DATABASE MANAGEMENT SYSTEMS (DBMS).

Starting date :-

~~28th July 2022.~~

7th August 2022

Lesson 01

28/08/2022

Data :- Collection of facts / Numbers.

→ Database :- The collection of data, usually referred to as data Bases, contains information relevent to particular enterprise.

⇒ Earlier (30-35 years Back) :- Data was eurything. collection of data — ✓

→ if you had the data :- You are a leader!

But Now a days the trend is :-
data processing is eurything!

↳ You have the data, But how you are managing the data so that you can makey money out of it. → Now A days that is what important → Data Processing!

→ database :- collection of all the necessary information, facts related to one particular thing, or an enterprise, etc.

DBMS :- Scientist have ~~to~~ thought to organise the huge data in such a beautiful manner so that user accesing that data should not face any problem and it should be amazing experience for the user while accessing the data!

↳ So Scientists have come up with an idea to make a tool which will manage & organise the data ~~in~~ according to our requirement.

and that tool is → DBMS!

So Data Base Management System (DBMS) :-

a tool or a software which is a collection of inter-related data & set of programs to access & manage & organise those data.

→ manages all the inter-related data so that the users can access the data in very easy manner & with amazing experience.

Goal of DBMS :-

1. Providing a way to store and retrieve database information that is Both convenient and efficient → In less time

   means the user should not face any difficulty to retrieve the Data.

2. Ensuring the safety of information.

   ↳ providing security

Hence :- " Information is important for most organisation ".

so as a CSE person :- Our responsibility :- develop a large Body of concepts & techniques for managing data.

↳ develope those all reqd tools to process & Manage data.

applications of DBMS :- 1. Needed for Data science, and Machine learning also.

2. Organisations like Unacademy (Ed-tech), fintech companies, Amazon, Microsoft, Google, FB, apple, etc. etc.

3. Social Medias. 4. E-commerce website

5. Banking 6. Sports info websites

7. Blogs 8. Stock Market 9. OTT 10. Schools, hospitals, medical, etc.

11. News portal 12. Railways 13. Govt. offices, etc, etc.

- - - - - - - ∞ applications are there!

But why DBMS ?? → and why Not Just storing the data in just a file.

↳ Just because we human beings Need everything related to data upto the perfection :-
the experience, security, efficiency, everything we need in the perfection.!

Hence :- it has many advantages over conventional data storage techniques.

→ So the disadvantages of storing the data into file Systems, rather than storing data into the DataBase Systems :-

1. Data Redundancy & Inconsistency :-

→ Multiples files can have same data → So the data in files are inter-related to each other.

and due to absence of linking b/w the files :-

Inconsistency happens :- like updating in one particular file will not update the other interrelated file because of absence of linking b/w them.

and in case of DBMS :- This linking b/w the inter related data is perfectly managed. you just change once in one data file/table then the same updation will be done everywhere.

So if interlinking b/w the data is present :-

Same updation / or any same activity will be done ~~all~~ in all files/tables/data.

↳ and this is what DBMS provide us!

#02. difficulty in accessing the data.

by keeping the data in files :- to access any specific content :- S/w developer have to write a prog. to fetch the content of the file.

So this was just one time → hence every time the S/w developer has to write a prog. to access the file contents

↳ which is actually a very tedious task to do it.

and for accessing the data in case of DBMS :-
you just have to write one query!
and that's it → everything you want in seconds!
↳ that s/w will make it very easy.

#03 data isolation :- If data is into different types of files :- then we want to have the corrective data from them :- writing a program is a very-very difficult task.
↳ and in DBMS :- there is No any data isolation problems.

#04. Integrity problems :- data stored in file :→
you write a code or prog. ← for each constraint

Now if data stored in DBMS :→ the DBMS Based S/w Handles everything & we don't have to write any code / prog.
↳ so during the building the DBMS S/w :- you have to mention all the required constraints and Afterwards the s/w will take care of everything → for lifetime.

#05 Atomicity :- word meaning :- "all or None"
means either everything will be computed or Nothing will start.

→ Storing in files :- and updating the data in files & Now our requirement is to grab the previous data before updation :- we don't Have

facility for that in file system. :- Hence Atomicity is not present here.

But in DBMS :- atomicity is ensured.

↳ automatically it handles everything.

→ you have write code everytime to ensure atomicity in case of file systems.

#06 Concurrent access anomalies

↳ Just like the Reader-writer's problem.

→ Problem of Mutual exclusion!

↳ which we studied in OS.

↳ So in files :-

this problem of concurrent access or Mutual exclusion is present.

But in DBMS we have the facility of locking the system if one person is accessing the shared data then another one has to wait for his chance to access the same data.

Hence in DBMS s/w → concurrent access cannot happen for same data.

#07. Security :- Security in file systems is Not at all ensured. → data is authorised by anyone in file systems :- which is actually very lethal for data.

But in DBMS system :- One User can access only his data and not anyone else's → which do not belong to him.

Now authorization can easily be changed using the DBMS systems → and hence the security is ensured.

So these were some disadvantages of Not using the DBMS

Hence nowadays DBMS is widely used everywhere.

08/08/2022

Lesson 02

Data Model :- A collection of conceptual tools for describing data, data relationships, data semantics & consistency constraints.

↳ So here :- how to represent the data, how to store it :- and also relationship b/w several data and also the constraints on several attributes :- all these collectively done and that collective thing is called data model :- in which some conceptual tools are needed to do all these things.

↳ like for ex :- representing a student data in the form of a table :-

Roll No.	Name	DOB	phone No.

→ This is how we'll store the data in data Base (DB)

finally :- through data modelling :- how are you representing the data conceptually.

and we have following two types of data Modelling :

1. The Entity - Relationship model
2. Relational Model → our major part of course.

The Entity Relationship Model

⇒ def^n :- The Entity relationship (E-R) data model consists of a collection of ~~data~~ Basic objects, called entities, and of relationship among these objects.

⇒ entity :- or objects :- things which have the similar characteristics

→ Having same property.

like :- student, teacher, course, dept, etc. are different objects.

Now one student = one entity and the collection of all the students = k/as entity set

→ Hence :- collection of multiple instances of same kind of objects/entity :- is k/as Entity Set.

So in E-R data model :- How we will represent the two (or more) different entity sets and the relationship among them.

→ Entity Set 1 ⟷ Entity Set 2
⇓
relationship

like :- teacher is related to course, dept & students.

Relational Model :- The relational model uses a collection of tables to represent both data and the relationships among those data.

↳ So here in relational Model :- we'll represent the data in the form of tables

→ table having some attributes/properties ~~[illegible]~~ like :-

Student → Table 1.

R.No	Name	dOB	phone
1	—	—	—
2	—	—	—
3	—	—	—

all these individual entries → 1, 2, 3 represents one student :- which is our one object.

another table we can have :- course → Table 2.

CID	C-Name	teacher
1	—	—
2	—	—
3	—	—

and these both two tables may have relationship b/w them.

Now if we have used Relational ~~Mode~~ Model to implement the DBMS :- then we call it as :-

Relational DBMS :- RDBMS → our whole syllabus!

Now here in Relational Model :-

A Table also means a Relation

Ex. Table of student also means Relation of student.

Instance & Schema :-

↳ Schema :- Overall table design. including table Name, columns, type of values each column can have, the No. of columns, etc.

like :- Roll No. :- only INT.

→ Name :- Only characters

→ phone No :- Only Numeric value with 10 digits only

↳ Now this is the table design which is klas the schema of the table

These are the rows of the table!

→ Instance :-

↳ This means that what kind of content / data is present at a particular time / Now.

↳ current data (values of the attributes) / rows in database tables.

↳ This may change at the later point of time

DataBase Languages :- Languages used for accessing the DB, its creation / deletion etc.

following are the two languages :-

#01. data definition Lang. (DDL) :-

This is related to the schema of the table.

like :- creating a table / deleting a table / adding extra attributes / columns to the table, etc.

→ anything you want to make changes in schema of the table :- we'll use the DDL

#02. Data Manipulation Language :- related to the instances or data values/rows

↳ insert a row → delete values → deletion of rows

→ fetching or reading rows, etc.

Here in DML :- we have further its two types :-

(i) Procedural DMLs :- This require a user to specify what data are Needed and how to get those data.

(ii) Non Procedural DMLs :- Require a user to specify what data are Needed w/o specifying how to get those data.

→ (declarative) ⇒ SQL is Nonprocedural.

DataBase Users and Admins :-

1. Naive users :- those which don't know anything about the DataBase and its concepts. like every other person who is using the Internet.

2. Application programmers :- S/W developers who uses DB tools for development.

3. Sophisticated users :- Just extracting the data.
4. Specialized users :- those who are developing DB tools.
5. DataBase Administrator.
↳ who is managing the DB.

View of data :-

any DB creator :- who creates the tables :- through the Database tool, Cannot access the tables directly in our disk.

He can only view the tables through the DB tools only.

→ the tables are stored on disk :- which is hidden from everyone including the creator of the tables.

and the DB file :- you can access through the DB tools.

Now you have the two levels of DB :-

1. physical level :- level where the DB is stored on disk :- all tables & DB file are stored on disk :- & No one knows that where it is stored on the disk.

2. logical level :- The level where you can see the tables you have created through the DB tools :- you can view your table in perfect format.

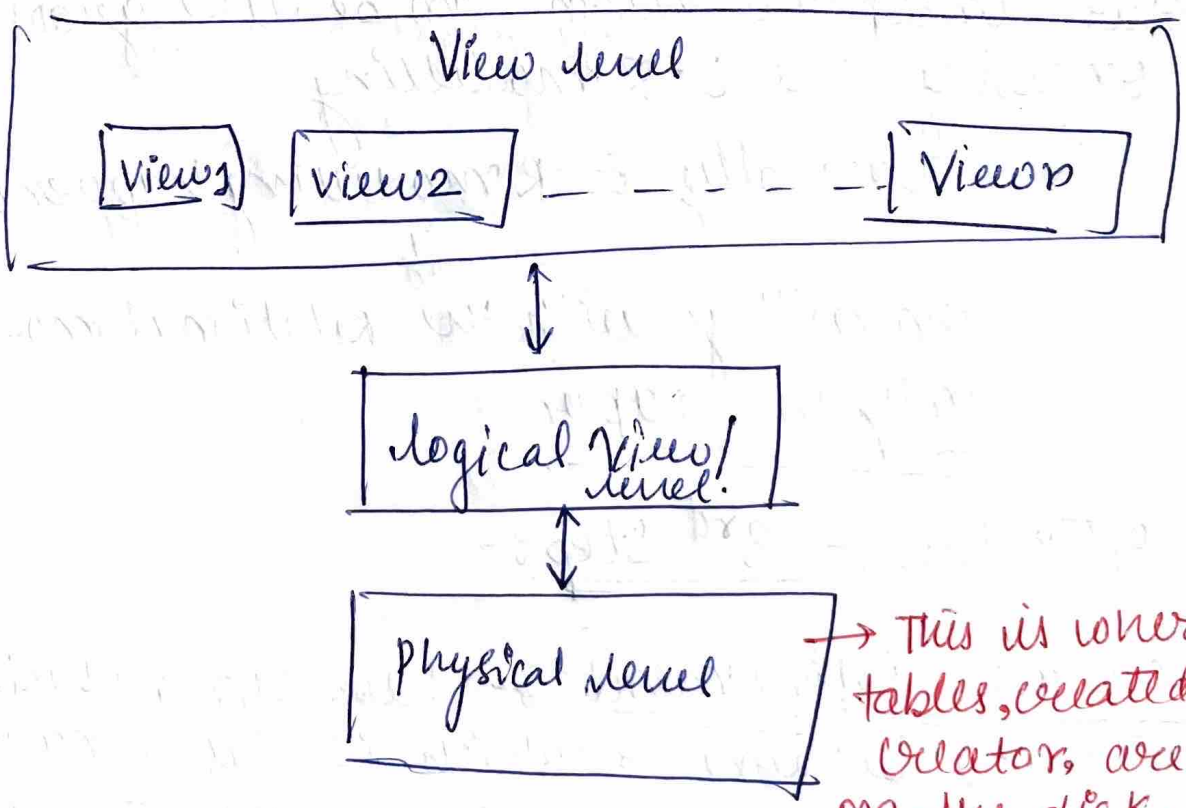
→ So where the tables are stored on the disk and how the tables are stored is Not known at all!

→ So the logical view of the tables :- will not show us how & where these tables/files are stored in disk.

→ View level :- Some value of database are collected from the entire DB :- and shown to the user → according to his Needs.

In a very beautifully arranged manner.

and hence the logical view of the complete database is hidden from the user who just wants the specific data acc. to his requirement.



→ This is where the tables, created By DB Creator, are stored on the disk.

09/08/2022

Lesson 03

Database Design :- Here if someone has to build a DB :- then what all things he need to do step by step to build a complete DB !

1. Requirement analysis :- Here the demand of the client is analysed, that how much data is need for the project of client, etc. etc.

2. Conceptual DataBase design :- it says that :-

   On paper (the Blueprint of dataBase)

   → This is for the rough idea of our DataBase → before building the DB in our server !

another conceptual design can be like graphical or using the E-R modelling

→ Basically E-R modelling happens

⇓

then we'll go with the Relational modelling only in paper

and after this :- 3rd Step :-

3. Schema Refinement :- Here Error checking will be done → which is Klas Normalisation. Here the anamalies present will be removed to make our DB design to its perfection.

→ So in Schema refinement :- refine & redesign the Schema to eliminate the anomalies.
This is also done only in paper.

4. Logical DB design :- Now after whatever things we needed → we have got in paper. Now the DB designer will use the DB tool to implement the Schema.

DB tools like :- Oracle DB, MySQL
Here the coding will be done using the DB languages

DDL and DML

5. Physical DB design :- Now this is the most impor. step of our DB design.
→ It takes decisions about the physical storage & indexing.
→ DB designer will give info to the DB tool for how to store the data in the disk → and all the reqd decisions :- DB designer tells to the DB tool.

6. Security design :- Providing authorization (means who can access what type of data) and authentication. (providing some kind of log in details to the authorised person)

* **Indexing** :- used for searching purpose, searching the data in our computer DB tables.

Entity-Relationship modelling :- coming under the conceptual DB design.

→ Entity :- any real world object having some characteristic that makes him different from other entities and those entities may have relations among them.

↳ lots of Entities in our DB having some relations.

So in E-R modelling :- through the diagrams we are going to present the entities and the relationship among them.

→ **In ER modelling** :- Objects/Entities having same or similar charac. are grouped together in one set and that set is klas **Entity set**

→ we represent entity set in ER modelling.

→ **attribute** :- An Entity is described using a set of attributes.

↳ like for entity set :- Student :-

attribute :- RollNO., PhoneNO., DOB
Name, Father_name, Address
etc. etc.

→ The table column Names are our attributes.

→ **domain** :- these are like the constrains that one attribute can have.

like in **Name** :- only charac. are allowed and not the integer values.

like phone No. :- only 10 digit. Integers are allowed. , etc.

→ A Unique Set of Values permitted for an attribute.

↳ Here we are talking about the data only → that will be stored in the tables

→ **Relationship** :- An association among two or more entities.

like for Ex:-

TP (Entity, Student) ——— VD sir Entity, Educator

Now this is one particular relationship. and there are many other relationships also.

and all the relationships combinely are the relationship Set !

→ **Relationship Set** :- A Set of similar relationships

↳ so here ~~the~~, just like the Entity set, similar relationships are group together and are represented in one relationship set.

→ **Key** :- An attribute or set of attributes whose values can uniquely identify an entity in a set.

↳ a particular attribute (like RollNo. incase of Student entity set) having a unique value and through which we can extract the entire row corresponding to the value of that attribute (Also Key).

↳ Sometimes more than one key together can also be the key.

E-R diagram :-

1. **Entity Set** :- represented as :-

   Entity_Set_Name

   ↳ inside a rectangular Box

2. **Relationship set** :- represented as :-

   inside a diamond Box

   Relat^n_Set_Name

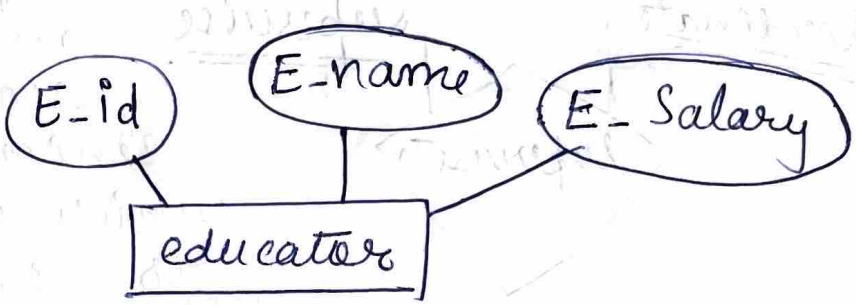
3. **attributes** :- attribute_name

   ↳ inside a oval Box

4. **association** :- represented through a line :-

→ lines to connect the Entity set, relation set & attributes.

Ex :-



→ types of relationships :- How many No. of Entity set are involved in one relationship.

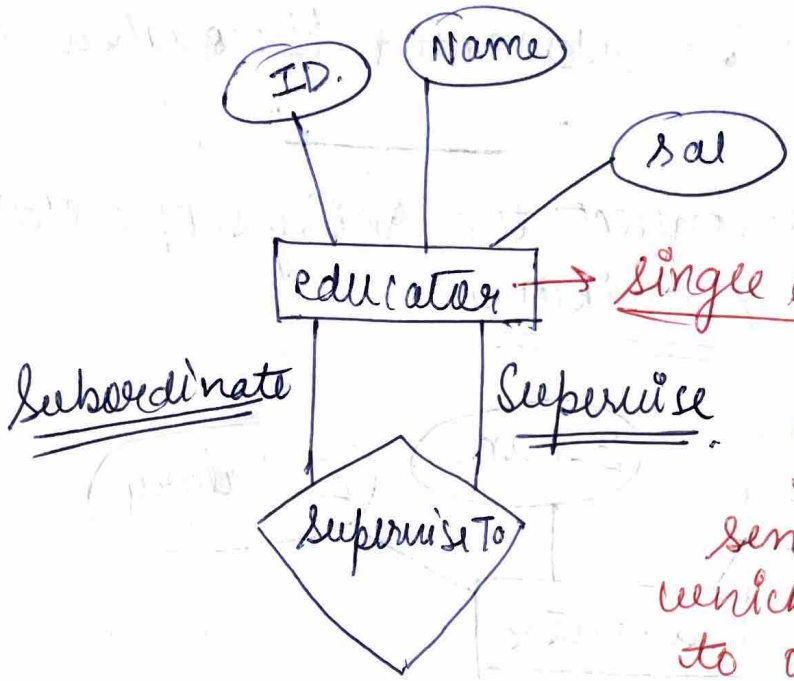
1. Unary → when only 1 entity set is involved
2. Binary → when 2 entity sets are involved
3. Ternary → when 3 entity sets are involved

→ Unary Relationship :-

Ex :- one senior educator is supervising its junior educator or subordinate :-

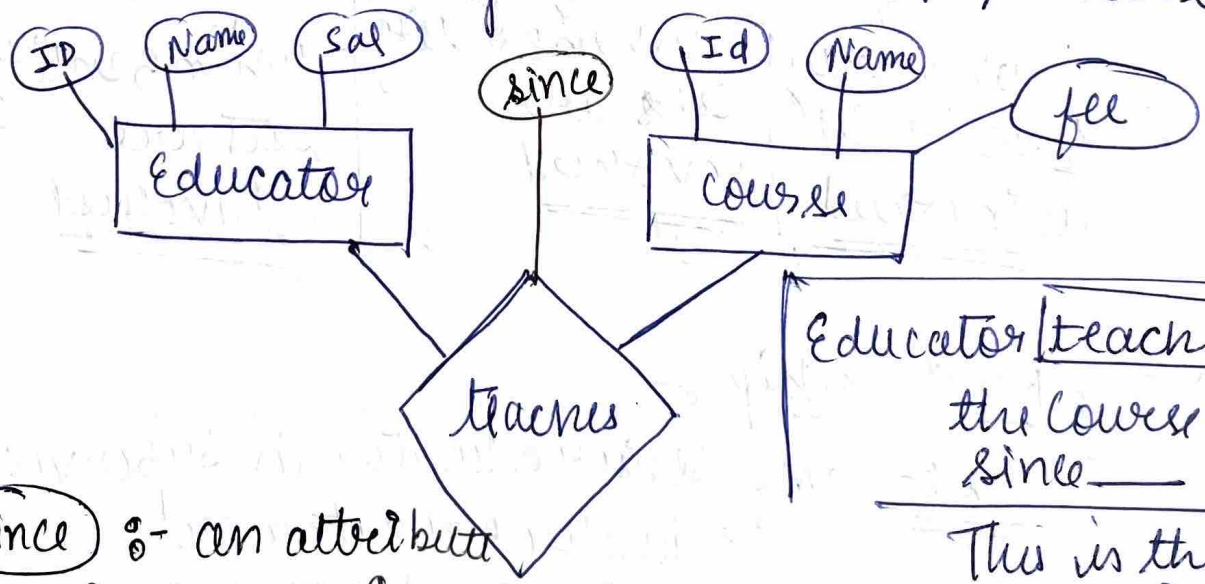
only one entity set is involved : educator

diagram →



→ single entity set
↓
and some rows are of senior educator which are related to other rows of Junior Educator through the Supervise to relationship.

→ **Binary Relationship** :- consider the two entity sets → educator & course.

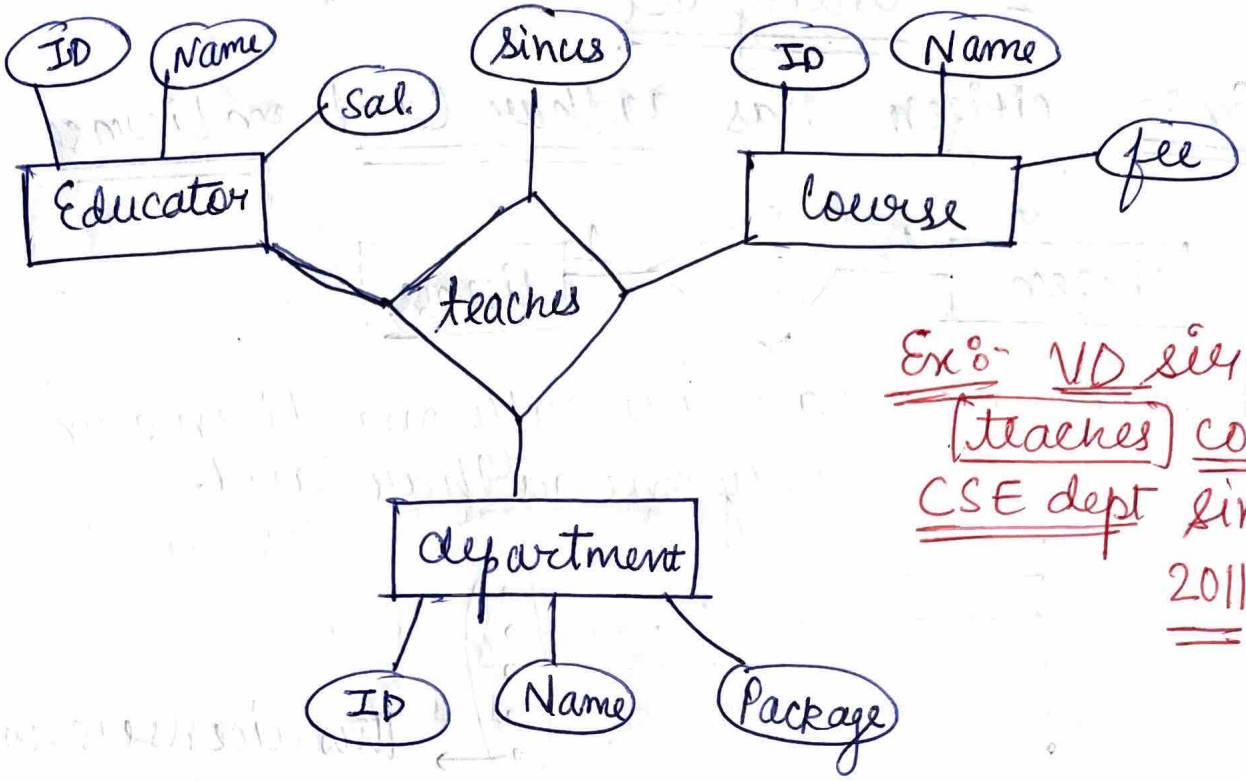


Educator teaches the course since ___
This is the relation b/w the two different entity set.

(since) :- an attribute of the relationship set.

descriptive attribute :- an attribute of the relationship set.

→ Ternary Relationship :- 3 different Entity sets involved in single relationship :-



Ex:- VD sir [teaches] COA in CSE dept since 2011 !

Now this is what an E-R model is !
↓
Entity - Relationship model.

Mapping Cardinality :- How many Entities from ~~the~~ an Entity Set is related to how many entities in other entity set.

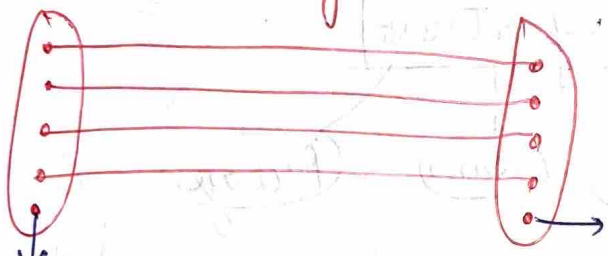
→ So if we are having two entities sets involved in a relationship → how many from 1st entity sets is related to how many of 2nd entity sets !

→ So Based on above factor we have, we have 4 types of Mapping cardinalities :-

1. **One to one** :- One entity from 1st entity set is related to only one entity in another 2nd entity set.

Ex :- citizen has aadhar card or License

↳ one citizen can have only one license or only one aadhar card.



This citizen don't have the license as of now.

This license is not allocated to anyone

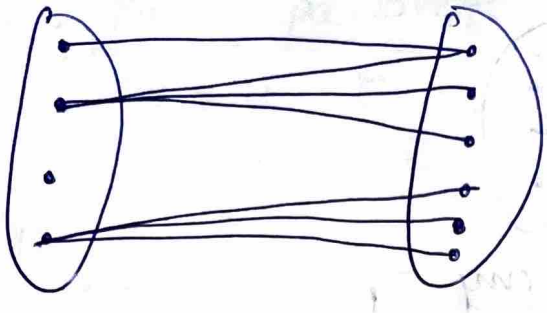
↳ Yes ✓ it will work

2. **One to many** :- One single entity from 1st Entity set is related to multiple Entities of 2nd entity set.

Ex :- One particular educator is related to many students.

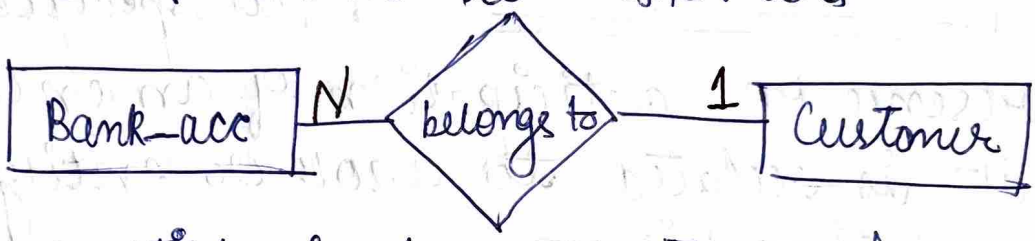
or

1 Employee manages multiple teams :-

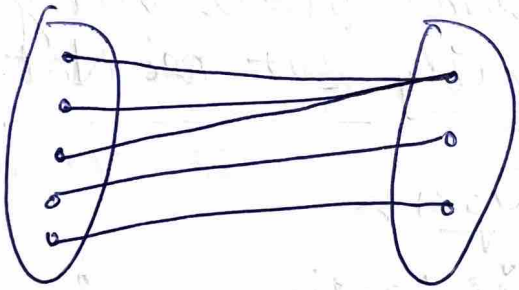


3. Many to one :- Many (multiple) entities of 1st Entity set is related to only one entity from 2nd entity set.

Ex :- Bank account to Customer



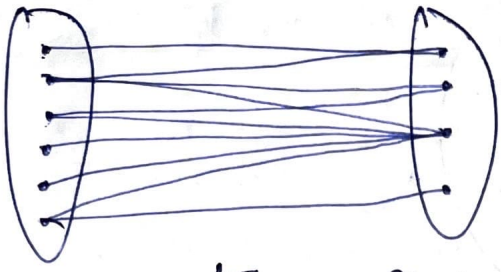
↳ multiple Bank accounts can be owned by 1 single customer.



4. Many to Many

Ex :- Educator to Course.

↳ Multiple Educator teaches multiple courses.



many to many !

→ So these mapping cardinalities will help us to Build the tables of Relational Modelling.

Participation Constraints :- This specifies the Presence or Participation of an entity when it is related to another entity in a relationship type.

↳ whether one particular entity is related to some other entity in some other entity set or Not ?

→ We have its two types :-

1. total participation :- Complete Participation of all entities from a particular entity set.

↳ Ex :- every Bank account belongs to someone customer.

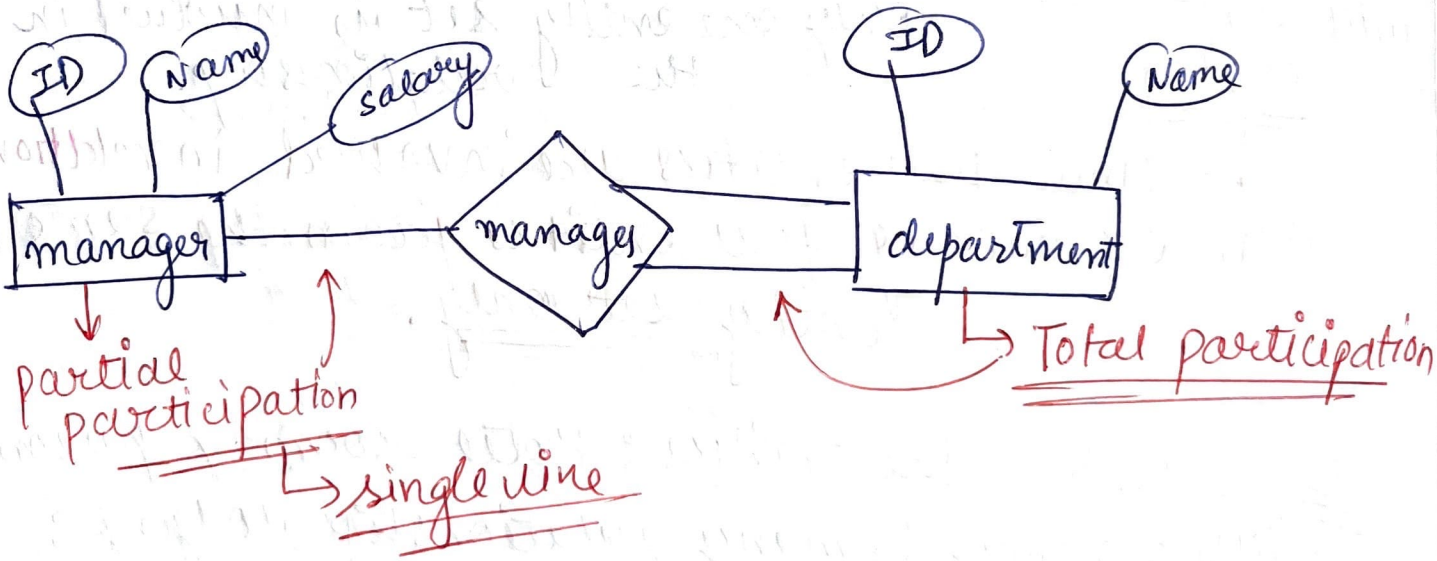
Hence Bank accounts is in total participation.

2. **Partial Participation** :- Some entities of an entity set do not participate / or are absent in the relationship.

like for Ex :- ~~Every~~ Customers ~~cannot~~ may / may not have a Bank account.

↳ some Customer may not have any Bank account.

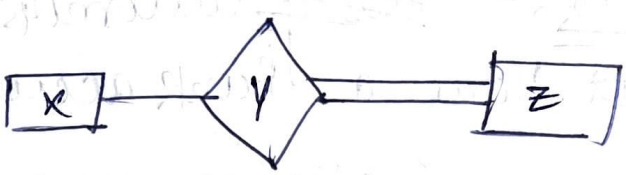
→ The total participation is denoted by the double association lines :-



10/08/2022

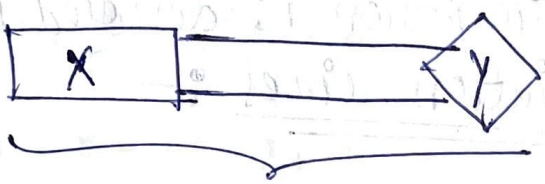
Lesson 5

Note :- In total Participation :- the double lines are closer to each other :-



and in Case of Unary relationship :- the lines are wide apart from Each other !

→ total participation in Unary relat^n will not exist.



only one entity set is involved in the relationship.

↳ since few entities are involved in relation with remaining few entities from the same entity set only.

Ques. Consider the entities : Hotel room & person with a many to many relationship lodging :

if we wish to store information about the rent Payment to be made by persons occupying different Hotel rooms, then this information should appear as an attribute of :- ?

Soln. So the rent Payment :- will go to the Lodging.

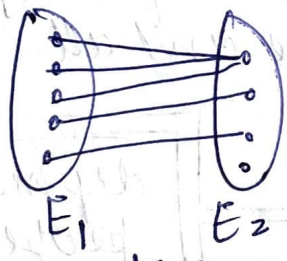
→ money payment is an attribute of lodging.

→ so rent depends on a person living in a Hotel room.

Que. In an ER model :- Suppose K is a many to one relation from set E_1 to entity set E_2. assume that E_1 & E_2 participate totally in R and that the cardinality of E_1 is greater than the cardinality of E_2.

Soln.

$|E_1| > |E_2|$



many to one.

Hence :-

Every entity in E_1 is associated with exactly one entity in E_2.

Now if there will be partial participation :- instead of total participation :-

then another possible answer :

Every entity in E_2 is associated with atmost one entity in E_1

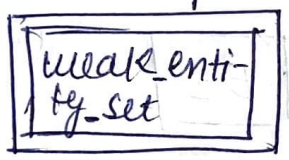
Weak or strong Entity sets :-

Any entity set having "Key" → is the strong entity set.

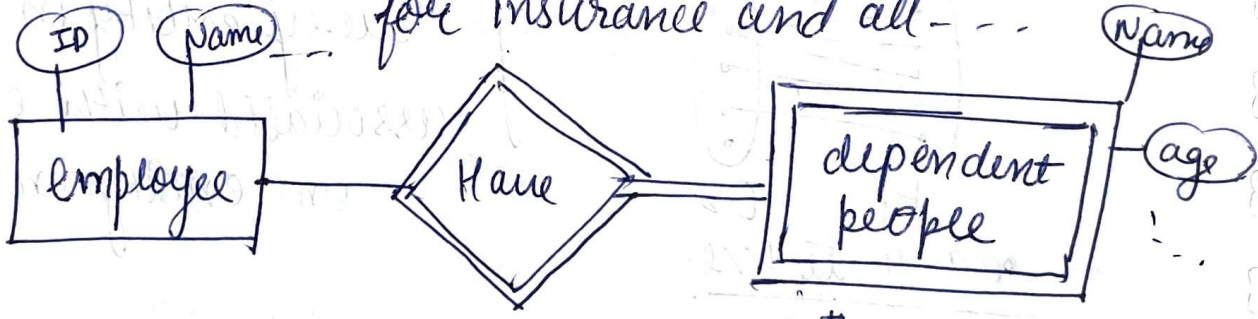
and those entity set which don't have any attribute as "Key" → is the weak entity set.

↳ Now why do we keep this :- because we want to have some information which are dependent on some other information.

→ weak entity set is represented in double boxes :-



Ex:- an employee's details about his family members for insurance and all - - -



⇓

Now this don't need any attribute like ID to uniquely identify the family members of an employee.

↳ company don't need at all.

Hence this is a weak Entity set!

→ the weak Entity set will have total participation :- for sure!

* two weak Entity set will not be connected to each other → as then how to identify uniquely the row?

↳ Hence each weak Entity set is always in a total participation relation with one strong entity set only

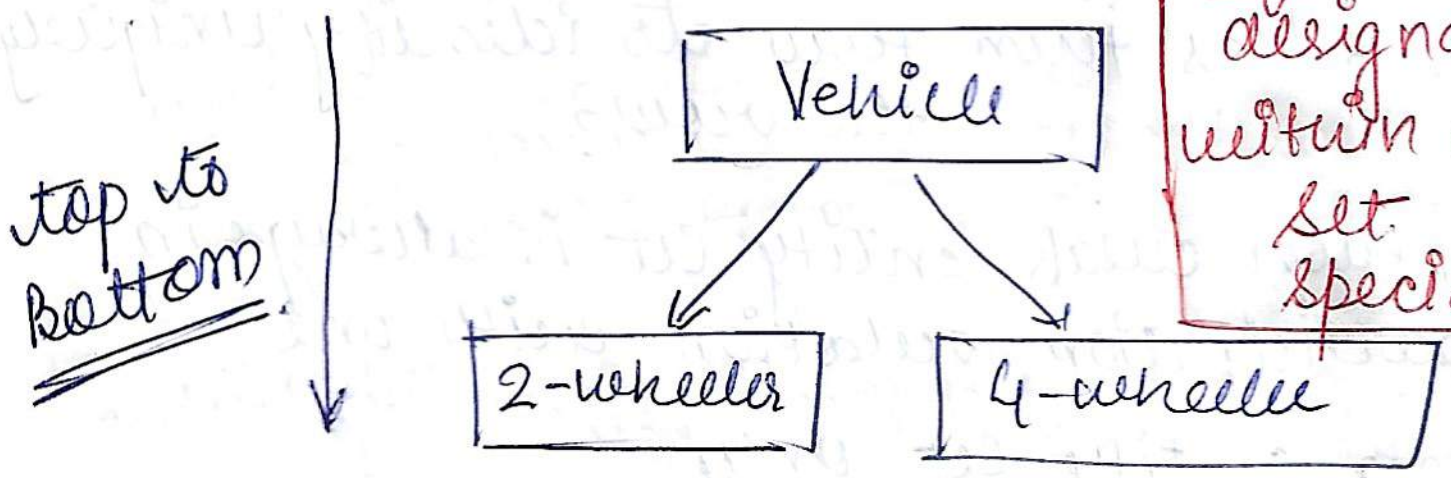
Now some points :-

1. Weak Entity Set does not have a Primary key.
2. Weak Entity set depends on strong Entity set.
3. —— " —— is represented by double rectangle
4. The relⁿ b/w one strong entity set and one weak entity set is represented by a double diamond.
5. weak Entity set always has total participation.

Extended E-R features :-

1. Specialization :- Whenever you feel that :- One particular entity set can be further more divided into multiple entity set Based on some specialization, then using this feature of ER modeling we can divide our original entity set. done for the better clarity of the data.

Ex:- Vehicle entity set can be divided into following two entity sets Based on the No. of wheels :-

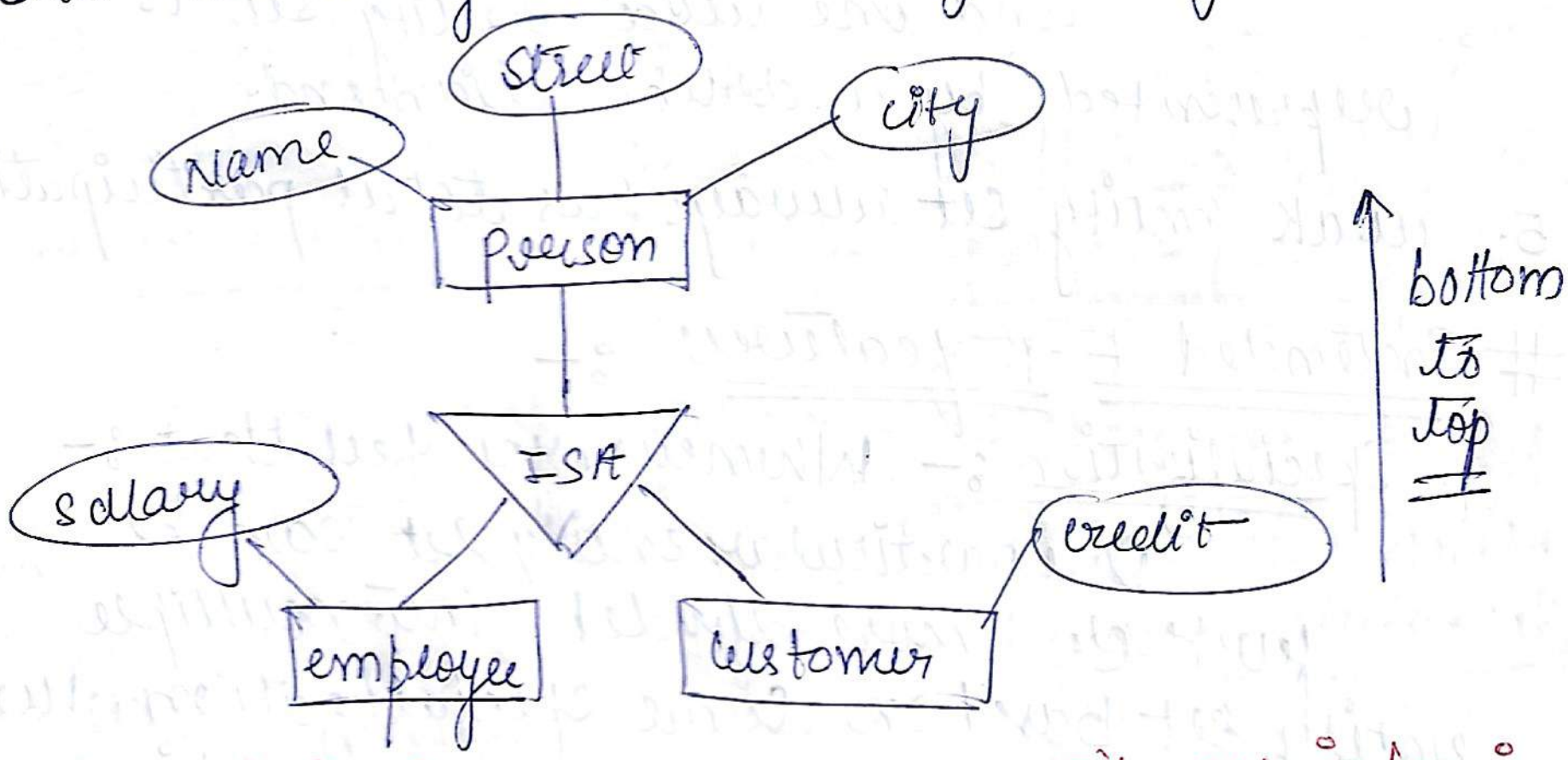


defn: The process of designating subgroupings within an entity set is called specialization

2. Generalization :-

Here multiple entity sets having similar characteristics can be merged into one single entity set in a generalized manner.

→ lots of things are common b/w multiple entity sets are merged into one single entity set.



Now in specialization :- the parent entity set is having few attributes → and all those attributes are inherited to its children entity set.

→ Every child entity set inherit all the attributes of its parent entity set. and apart from these inherited attributes :- Each child entity set is having its own separate few attributes which distinguishes him from other children entity set.

↳ and this is klas attribute inheritance.

→ we have two types of specializations :

1. disjoint :- Here the Multiple child entity sets don't have the same or common rows/entities b/w them. All have unique and different entities. like :- savings and current accounts of a Bank cannot have common entities.

2. Overlapping :-

↳ Here the child entity sets have common or overlapping/same entities/rows.

↳ like :- an employee of a company like amazon can also be a customer of same company like amazon.

Now the generalization & specialization both are categorized into further two types : →

total generalization or specialization :-

Each higher level Entity must belong to a lower level entity.

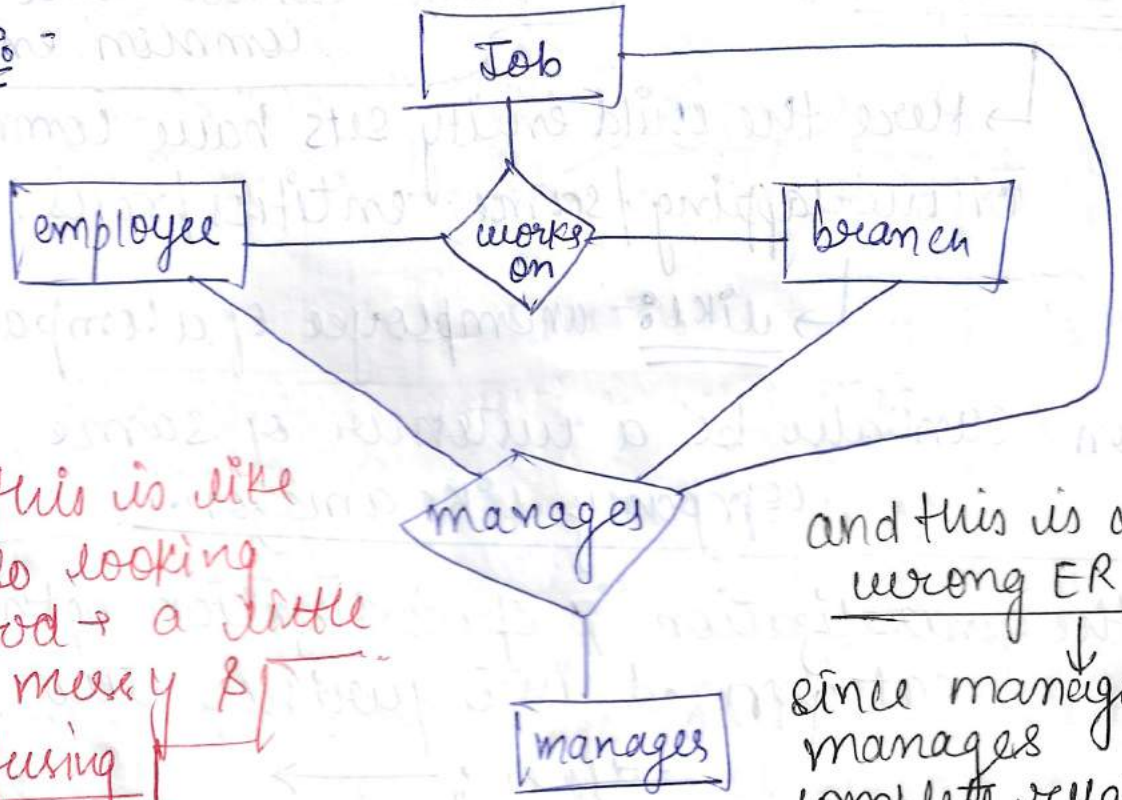
↳ means Union of all lower level entity set should result into the higher level Entity set.

Partial generalization or specialization :-

Some higher level entities may not belong to any lower-level entity set.

4. aggregation :- Relationship set participating in relationship, then aggregation is used.

Ex:-



Now this is like Not so looking good → a little messy & confusing

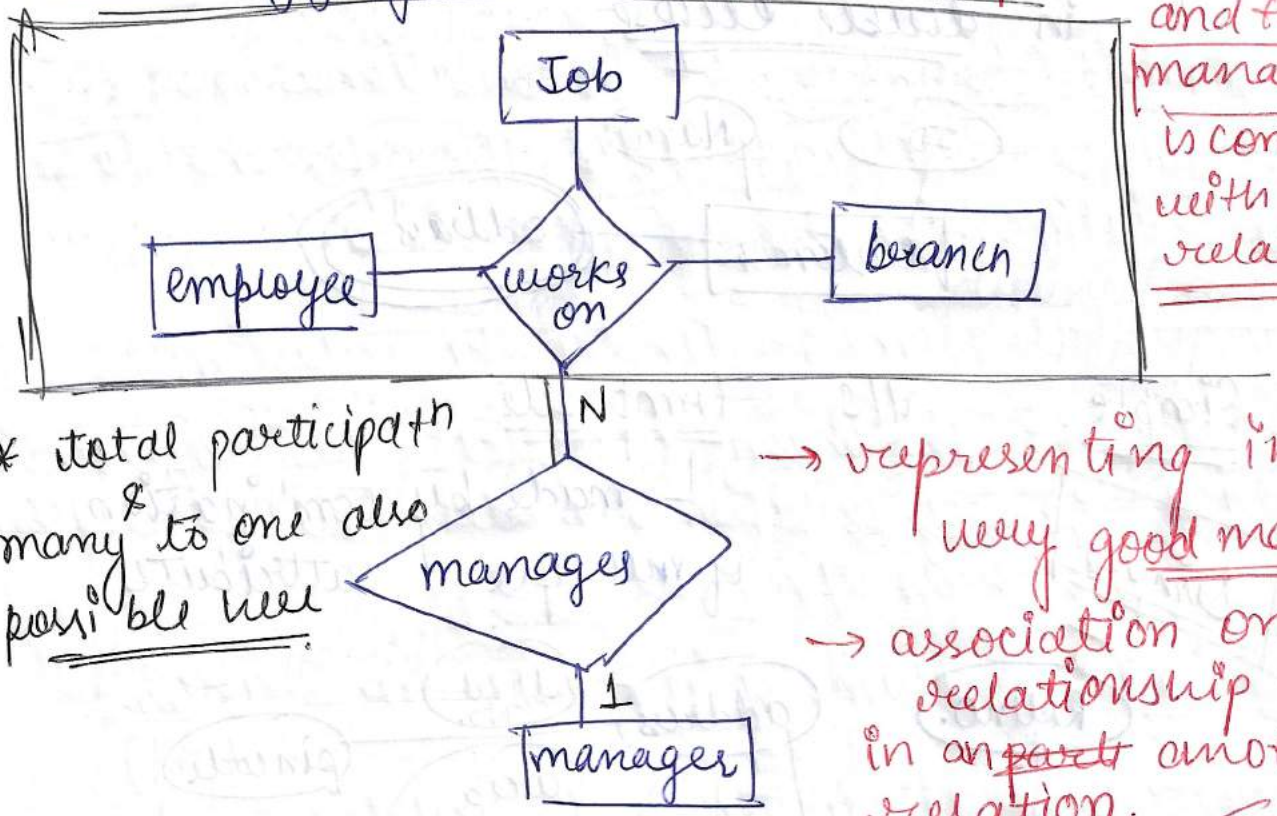
and this is actually wrong ER model

↓

since manager manages the complete relation & Not the individual entity set.

Now the aggregation :-

→ This rectangle is showing one complete relationship and the manages rel^n is connected with this relation.



* total participat^n & many to one also possible here

→ representing in a very good manner!

→ association one relationship set in another relation.

rather than connection the Manages relationship set with entity sets.

↳ which didn't look good and was confusing!

types of attributes :-

1. Single valued V/s Multi valued

Single valued → for each entity only 1 single value

↳ Roll No., Name of student

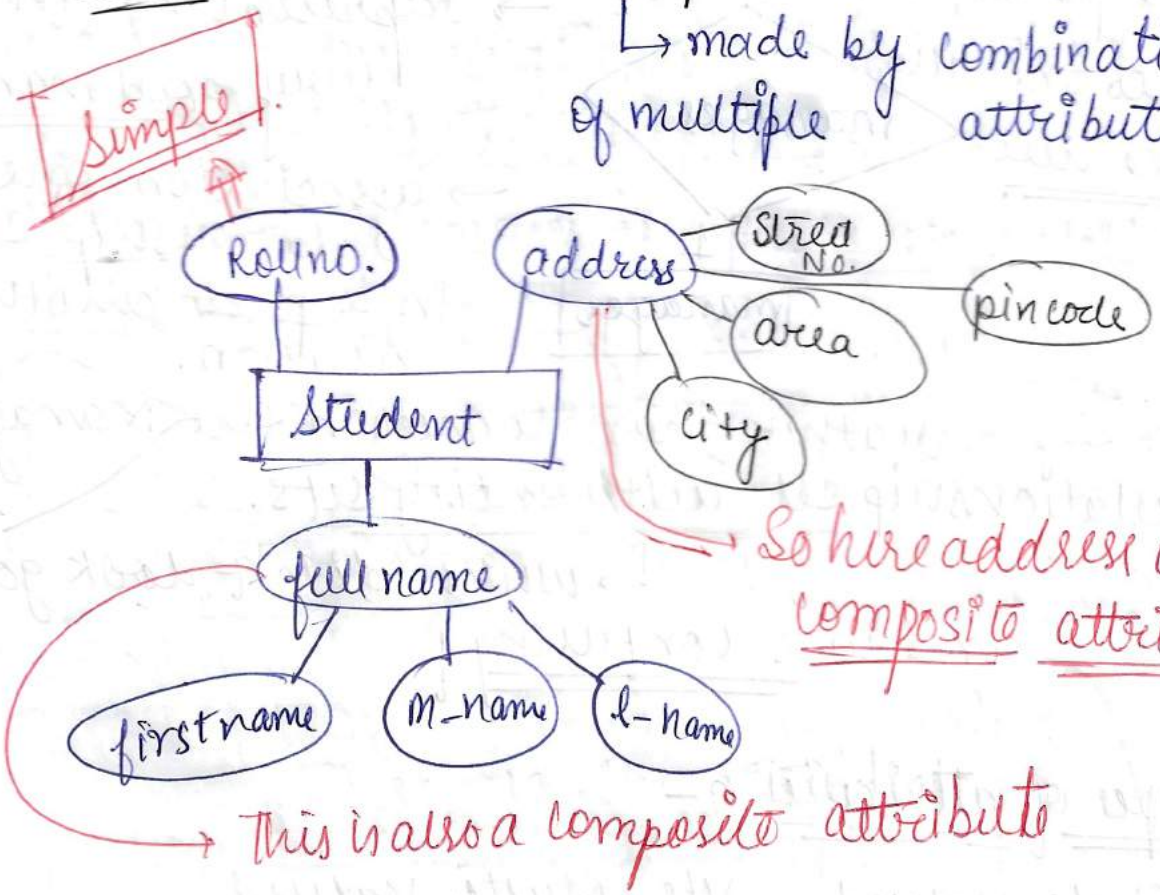
Multi valued → for each entity → multiple values of a single attribute.

↳ phone No. of student

→ multiple valued attributes are represented in double ellipse

2. Simple v/s Composite

↳ made by combination of multiple attributes



→ So Simple attribute :- we cannot divide further into multiple attributes

whereas :-

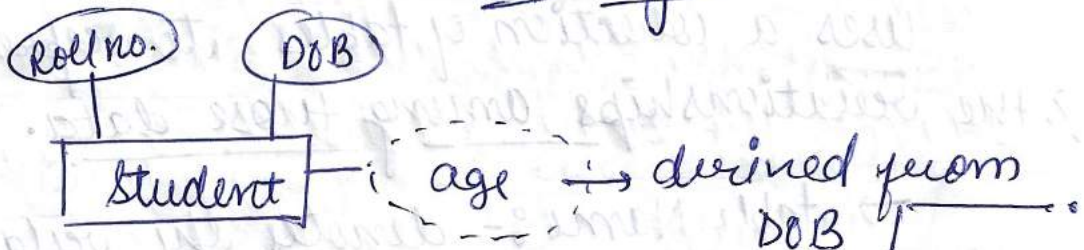
composite → we can divide it.

↳ full name
↳ address.

3. Given (Stored) V/S derived

Ex: DOB

Ex: age



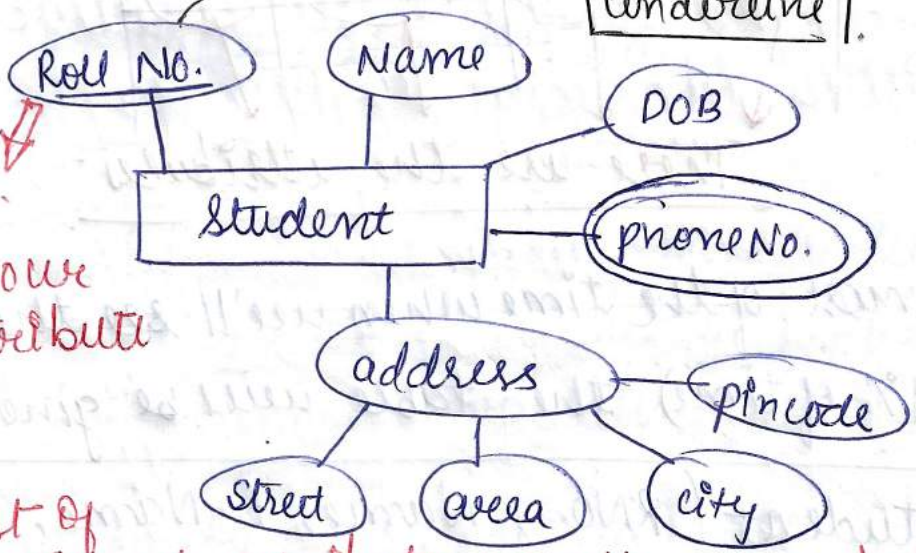
→ So derived :- derived from stored attributes or given — " — !

4. Prime V/S Non prime :-
(Key) (Non-Key)

↓ attribute which is part of the key.

That → attribute which is Not the part of key.

→ prime attribute/s have an [underline].



Key.

Hence this is our prime attribute

→ and rest of the mentioned attributes are the Non prime because they are not the part of the [Key].

→ Key :- can have the combination of multiple attributes

11/08/2022

Lesson 06

 # Relational Model :- The relational model uses a collection of tables to represent data & the relationships among those data.

→ table Name :- denotes the relation.

→ **Relation** :- The main construct for representing data in the relational model is a relation which is the table it self.

→ **attribute** :- Colums of the table or relations.
(or field) ↳ used to describe the relations.

Student → Table Name or Relation.

Ex :-

RNo.	Name	F-Name	DOB

These are the attributes.

Now most of the time when we'll see the questions, (99% of time) the table will be given like :-

Student (RNO. , Name, F-Name, DOB)

↳ almost all the question will have table mentioned in this format.

→ **Record or tuple** :- A row or an entery/entity in the table/relation.

→ **DataBase Schema** :- Logical design of dataBase
↳ all the designs of each and every table of our DataBase

→ **DataBase Instance** :- Snapshot of the data in the DataBase at a given instance in time.

→ **domain** :- Unique set of values permitted for an attribute.

→ **domain Constraint** :- Specifies an important condⁿ that we want each instance of the relation to satisfy
↳ each row or each value of an attribute must satisfy the constraint applicable on the attribute.

like :- the acc. No :- cannot have values $\boxed{<0}$
↳ or :- cannot be kept empty and cannot be repeating value

→ **degree or arity** :- No. of attributes in a relation

→ **cardinality** :- No. of tuples/rows in a relation

→ Now, Can we skip any Value while inserting a new row in a table?

↳ Now, this is also a kind of domain constraint for some attributes :- that if at present we are not having any value of an attribute then we can skip it.

↳ So we will fill [NULL] inplace of any value. [NULL] denotes that no value is available.

→ Some attributes → cannot have :- [NULL]

↳ these having the constraint :- NOT NULL

→ Keys :- An attribute or set of attributes whose values can uniquely identify a tuple in a relation.

Ex :-

Student

RNO.	Name	FName	DOB
1	TP	AJP	20-01-99
2	BP	AJP	13-08-01
3	TP	SP	22-01-22
4	SP	HP	18-02-22

→ Now here :- Name of two students are same But Not their [RNO.] → which is actually unique for all entities.

Key :- can be [RNO]

↳ can also be :- [Name + FName]

↓

Combination of two attributes

→ This combination also uniqely identifies each row in our student table/rel^n.

Now, various kinds of keys we have :-

#01. Super Key :- all possible keys in a relation

↳ Basically its the Super Set of all keys.

like in our previous student table :- following are the possible keys :-

Rollno. , Rollno. + Name , RNO. + FName , RNO+DOB, RNO+FName+ Name, RNO. + Name+DOB , RNO. FName +DOB , RNO+Name + FName +DOB.

→ So here we can see :- when we identify an attribute as a key :- then all the remaining attributes can also be combined to form a key → which will be the part of the [Super Keys]

#02 Candidate Key :- Minimal Super Key.

The Super key whose proper or Subset is not a Key.

→ identify such kind of a key :- which contains ~~the~~ or involving min^m No. of attributes.

So in our previous ex :- RNO. → is our candidate key

also :- Name + FName

↳ alone Name & alone FName cannot be a key.

Ex :- R(A, B, C, D, E)

Keys :- AB, AC, ACD } So here the Candidate key :- AB, AC.

↓ Super Key.

A :- alone not a key.

↓

and if we invoke other attributes with AB or AC :- then that key will be the super key.

→ Now from the given keys :- one possible key = ACD

check if alone A / C / D / AC / AD / CD is a key or Not.

and if any ~~Key~~ attribute is the key :-
among A/C/D/AC/AD/CD :- then
ACD will be the Super Key.

→ So here AC :- alone is the key. → Hence ACD is the super key

#03. Primary Key :- In a relation we can have the multiple Candidate Keys
So out of all candidate keys
↳ we will select one key as our Primary key.

→ So :- its chosen candidate key for implementation of our DataBase.

→ So it must be :- Unique & NOT NULL.

Que.

R

A	B	C	D
A1	B1	C1	D1
A2	B1	C1	D2
A3	B2	C2	D1
A4	B3	C2	D2
A4	B4	C3	D3

P. Key # AB

So if we are inserting a new row with following values :-

A5 Null C4 D4

↳ So is this row Valid?
↳ allowed or Not?

Sol" → ~~Ok~~

So we will keep this question unanswered here only as of now. and will answer during the implementation
↳ SQL

#04. alternate key :- those (candidate keys) which are not chosen as Primary keys are called alternate key.

#05. foreign Key :- Based on refertial integrity constraint.

Eg :-

primary key

BID	Name
B_1	mgrood
B2	Teli
B3	Market
B4	magneto

⇒ Bank Branch table

referring to Primary key of another table

Account

ACNo.	Name	Amo.	BID
1234	sohil	5000	B5

So this is made the foreign key which is referring to BID of Branch table

we cannot open an account on the Branch having ID = B5 ∵ becoz there is no any B5 Branch ID in our Bank Branch table.

So here to solve this issue :- we will have the constraint on the BID attribute of account table :- that it can have only those values → which are mentioned in the Bank Branch table.

↳ So BID of account table can have :- B_1 | B_2 | B_3 or B_4 values only.

→ So Referential Integrity :- we wish to Ensure that a value that appears in one relation for a given set of attributes also appears for a certain set of attributes in another relation.

This condⁿ is called referential integrity.

Ex:- Publisher (PubID, PubName, PubAddress)

Book (ISBN, Title, Revision, PublisherID)

Now here :- PublisherID in Book relation ⇒ foreign key.

↓

referring to PubID of Publisher relatⁿ

So defⁿ of foreign key :- The foreign key constraint prevents invalid data from being inserted into the foreign key column, because it has to be one of the values contained in the parent table.

Ex:-

Customer

Cid

Employee

EID	E Name

Sale order

Soid	Cid	Eid

→ must match with the EID of the Employee table.

→ foreign key :- Can be NULL or repeated value.

→ following two constraints are automatically applied here :-

1. On delete Cascade
2. On-update Cascade.

Consider :-

Branch

BID	Name
B_1	
B_2	
~~B_3~~	
~~B_4~~ → B_6	

account

ACID	Name	BID
		B_1
		B_1
		B_2
		~~B_3~~
		~~B_4~~ → B_6
		~~B_4~~ → B_6
		~~B_4~~ → B_6
		~~B_3~~

Now in Branch table :- we are updating the B_4 value to B_6 in BID attribute.

Now the BID is also the foreign key in "account" table → Hence automatically the B_4 value (wherever it is present) will be updated to B_6 → This is On-update Cascade.

And Now :- from Branch table & from BID column :-
I am deleting all records corresponding to "B3" → Now since BID is the foreign key in "account" table :- so in "account" table also the records of B_3 will be deleted.
the "B_3" value of BID in account table will be deleted.
So wherever B3 will be present in account table :- all those rows will be removed from account table. → and this is what on-delete cascade is!

⇒ and these two constraints are automatically applied in foreign keys!

SQL : Structured Query Language.

↳ Domain specific language used in programming and designed for managing data held in a RDBMS!

→ operations performed using SQL :-
operations like :- creating a table, inserting data, retrieving data, updating the data, ... and many more ...

12/08/2022

SQL datatypes :- will write this afterwards!

→ case sensitive :-

→ Keywords of SQL ⇒ any case → either in upper case letters or in lower case letters.

→ Tablename or columnName = any case

→ Value :- same to same as it is stored!

→ Semicolon :- Not all DB tools
↳ Not mandatory.
↳ But it is used to separate multiple DB queries.

→ Select command :- Used to retrieve data from one or more tables.
↳ Syntax :- Select ---- from tablename

↳ when selecting all columns & all rows :-

Select * from tablename

We use '*' for all the records of all columns!

→ Selecting few selected column :-

Select ColumnName from Tablename
↳ all rows But only one column.

Now selected multiple columns :-

Select C1, C2, C3 from tablename.

C1 = column ~~No.~~ 1
C2 = column 2
C3 = column 3.

} any No. of columns we can select and we have to mention those column Names separated by commas in whatever sequence you want.

→ select command with distinct keyword :-
Need to be used with select, to fetch only unique values of designated column (s).

↳ Syntax :- Select distinct Column Name(s) from tableName.

multiple column(s) are seperated using comma. here also

Now consider one table :- itemmaster :-

item	itemp
1	2
2	4
3	Null
4	Null

running the query :-
select distinct itemp from itemmaster :-
will result into :-

itemp
2
4
Null

→ So null also selected as one distinct Value.

14/08/2022

Lesson 08

→ Using dot to access the column

Syntax

So ~~ex~~:- Select tablename. columnname from tablename.

→ So this is used when we are accessing the same column having same Name from two or more tables → So dot operator is useful here!

↳ using this, we can also fetch multiple columns from multiple tables!

→ Select command with 'WHERE' clause :-

Now this where clause is used to filter out the rows → Based on some cond^n mentioned!

↳ selecting very specific rows only

Syntax :- Select columnName/(s) from tableName WHERE cond^n.

→ where clause :- used with select, update, delete, insert commands

↳ used to filter specific rows from table.

Ex:- return all such customers' info who live in country germany.

↳ Select * from customers where country = 'Germany'.

This value must be exactly same as that of value entered in the customer table

Ex:- Return all such customers' info who have their customer ID = 3.

↳ So the query:-

Select * from customer where Customer ID = 3

Integer value don't need to be mentioned in the quotes.

Ex:- Now selecting some specific columns:-

return city and country of customers who have their customer ID = 3.

↳ query:-

Select city, country from customers where customer ID = 3.

→ Relational operators used in where clause:-

1. Equal (=)
2. Not equal (<>)(!=)
3. Less than (>)
4. Less than or equal to (>=)
5. greater than (<)
6. greater than or equal to (<=)

Ex:- referring to the Order details table.

→ return all such order details when quantity is atleast 10.

So the query :-

Select * from Order details where quantity >= 10

→ return all such order details when quantity is max^m 10.

Select * from Order details where quantity <= 10

→ when quantity ≠ 10.

Select * from Orderdetails where quantity <> 10

or

quantity != 10

Both operators will work

Sequence :- the sequence in which our SQL query runs :-

↳ firstly :- "from" is executed very first

⇓

"where" → next executed

⇓

then "Select" → —"— at last

Now, the only thing which runs after Select is :- "Order By"

and No any command runs after "Select".

→ Now specifying multiple conditions in where clause :- So here we'll need to use the logical operator :- the logical operators :-

1. AND
2. OR
3. NOT

} we'll use the keyword as it is.

So ex :- from order details table :-

→ Select all such orders where quantity is atleast 5 & atmost 30.

So query :- select * from order details where quantity >= 5 AND quantity <=30

→ Between operator :- Used to filter the records in the specific range.

↳ Between LB and UB → Both LB & UB are inclusive.

LB ↓ lower Bound

UB ↳ upper Bound.

So above example query :-

select * from Order details where quantity between 5 and 30

↳ better query than previous.

Ex :- return all such order details when quantity is less than 10 or greater than 30

↳ Now this is equivalent to :-

quantity NOT in b/w 10 and 30

So here we'll be using the NOT operator.

Hence the query :-

```
Select * from orderdetails where
    quantity NOT between 10 and 20
```

The query can also be written as :- w/o using between operator :-

```
where quantity <10  OR  quantity >20
```

→ Ex: find all customers details who are from country either germany or UK or USA

↳ referring the customers table :-

↳ query :- Select * from customers where country = 'Germany' or country = 'UK' or country = 'USA'.

→ NULL in RDBMS :- Null is representation of No any value of a column. or of a row corresponding to some attributes

↳ Ex :- Now selecting all those rows from itemmaster where itemp is Null :-

So query: Select * from itemmaster where itemp is NULL

Here we cannot use equal to (=) with NULL.
Instead we have to write 'is' — " —

→ when selecting rows having No Null :-
then will use NOT keyword with Null.

↳ query :- Select * from Itemmaster where itemp is NOT NULL

→ limit :- used to limit No. of fetched records from a huge DB table

↳ Ex :- like fetch only top 4 records from products table :-

query :- Select * from products limit 4.

↳ Now if we want to skip first few records and we want next few records :-

so here :- limit 2, 4 will be written.

skip first two records

fetch next four records

Ex :- fetch 4 records after starting 5 records.

↳ so the query :- select * from products limit 5, 4.

→ Aggregate functions :- performs a calculation on multiple values and returns a single value

↳ These ignore NULL value

→ Some of the functions :- max, min, sum, avg, count.

Ex:- Select min(price) from products

↳ selecting minm price of all the products.

↳ similarly we can have the maxm of the price also.

Likewise sum(price) and avg(price) also

↓ sum of all the prices ↓ avg. of all the prices

→ count function :-

Select count(price) from products

↳ Now this will return the No. of rows entered in the price column!

→ Now referring to itemmaster table :-

So query :- Select count(itemp) from itemmaster

↳ Now this query will return the value of count = 2.

Because :- aggregate functions just ignore the NULL values!

→ Max, min, sum & avg will work only for Numbers and count can work on any type of value.

→ Ex:- No. of products having price greater than 50

query :- Select Count(price) from product where price > 50.

→ IN Operator :- This allows you to specify multiple values in a WHERE Clause.

→ The IN operator is a shorthand for multiple OR conditions.

→ The IN operator is used more with Subqueries also.

Ex:- So we want to have details of all such customers who are from germany or UK or USA :-

So query :- Select * from customers where country IN ("Germany", "UK", "USA")

→ Alias :- Used to give a temporary name to a table or a column in a table.

→ Not renaming :- Just another Name to the table or column → Just like the Nick Name

→ used to make table Names or column names more readable.

→ Only exists for the duration of the query

→ created with the Keyword : AS

Ex:- Select * from Customer AS cus.

then after this query:- you can use your tablename as cus until the expiry of this query.

like:- Select cus.Customername from Customer as cus.

→ we can also use Alias with column Name also:-

Select price AS P1 from products
where P1 > 30

↳ here before executing the where clause:- alias will be picked up.

15/08/2022. # Lesson 09 #

SQL Joins : Needed to retrieve the records from more than one table collectively.

→ when the reqd data is distributed over multiple tables :- then we combine those tables and then we can retrieve those reqd data.

→ types :- we have 6 types of Joins. will discuss now

→ Now why Joins? → when we want the collective data from multiple tables, then Joins are useful!

Now one condn must be satisfied :- when you Join two or more tables then atleast one column must be same in Both tables.

↳ common column should be there.

Now consider the following two tables :- which we'll be using for illustration of various types of Joins.

Student.

R.No.	Name	F_Name
1	Sumit	Suresh
2	Amit	Naresh
3	Priya	Manesh
4	Neha	Dinesh.

Library

R.NO	BOOK ID	Book Name
1	B1	Harrpotter
3	B2	gawin
~~B~~5	B3	CLRS

#01. Inner Join :

Syntax :-

Ex :-

Select ~~columnName(s)~~ *
from Student inner Join Library
on Student. RollNo. = library. RNo.

↓

Select Column Name (s)
from table 1 inner join table 2
on table1. column = table2. column.

↳ Here is the cond^n of innerjoin.
that on which column → you want to join these two tables.

→ So this query will match every row of 1st table with every other row of 2nd table. And it will check the entire common ~~column~~ column :- and it will match with the common entries.

↳ and only the matching records will come out.

So the result of above ex. query :-

RNo.	Name	FName	BookID	Book Name
1	Sumit	Suresh	B1	Hamacher
3	Priya	mahesh	B2	Galvin

→ Now if you want to print only the Roll no. :-
then you have to specify with the dot operator.

Now, the same thing can be written in another way also :-

```
Select columnName(s) from table1, table2
where table1.column = table2.column.
```

Now Both queries are same → Both are Inner Join!
Both will match the common column entries and will give us the entries of common rows Based on common col.

→ So Basically :- Join Combines two tables Horizontally and returns records.

→ and whichever you are writing the table first :- that will be first printed.

#02. Left Join (Left Outer Join) :-

This says that :- all the info from the left table and only matching info from right table Based on the common column.

→ complete info from left table.

So Ex:- we want to have the info of all the students and if any student have issued Books from library then that info also :-

So the query :- Select columnName(s) from table1 left join table2 on table1.column = table2.column

Syntax ⇒

and the query reqd for our Ex:-

```
Select * from student left outer join library
on student.RNO. = Library.RNO.
```

↳ so left table (student) will come as it is and only matching records of right table (library) will be shown.

So the resultant table:-

RNO.	Name	F.Name	RNO.	BookID	BookName
1.	Sumit	Suresh	1.	B1	Ramaynar
2.	Amit	Naresh	NULL	NULL	NULL
3.	Priya	Mahesh	3.	B2	Galwin
4.	Neha	dinesh	NULL	NULL	NULL.

Now, Rollno.5 = dNE in student. So no matching.

Right Join (right Outer Join).

↳ Here the right table's all records are fetched and left table's only matching records are fetched.

Syntax :-

```
Select columnName(s)
from table1 right join table2
on table1.column = table2.column.
```

↳ right table will come completely & only matching records of left table.

table1: left table
table2: right table.

Now the query of right Join of our two tables :

```
Select * from student Right outer Join
library ON student. RNo. = library. RNo.
```

Now the resultant table :

RNO.	Student Name	F_Name	RNO.	BookID	Book Name.
1.	Sumit	Suresh	1.	B_1	Hamacher
3.	Priya	mahesh	3.	B2	galvin
NULL	NULL	NULL.	5.	B_3	CLSR

↳ Not matching with right table so entries will be NULL.

Full Join (full outer Join) :- Here we will have all the data from left table & from right table and common - common entries will be together.

Syntax :-

```
Select ~~columnNames~~ (s) *
from ~~table1~~ student Full outer Join ~~table2~~ library
on student. RNo. = library. RNo.
```

So the resultant table :-

RNo.	Name	FName	RNo.	BookID	BookName
1.	Sumit	Suresh	1.	B1	Hamacher
2.	Amit	Naresh	NULL	NULL	NULL
3.	Priya	Mahesh	3.	B2	galvin
4.	Neha	dinesh	NULL	NULL	NULL
NULL	NULL	NULL	5.	B3	CLSR

Cartesian Product :- No common column Needed
↳ No joining cond^n

Now here all records of table 1 X all records of table 2. will be fetched.

Here student table = 4 records and library = 3
so total = 4X3 = 12 records will be fetched.

Here in cartesian product : we don't have any keyword. will write directly

So Syntax :-

```
Select columnNames(s)
from table1, table2
```

Self Join :- If one table is involved in the Join.
↳ if within the Same table :- if two columns are matching then self join comes into the picture.

↳ Here you cannot write the cond^n directly. Here you have to write with the help of aliasing

Syntax :

```
Select columnName(s)
from table1 t1, table1 t2
where t1.column1 = t2.column2.
```

Here you can think like :- t_1 and t_2 are like two pointers which are pointing to same table.

Now consider the following Employee table :-

Employee

E.NO.	Name	ManagerID
1	sumit	NULL
2.	Amit	4
3.	Priya	4
4.	Neha	1

Self Join on diff. columns of same table.

Now we want to know all those Employees who are manager and we want their Names.

So the query will be like :-

```
Select E1.Name from
Employee E1 , employee E2
where E1. ENO. = E2. Manager ID
```

→ Now, w/o aliasing → we cannot do this Self Join.

Ex :- Consider the itemmaster table :

item	item p
1	2
2	4
3	NULL
4	NULL

item p → parent item

Now find all ids of items which are items and parents also.

So the query :-

```
Select i1. item from itemmaster i1,
itemmaster i2 where
i1. item = i2. itemp.
```

→ if you Have following two tables :-

table t_1	table t_2
No. of col. = C_1	= C_2
No. of rows = R_1	= R_2

then cartesian product :-

No. of columns = $C_1 + C_2$

No. of rows = $r_1 * r_2$.

group By Clause :- Used in collaboration with the Select statement to arrange identical data into groups.

↳ when we don't want the complete table info and we only Some Selected data in such a way that it should be grouped together Based on some criteria.

Now consider table :- Student (RNO, Name, FName, DOB).

Now we want those students who are sharing same DOB:

So the query: Select DOB, count (RNO.) from student group By DOB.

So this query will give us the No. of students having same DOB along with the DOB

like for this DOB → there are this many students.

→ So group By is used to make groups.

→ Now referring to the product table :-
So write a query to fetch sum of prices of all products for each of the Category.

↳ So the query :-

Select Category ID, Sum(price) from products group by Category ID.

→ write a query to fetch min^m of prices for all products for each of the category, only when the price of product is greater than 20.

↳ query :- Select Category ID, min(price) from Products where price > 20 group By Category ID.

So here the where clause will come first.

→ write a query to fetch the sum of prices of all products for each of the category, only when the sum of the price is greater than 200.

Soln. Here using the where clause on sum :- will give you the Error → So you cannot use the where clause with the Aggregate functions

So here you have to use another clause :- Has Having clause

Having Clause

The Having clause in SQL is used if we Need to filter the result set Based on aggregate functions such as MIN(), MAX(), SUM(), AVG() & count().

The Having clause was introduced because where clause does not support aggregate functions.

Also, Group By must be used before the Having clause.

So the query of previous question:

```
Select Category ID, Sum(price) from products
group By Category ID Having Sum(price)>20
```

→ Sequence: from ⇒ where ⇒ group by
⇓
Select ⇐ Having
↳ (at last)

→ NOW WHERE V/S HAVING :-

⟶

Having clause	WHERE clause
1. The Having claus checks the condn on a group of rows	1. The where clause checks the condn on each of the row
2. used with aggregate functions	2. cannot be used with aggregate functions.
3. Having clause is executed after the group By clause	3. The WHERE clause is executed Before the group By clause.

→ consider the table :-

Employee

EID	EName	Dept.	Sal.

Que. Find total salary taken By employees of each dept. only when total salary exceeds 1 Lakh.

Soln Select dept., Sum (sal.) from Employee
group By dept.
Having Sum (Sal.) > 100000

16/08/2022

Lesson 10#

consider the table:-

itemmaster

item	itemp
1	2
2	4
3	Null
4	Null
Null	4

grouped together.

Now the query is written :-

Select itemp count(item) from itemaster group by itemp.

then the result of above query:-

itemp	Count (item)
2	1
4	2
Null	2

→ Count : it does not ignore Null and the remaining aggregate functions will ignore NULL, since they need only the Numerical value.

Order By clause :- This keyword is used to Sort the result set in ascending or descending order.

when we want to arrange the result set of the query in either ascending or descending order.

Based on only one column.

Ex: fetch all customers records but sorted by country Name.

So query :- `Select * from Customers order By country.`

So Based on the country column :- The resultant table will be in sorted order → Based on only on country.

→ default :- the result will be sorted in ascending order. and if you want to be very specific :- then mention 'asc' keyword for ascending order and 'dsc' for descending order.

Ex: fetch all customers records but sorted by country Name in descending order :

Now the query :- `Select * from Customers order By country dsc.`

→ Now can we fetch records which are sorted for two columns? → So here Based on two columns we are sorting our result table.

→ if the Country is same for few records :- then they will be sorted Based on their city name!

query: `Select * from Customers order By country, city` → city will come into picture only when we have same country for few entries.

→ ~~As customer country~~

→ So our table will be sorted Based on country column first → then for same country :- the table will be sorted ~~in~~ Based on City.

and 'asc' or 'desc' :- mentioned just after the columnName

→ So in descending order of both columns :- we'll just mention "desc" keyword ajust after the columnName :-

```
Order By country desc, City desc.
```

→ So Order By :- it runs at the very Ending.
↳ after the select command.

→ So sequence :

from → alias → where → group by → Having
↓
order By ← select

itemmaster

item	itemp
1	2
2	4
3	Null
4	Null

item of the itemp value 4.

query :

```
Select item from
itemmaster where
itemp = 4
```

Now like we want to find out all those customers who are from same country as the country of the one of the customer = Alfred

(Refer the customer table).

→ So firstly we want to find out that Alfred is from which country :- then we'll search for the other customers who belong from the same country as that of alfred.

→ So this is where the subqueries come into picture.

Subquery : A Subquery or Inner query or a Nested query is a query within another SQL query & embedded within the where clause

↳ when in WHERE clause :- we don't have the column's value for comparision then in where clause we write one more query which will fetch us the reqd value for comparision!

↳ So here first Inner query will run then the outer query.

→ Two types :-

1. Single row subquery :- Inner query returns only 1 row (value).

2. Multiple row subquery :- Inner query returning more than one row.

Now Ex:- refer the product table:

find all such products which are having price less than the price of product name 'Tofu'

So query :-

```
Select * from product where
price < (select price from
product where productName='Tofu')
```

→ from Inner query: only one value we have retrieved!

Ex: find all such products ⊆ are having price greater than avg. price of products.

Soln

```
Select * from products where
price > (Select avg(price) from product)
```

Ex: find the product with 2nd higest price.

Soln So here we want to find out max^m of all those prices which are lesser than the Max^m of all prices of the column.

query:

```
Select productName, max(price) from products where
price < (select max(price) from product)
```

Now doing same with order By :-

```
Select * from product order By price desc
limit 1,1
```

Ex: Now find the product with 3rd Highest price.

```
Select product Name, max (price) from product
where price < ( select max (price) from
                products where price < ( select
                max (price) from product))
```

↳ Here Subquery ke andar ek aur subquery

→ Now it's Not at all Necessary that the outer query and Subquery will run on same table. tables may be different for outer query and inner query!

Now in case of multiple row Subquery :-
you cannot compare directly :-
you cannot compare multiple values with another set of multiple values of Subquery.

→ So we have Some operators to be used here :-
→ 1. IN 2. Any 3. ALL 4. EXISTS
four operators used for comparision.

Ex: consider the Customer table and Suppliers table.
↳ So find all customers that are from the same countries as the suppliers.

query →

```
Select * from Customer where
country IN
distinct
(select country from suppliers)
```

→ This will list out all the countries of suppliers table

and IN operator will work like OR operator

and each country will be compared with that of customer's country

and matching records will be fetched.

↳ and if we write equal to (=) instead of IN operator than only single country value will be picked up from suppliers table and remaining legit values (which we wanted) will be left out.

So we have to use IN for comparing multiple values with another set of multiple values!

Ex: find all such customers that are from those countries where there is no any suppliers

query:- select that customer, in who's country there is no any supplier.

↳
```
Select * from Customers where
country NOT IN (select distinct country
from Suppliers)
```

Ex: consider the Orders table :-

Orders (OrderID, customerID, Employee ID, OrderDate, Shipper ID)

Now find all the customers, who have placed more than 2 orders.

Soln. Here we'll need customers table for customer details

```
Select * from Customers where
customerID IN ( Select customerID
from orders groupby customerID
having count (orderID) > 2)
```

DPP of Lesson 06

Ques.1 consider the customer's table.

Ques(i) Select * from Customers where country IN('Germany', 'Sweden')

(ii) Select CustomerName, address, city, postal code, country from customers where contactName = 'Yang Wang'

(iii) Select * from customers where customerID <= 19

(iv) Select * from customers
where country NOT IN('Germany', 'UK', 'USA')

Que2. refer to the products table.

(i) Select * from products where
supplier ID IN (1, 2, 3)

(ii) Select product Name from products where
price ~~as~~ Between 5 and 25
or
distinct price >=5 and price <=25

(iii) Select supplier ID from products where
~~product~~ ~~category~~ ID = 2.
category

* iii → can also be done with the help of subquery also

(iv) select * from products where
supplier ID = 2 AND price >30

(v) select * from products where
price > 50 AND supplier ID != 6

(vi) select * from products where
price <30 AND (supplier ID != 2 OR
supplier ID != 6)

DPP of Lesson 9

Eleminating that name who we already know.

customerName <>
'Around the Horn'

Ques1. refer the Customers table

(i) Select ~~customerName~~ customerName from customers where AND country = (select country from customers where customerName = 'Around the Horn')

(ii) select postalCode, count(customerID) (or count(*)) from customers group by postalCode.

(iii) Select country, count(customerID) from customers group by country order by count(customerID) desc limit 1.

select country, max(cust_count) ~~or~~ from (select country, count(customerID) AS cust_count from customers group by country

(iv) Select country from customers ~~where~~ group by country Having count(customerID) = 1.

Ques2. refer products and suppliers table.

1. select * from products where productName != 'Aniseed syrup' And price = (select price from products where productName = 'Aniseed syrup')

2. select supplierName from Suppliers where supplierID IN ~~where price > 20~~ (Select supplierID from products where price > 20)

3. select * from products where productName <> 'Tofu'
Category ID = (select category ID from products where productName = 'Tofu')

17/08/2022

Lesson 11

Now :- one drawback of using IN :-

the drawback is :- that IN operators works like :-
[equal to and OR] → (=)
[drawback :- IN operator ~~can~~ only has "Equal to" comparision.]

_____ IN (_____)
↓
subquery :- the values obtained from the subquery are compared with the multiple values fetched from outer query & the comparision is done only on the Basis of the equal to (=) and OR operator when we are using the IN.

so drawback :- Using IN :- we can not have comparision Based on other Relational operators = [>, <, <=, >=]
The IN Operator don't work for them.
→ so for these other relational operators we have :- [ANY & ALL] operators!

→ ANY with Subquery :-

Select columnName(s) from TableName
WHERE columnName [Operator] ANY
(select columnName from TableName
WHERE Cond^n);

Now here while using any :- this operator is very important to be mentioned :- like what type of comparision you want to have?

So this operator must be a standard comparision operator → (=, <>, !=, >, <, >=, <=)

So the query should look like :-

columnName > any (———)
— " — >= any (———)
— " — <= any (———)
— " — != any (———).

Ex: Consider the OrderDetails table
Orderdetails (orderDetailID, OrderID, productID, Quantity)
and also the products table (ProductID, ProductName, SupplierID, CategoryID, Unit, price)

So: find the ProductName of all those products which have their orders qty larger than 50