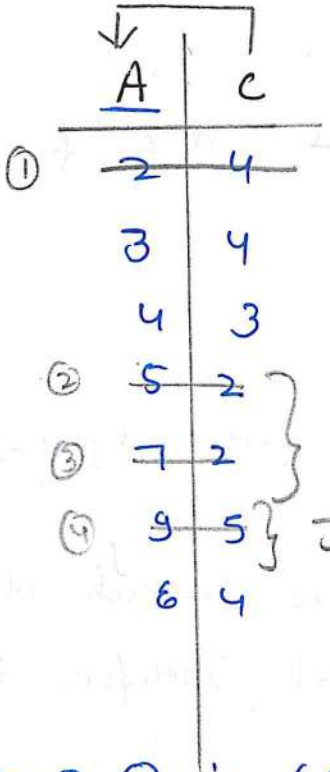
(A) (3,4) and (6,4)

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

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

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

⇒



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

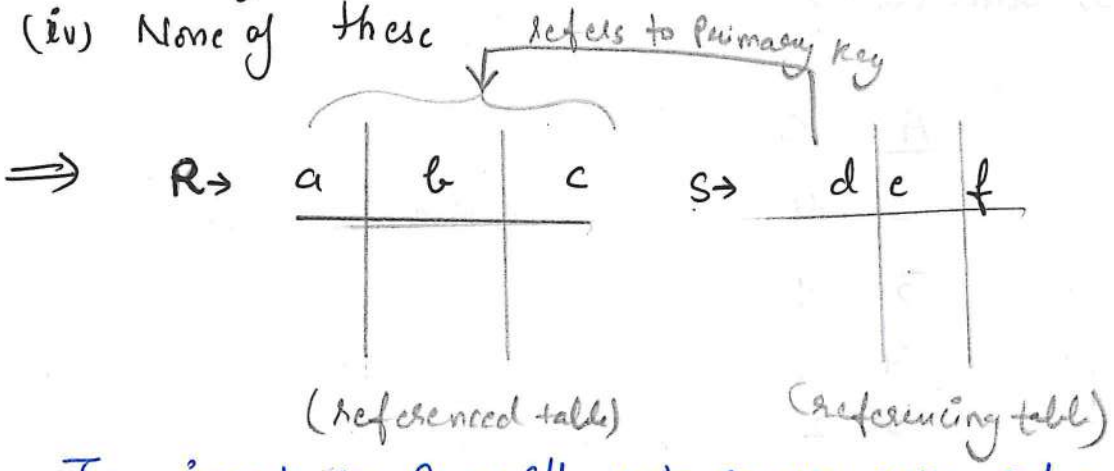
Which of the following can cause violation of the referential integrity constraint above?

(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)

(Q4) 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

(Q5) 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 super key 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 super key
2. It is not a ~~super~~ super key 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 ~~as~~ possible superkey. Hence, it is correct.

Therefore, Correct option is (A)