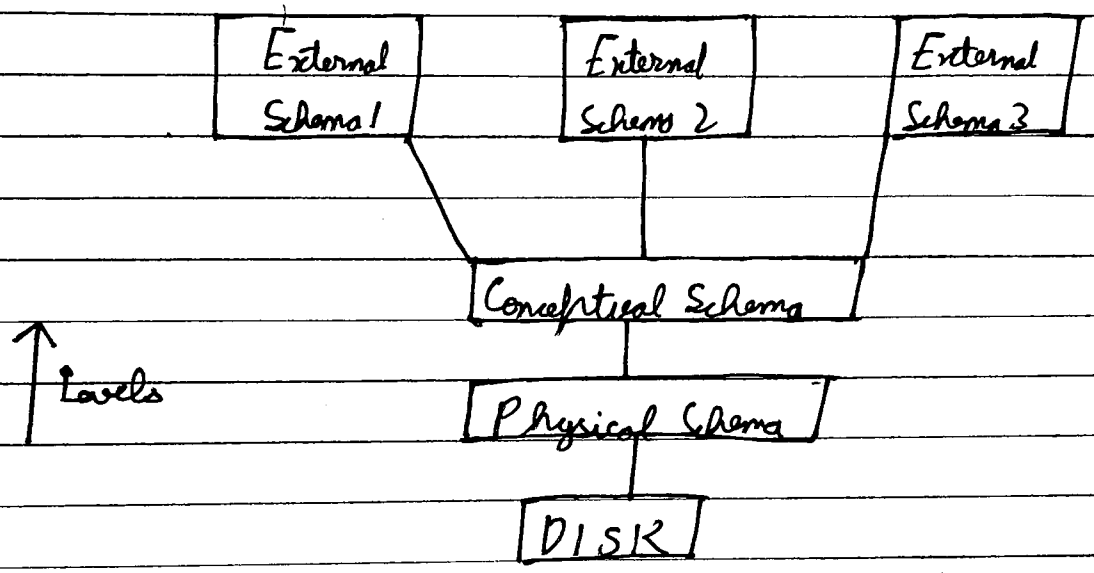
DBMS

⇒ **Instance of a Database:**
Collection of information stored in the database at a particular moment is called an instance of database.

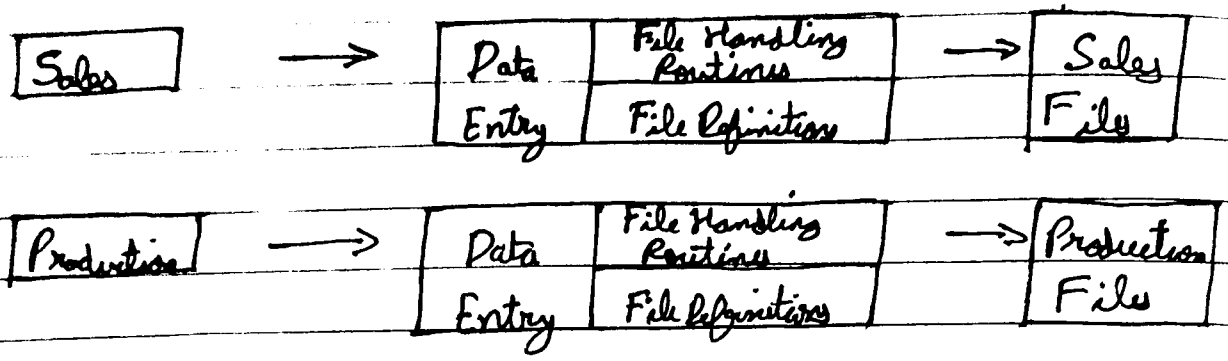
⇒ **Schema of a Database:**
The overall design of the database is called a database schema



⇒ **Data Independence**
1. Physical Data Independence
2. Logical Data Independence

The ability to modify a schema definition in one level without affecting a schema definition in the next higher level is called data independence.

▷ File System



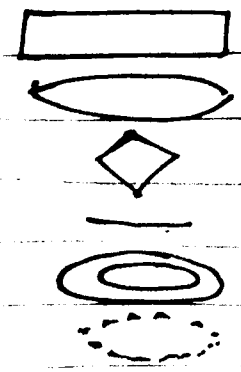
▷ Advantages of DBMS :

1. No data duplication / redundancy.
2. Easy and fast access of data.
3. Specification of intergrity constraints.
4. Atomicity of the transaction.
   ↓
   Logical Unit of Work
   (Either everything or nothing)

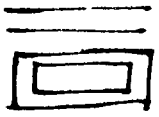
▷ Briefly write about types of DataBases.

▷ ER-DIAGRAM (Entity Relationship Diagrams)
↳ Represents logical design in pictorial form

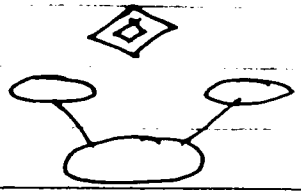
ER-Notations :



Entity Set
Attributes
Relationship Set
Link
Multi Valued Attribute
Derived Attributes



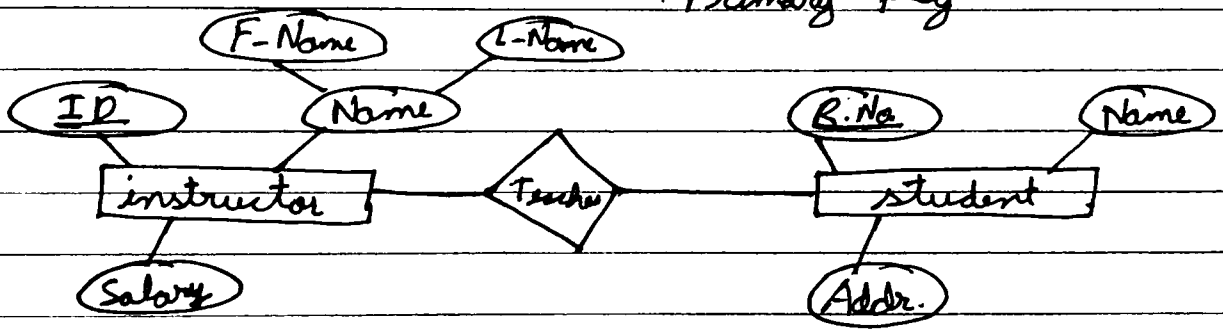
Total participation
Weak Entity Set
No Attribute can be depicted as primary key, dependent on Strong Entity Set



Weak Relationship

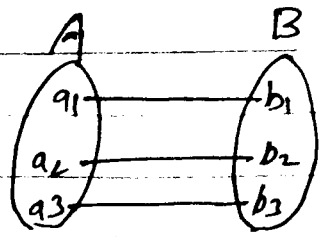
Composite Attribute
Can be further subdivided into smaller attributes

Key Attribute
↳ Primary key



▷ Mapping Cardinality :
Express the number of entities to which another entity can be associated via a relationship set.

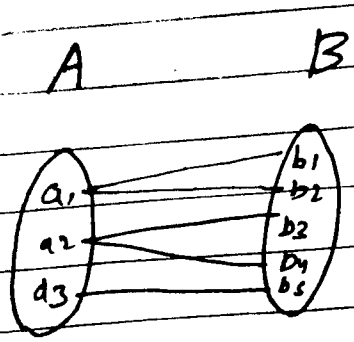
1. One to One



An entity in A is associated with almost 1 entity in B and entity in B is ~~some~~ associated with

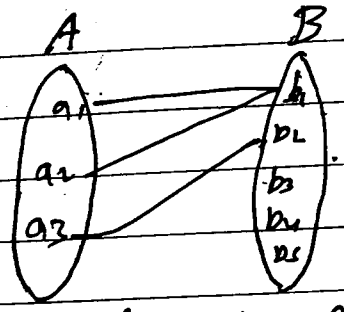
atmost one entity in A.

2. One to many



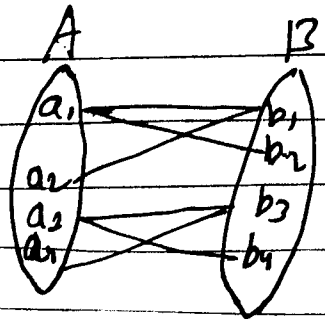
One element of set A ~~associated~~ ~~with~~ ~~many~~ ~~elements~~ are associated with many elements in set B.

3. Many to one:



One element of set B are associated with many of set A.

4. Many to many



Many element in set A are associated with ~~out~~ elements of set B.

▶ Representing Cardinality constraints in ER diagram

1. We can express cardinality constraints by either directed lines signifying "one" or undirected lines signifying "many" between relationship set and the entity set.

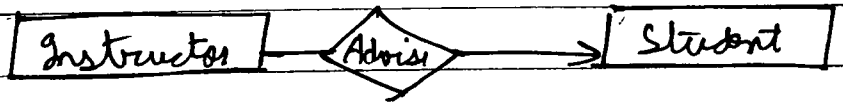
1) One to one

⟶ "One"

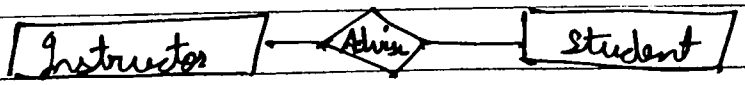
—— "many"

2) One to Many

3) Many to One



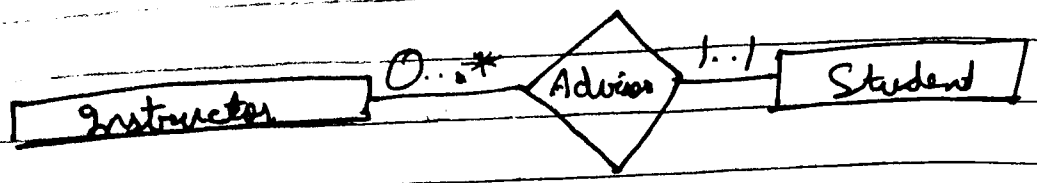
4) Many to Many



A line may have an associated minimum and maximum cardinality shown in the form of l:h where l represents minimum and h represents maximum participation.

1- A minimum value of 1 indicates total participation.

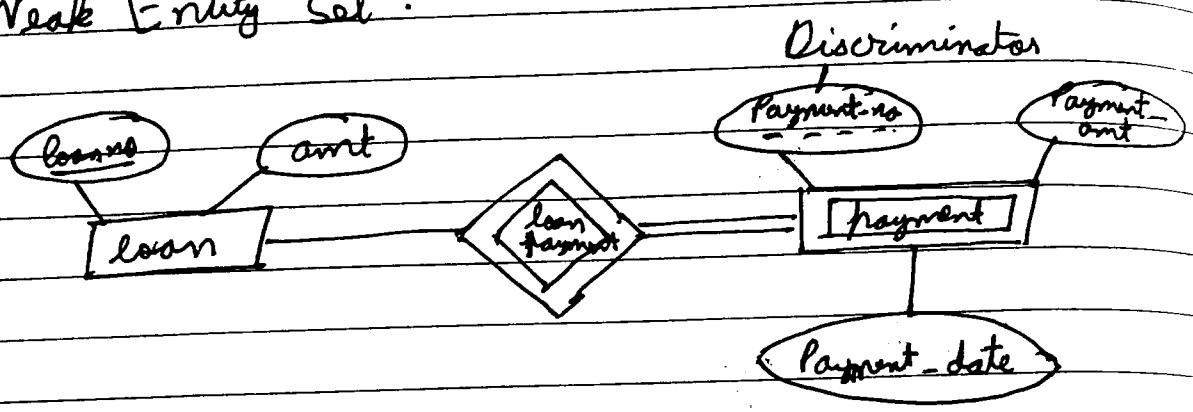
2- A maximum value of 1 indicates that the entity participates



in atmost1 relationship.

3. A maximum value represented by * indicates no limit.

▷ Weak Entity Set :



Payment

LoanNo	P_No.	P_Amt	P_date
101	1	x—x	31-Jul-24
102	2	x-x	31-Jul-24
103	3	x-x	31-Jul-24
104	4	2-2	31-Jul-24

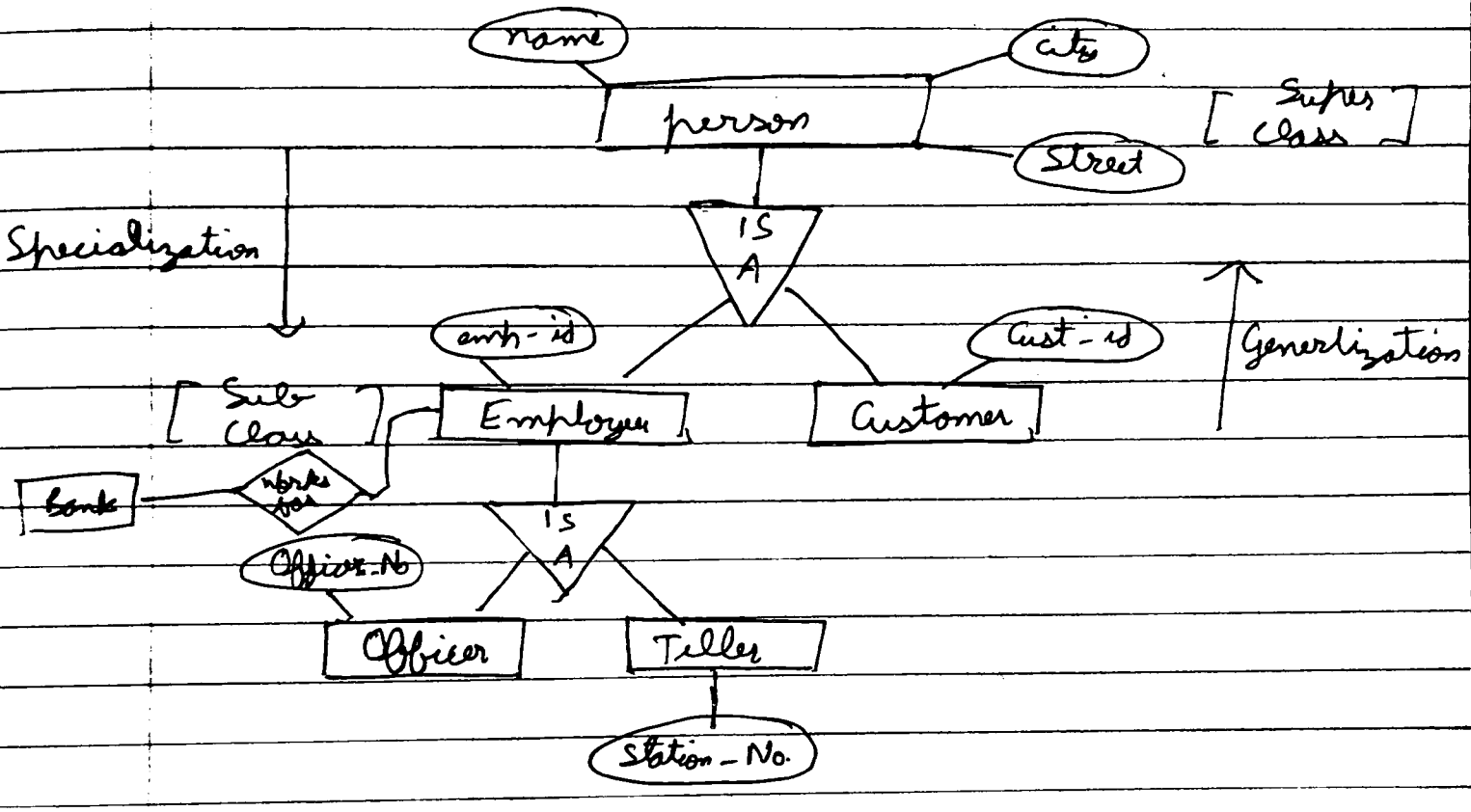
Loan

Loan_No	Amt
101	10000
102	11000
103	13000
104	14000

▷ Extended ER Specialization Features:

person = { name, street, city }

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



⇒ Generalization :

Customer : { name, city, street, cust-id }

Emp : { name, city, street, e-id, Salary }

Generalized (Person) = { name, city, street }

▷ Attribute Inheritance :

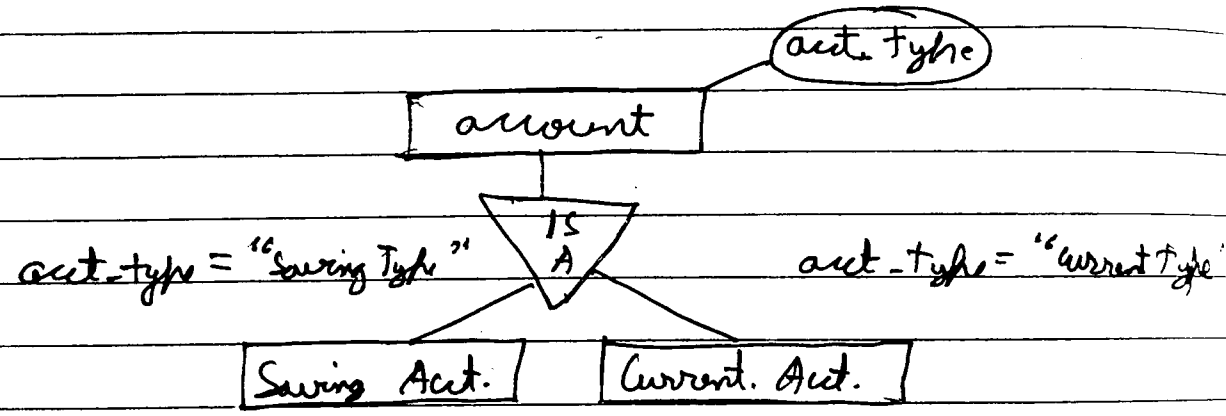
The attributes of the higher level entity set are said to be inherited by lower level entity set.

The lower level entity set also inherits participation in the relationship sets in which its higher level entity set participates.

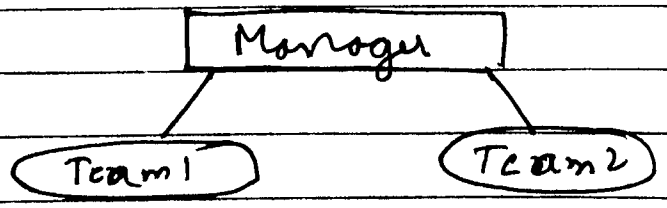
⇒ Constraints :

1- One type of constraint involves determining which entities can be members of a given lower level entity set.

Condition defined



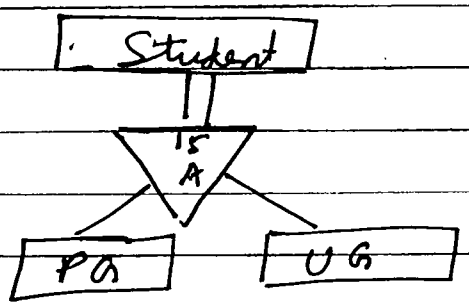
User defined :



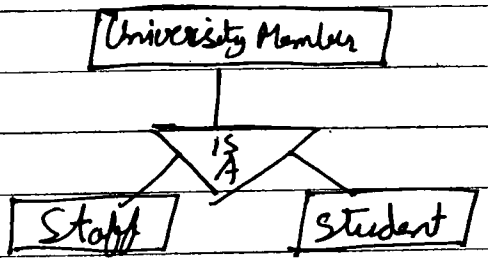
Manager will decide who goes into Team1 and Team2.

2. Second type of constraint relates to whether or not entities may belong to more than one lower level entity set within a single generalization.
Eg: Act type cannot be both Saving and Current, disjoint set.
Overlapping: eg: Organizing Manager can be part of both teams.

3. Final constraint:
Total generalization or specialization.

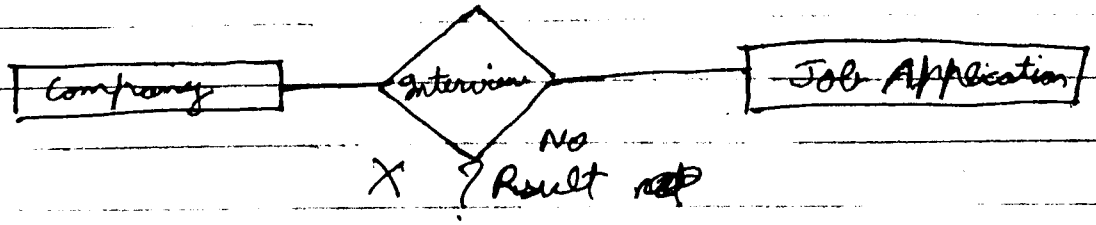


Partial Generalization

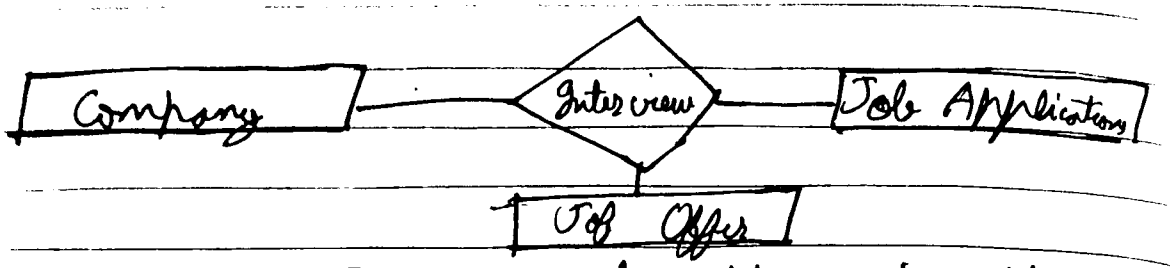


▷ Aggregation: Aggregation is an extraction in which relationship sets along with their associated entity set are treated as higher level entity sets and can participate in relationships.

a)

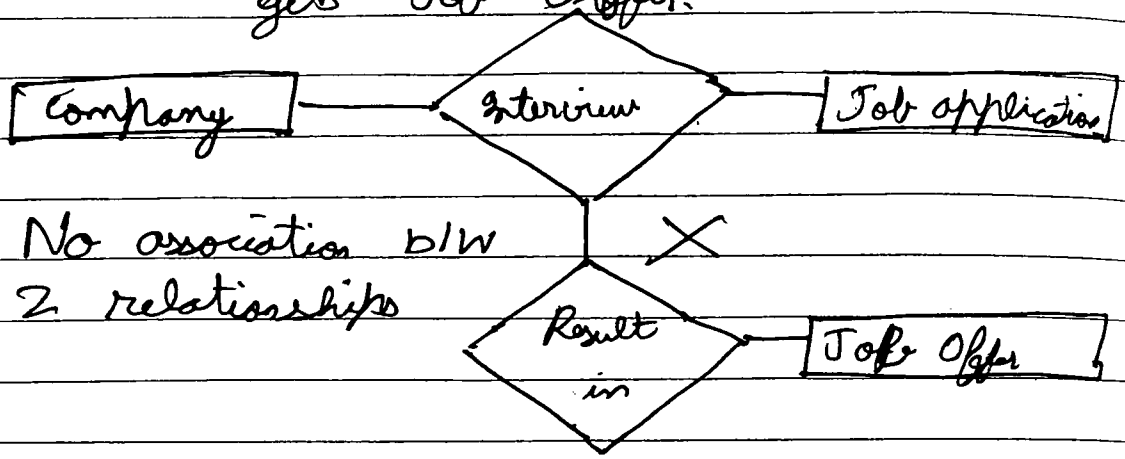


b)



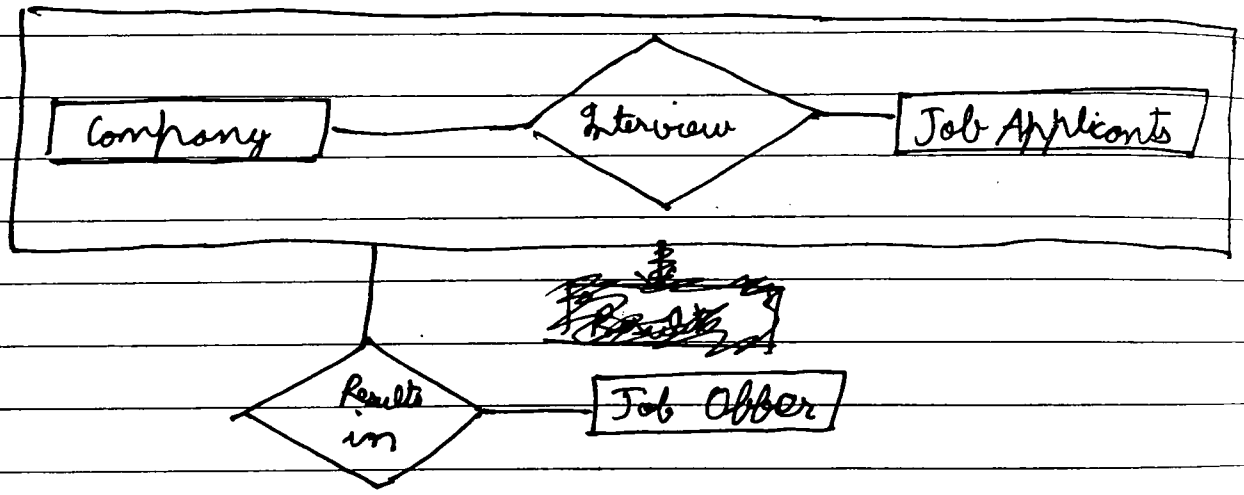
× Everybody who appears in interview gets Job Offer.

c)



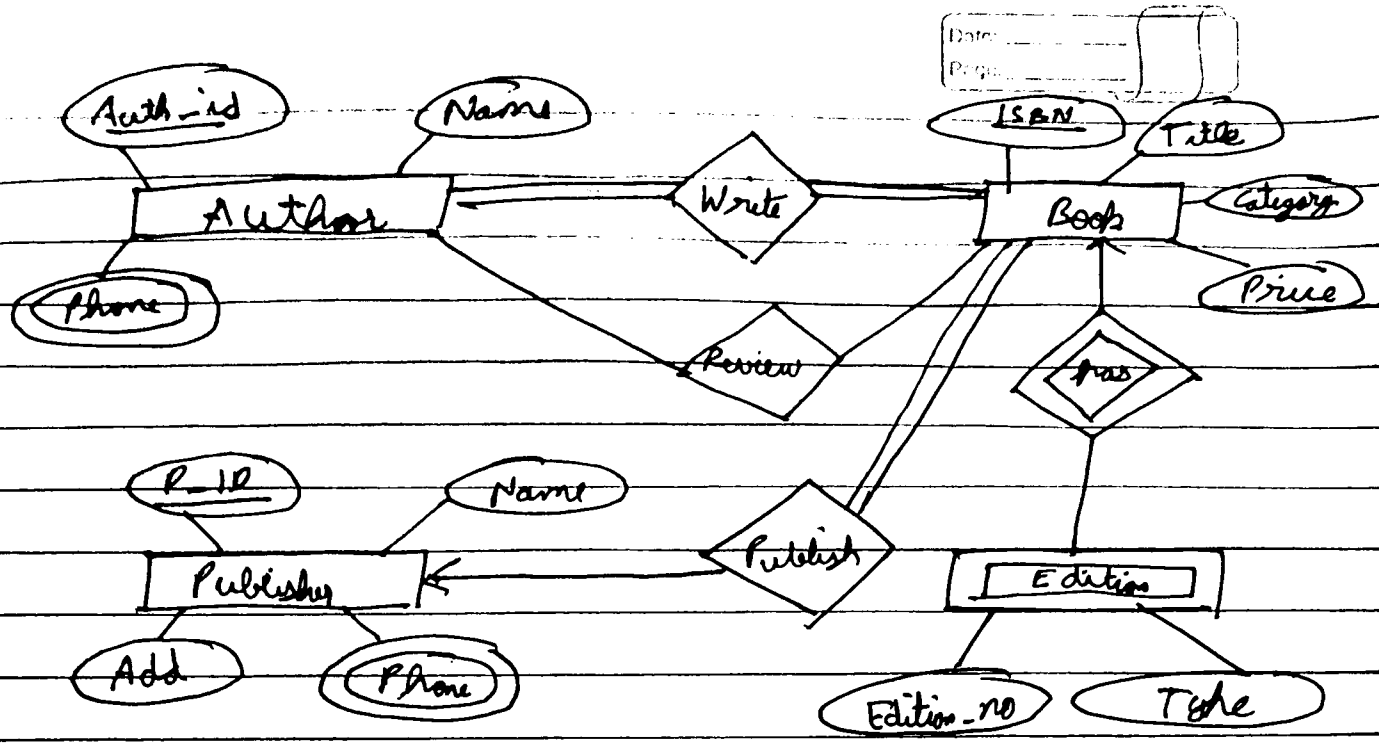
★ No association b/w 2 relationships

d)



▷ Design an ER diagram:

Author	Author	m writes m	Books
Book	Author	m review m	Books
Publisher	Book	1 has m	Editions
Edition	Book	m publish 1	Publisher



=> Sale_Id (5)
Product_Id VChar (10)
Quantity Sold Int
Sale Date Date
Price Number (10, 2)

Product_Id VChar(5)
Product_Name VChar (20)
Category VChar (20)
Unit_Price Number (10, 2)

▷ Codd Rules:

RDMS

1- Information Rule:
1- Data are presented as values within columns within rows.

2. Guaranteed Access Rule:
Every value can be accessed by providing table name, column name and key.

3. Systematic Treatment of NULL values:

Date:
Page:

1. Separate handling of missing or non applicable data. This is distinct to zeroes.

4. Relational online catalog:
A relational database must provide access to its structure through the same tools that are used to access data.

Catalog / Data Dictionary can be queried by authorized user as part of the database.

5. Comprehensive data Sublanguage:

Supports Data Definition, Data Manipulation, Security, Integrity Constraints and transaction processing.

6. View Updating Rule:

All theoritically possible view updates should be possible.

Emp = {Name, Addr, Phone, Salary, Increments}.

7. High level insert, update or delete

8. Physical Data Independence:

9. Conceptual Data Independence

10. Integrity Independence.

11. Distribution Independence:
Application should work in a distributed data base

12. Non- Subversion Rule:
Security and Integrity of the database must not be violated.

⇒ Group By
Group by statement is often used with aggregate set to group result of 1 or more columns.

Syntax:
```
Select ColNames
From TableName
Where Cond
Group by Col Names
Order by Col Name.
```

⇒ Having Clause
Having Clause was added to SQL because the where clause cannot be used with aggregate functions.
List number of customers in each country and include countries with more than 5 customers

⇒
```
Vchar (20
Job Vchar(20)
Date (Date)
Int
Int.
```

Date:
Page:

Relational ~~Algebra~~ Algebra

1. Procedural Language.
User instructs the system to perform a sequence of operation on the database to compute the desired result

2. Non-Procedural Language.
User describes desired information without a specific procedure for obtaining that information.

I. 1- Relation Algebra is a theoritical language, relation algebra.

2- Relational algebra is a procedural query language
3- It consits of set of operations that takes one or two relations as input and produce a new relation as result.
4- Fundamental operations in relation algebra are select, project, union, set difference, cartesian product and rename

II- Fundamental Operation
1. Select
1. We represent Select (σ) statement by Sigma (σ)
2. The select operation selects tuple that satisfies a given predicate.
(σ) denotes selection
3. The predicate appears as a subscript to sigma.
4. The argument relation is in parenthesis after the sigma (σ).

loan = {loan-no, branch-name, amt}

1- Select those tuple of the loan relation where the branch name is 'Perryridge'.

$\Rightarrow \sigma_{branch\text{-}name = \text{'Perryridge'}} (loan)$

2. Find all tuples in which the amount loan is more than 1200.

$\Rightarrow \sigma_{amt > 1200} (loan)$

AND ($\wedge$) & OR ($\vee$)
NOT ($\neg$)

3. Find those tuple pertaining to loan of more than 1200 and made by Perriridge branch.

$\Rightarrow \sigma_{amt > 1200 \wedge branch\text{-}name = \text{'Perryridge'}} (loan)$;

[1] Project operation

1. A project operation is a unary operation that returns its argument relation, there are certain attributes left out.
2. Projection by denoted by Greek letter (T C) Pi.
3. We list both attributes that we wish to appear in the result as a subscript to pi.
4. The argument relation follows in parenthesis

1- Write a query to list all loan numbers and the amount of the loan.

$\Pi_{loan\text{-}no, amt} (loan)$

Cust = { cust_name, cust_street, cust_city }

1- Find those customers name who live in Mumbai.

$\Pi_{cust\_name}(\sigma_{cust\_city = \text{'mumbai'}}(cust))$

3- Union :

depositer = { cust_name, acc_no }
borrow = { cust_name, loan_no }

1- Find the names of all bank customers who have either an account or loan or both.

$\Rightarrow \Pi_{cust\_name}(borrower) \cup \Pi_{cust\_name}(depositer)$

⇒ Conditions :

For a union operation $r \cup s$ to be valid, we require that 2 conditions hold.

1- Relation r & s must be of the arity that is they must have same number of attributes.

2- The domains of the ith attribute of r and the ith attribute of s must be the same for all i's.

4- Set difference : (−)

1- The set difference operation denoted by (−) allows us to find the set that are in one relation but are not in another.

2. The expression $r - s$ produces a relation containing those tuples in r but not in s.

Depositor		Borrower	
Cust_name	Acc_No	Cust_name	Loan_no
Hayes	A-102	Adams	L-16
Johnson	A-101	Curry	L-93
Johnson	A-103	Hayes	L-15
Jones	A-217	Jackson	L-14
Lindsay	A-222	Jones	L-17
Smith	A-215	Smith	L-11
Turner	A-305	~~Williams~~ Smith	L-23
		Williams	L-17

1- Find all customers ~~of~~ in bank who have account in bank but not taken any loan.

→ $\Pi_{cust\_name}(\text{Depositor}) - \Pi_{cust\_name}(\text{Borrower})$

5. Cartesian Product (X):

1- We write the cartesian product of relation as $r_1 \times r_2$
2- A relation is a subset of a cartesian product of a set of domains
3. Since the same attributes may appear in both r_1 and r_2 we need to devise a naming schema to distinguish between these attributes.

borrower = { cust_name, loan }
loan = { loan_no, branchname, amt }
borrower × loan = { b.cust_name, b.loan, b.loan_no, l.branchname, l.amt }

Date:
Page:

Loan

loan_no	b_name	amt
L-11	Round Hill	900
L-14	Down Town	1500
L-15	Perryridge	1500
L-16	Perry ridge	1300
L-17	Down Town	1000
L-23	Red wood	2000
L-93	Mianus	500

Borrower

Cust_name	loan_no
Adams	L-16
Curry	L-93
Hayes	L-15
Jackson	L-14
Smith	L-11
Smith	L-23
Williams	L-17
Jones	L-17

1- We want to find names of all customers who have a loan at the Perry ridge branch.

Cust_name	Borrower loan_no	loan loan_no	b_name	amount
Adams	L-16	L-11	Round Hill	900
Adams	L-16	L-14	Down Town	1500
Adams	L-16	L-15	Perryride	1500
Adams	L-16	L-16	Perryridge	1300
⋮				
Hayes	L-15	L-15	Perry Ridge	1500

$\sigma_{b\_name = \text{'Perry ridge'}}$ (Borrower × loan)

⇒

Cust_name	b_loan_no.	l_loan_no.	b_name	Amount
Adams	L-16	L-15	Perryridge	1500
Adams	L-16	L-16	Perryridge	1300
Curry	L-93	L-15	Perryridge	1500
⋮	⋮	⋮	⋮	⋮
Hayes	L-15	L-15	Perryridge	1500

$\Rightarrow \Pi_{cust.name}(\sigma_{b.loan\_no = l.loan\_no}(\sigma_{b.name = 'Perryridge'}(Borrower \times loan)))$

$\Rightarrow$ Pattern Matching : LIKE Operator

Wildcards :

% : Represents (zero, 1) or multiple characters

_ (Underscore) : Represents 1 single character.

$\Rightarrow$ Select * from Customers
where Name LIKE 'A.%'

▷ Natural Join ($\bowtie$)

1. It is often desirable to simplify certain queries that require a cartesian product.
2. Usually a query that involves a cartesian product includes a selection operation on the result of a cartesian product.

Q- Find the names of all the customers who have a loan at the bank along with the loan number and loan amount.

$\Rightarrow \Pi_{cust\_name, l.loan\_no, amt}(\sigma_{b.loan\_no = l.loan\_no}(Borrower \times loan))$

$\Rightarrow \Pi_{cust\_name, l.loan\_no, amt}(Borrower \bowtie loan)$

I Natural join is a binary join that allows us to combine certain selection and a cartesian product into one operation

II - Natural join operation forms a cartesian product of its 2 arguments performs a selection forcing equality on those attributes that appear in both relations

schemas and finally removes duplicate attributes.

▷ IN & BETWEEN Operators

IN: 1- Used when we are Multiple values in a where clause.
2. Is a short stands for multiple or conditions

Syntax:
Select col name
From Table Name
where ColName IN (Value1, Value2, ...);

⇒ Select * from Cust
where country = 'INDIA' or Country = 'Brazil' or country = 'UK'
or country = 'US'

⇒ Select * from Cust
where Country IN ('India', 'Brazil', 'UK', 'USA');

BETWEEN: 1. Selects value within a given range.
2. Values can be number, text or date.
3. The between operator is inclusive that is begin and end values are included in the result.
Select details of those product whose Price is between 100 and 200 rupees.
Select * from product where Price between 100 and 200;

▷ Rename Operator (ρ) rho

1- Given a relational algebra expression E:
$\rho_x(E)$ returns result of expression E under the

rame x.

2: Assume that a relational algebra expression E has arity n then expression $\rho_x(A_1, A_2, \ldots, A_n)$ returns result of expression E under the name x and attributes rename to $A_1, A_2 \ldots A_n$.

D Find the largest account balance in the bank

Acc-no.	Branch_name	Balance
A-101	Downtown	500
A-215	Mianus	700
A 102	Perryridge	400

Strategy:

1- Compute first a temporary relation consisting of those balance that are not the largest.
2- Take the set difference between the relation: $\Pi_{balance}$ (account) and temporary relation just computed to obtain the result.

A. acc-no.	A. Branch-name	A. Balance	D. Acc-no	D. Branch-name	D. Balance
A-101	Downtown	500	A-101	Downtown	500
A-101	Downtown	500	A215	Mianus	700
A-101	Downtown	500	A-102	Perry Ridg	400
A-215	Mianus	700	A-101	Downtown	500
A-215	Mianus	700	A-215	Mianus	700
A-215	Mianus	700	A-102	Perry Ridge	400
A-102	Perry Ridg	400	A-101	South Town	500
A-102	Perry Ridg	400	A-215	Mianus	700
A-102	Perry Ridg	400	A-102	Perry Ridge	400

$\Pi_{a.balance}(\sigma_{a.balance < d.balance}(account \times \rho_a(account)))$

↳ a. balance

500

400

400

Π balance (account)

500

200

400

Π balance (account)	— Π a. balance
500	500
200	400
400	400

▷ Division operator : $(\div)$

1. The division operator is used for queries which involve the all operator.

2. $R_1 \div R_2$ means ~~as~~ tuples of R_1 associated with all tuples of R_2.

Account:

acc_no.	b_name	balance
A-101	Downtown	500
A-102	Perryridge	400
A-201	Brighton	900
A-215	Mianus	700
A-217	Brighton	750
A-222	Redwood	700
A-305	Round hill	350

Depositor

C_name	Acc_no.
Hayes	A-102
Johnson	A-101
Johnson	A-201
Jones	A-217
Lindsay	A-222
Smith	A-215
Turner	A-305

Branch

branch_name	branch_city
Brighton	Brooklyn
Downtown	Brooklyn
North Town	Rye
Redwood	Palo

1. Find all customers who have an account at all the branches located in city Brooklyn

Step 1: Obtain all branches in city of Brooklyn

$$\gamma_1 = \Pi_{branch\_name}(\sigma_{branch\_city = \text{"Brooklyn"}}(Branch))$$

Step 2: Find all customer names and branch for which customer has an account at a branch.

$$\gamma_2 = \Pi_{c\_name, b\_name}(Account \bowtie Depositor)$$

Step 3: We need to find customers to appear in γ_2 with every branchname in γ_1.

$\sigma_3 = \Pi_{cust\_name} (\sigma_2 \div \sigma_1)$

▷ Outer Join :

emp

e_name	Street	City
Coyote	Toon	Hollywood
Rabbit	Tunnel	Carrot Villa
Smith	Bell Road	P. Run
Williams	Sea View	Seattle

ft_works

e_name	b_name	Salary
Coyote	Mesa	1500
Rabbit	Mesa	1300
Gates	Redmond	5300
Williams	Redmond	1500

1. Generate a single relation with all the information about employees.

$\sigma_1 =$ emp ⟗ ft_works

e_name	Street	City	b_name	Salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrot Villa	Mesa	1300
Williams	Sea View	Seattle	Redmond	1500

▷ Left Outer Join (⟕)

emp ⟕ ft_works

left outer join takes all tuple in left relation and that did match with any in right relation, fill left over with NULL

Emp-name	Street	City	b-name	Salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrot Ville	Mesa	1300
Williams	Sea View	Seattle	Redmond	1500
Smith	Bell Road	D.Den	NULL	NULL

▷ Right outer Join (⟖)

The right outer join is symmetric with left outer join. It inserts tuple from right relation that did not match any from left ~~with~~ and add them to result.

Emp-name	Street	City	b-name	Salary.
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrot Ville	Mesa	1300
Williams	Sea View	Seattle	Redmond	1300
Crater	NULL	NULL	Redmond	5300

▷ Full Outer Join (⟗)

The full outer join does both the operations inserting tuples from the left relation that did not match any from the right relation as well as tuples from the right relation and adding them to the result of join.

Emp-name	Street	City	b-name	Salary
Coyote	Toon	Hollywood	Mesa	1500
Rabbit	Tunnel	Carrot Ville	Mesa	1300
Williams	Sea View	Seattle	Redmond	1500
Crater	NULL	NULL	Redmond	5300
Smith	Bell Road	D.Den	NULL	NULL

Date:
Page:

▷ Modification of Database

Deletion in Relational Algebra:

We can delete whole tuples, We cannot delete values or any particular attributes.

In relational algebra a deletion is expressed by $r \leftarrow r - E$ where r is the relation and E is a relational algebra query.

1- Delete Smith's record from the table depositor.

$$Depositor \leftarrow Depositor - (\sigma_{c\_name = 'Smith'}(Depositor))$$

2- Delete all loans with amount in the range 0 to 50.

$$Loan \leftarrow Loan - (\sigma_{amount \geq 0 \text{ and } amount \leq 50}(Loan))$$

▷ Insertion:

The relational algebra expresses an insertion by $r \leftarrow r \cup E$ where r is the relation and E is a relational algebra.

1. We wish to insert the fact that Smith has 1200 in account A-973 at Perryridge branch.

$$Account \leftarrow Account \cup (A\_973, Smith, 'Perry Ridge', 1200)$$

▷ Updating

→ Suppose that the interest payments are being made and all taxes are to be increased by 5%.

$$Account \leftarrow \Pi_{acc\_no,\ b\_name,\ balance * 1.05}(account)$$

→ Suppose the balance is over 1000, receive 6% interest whereas all other receive 5% interest.

$$account \leftarrow \Pi_{acc\_no,\ b\_name,\ balance * 1.06}(\sigma_{balance > 10000}(account))$$

$$\cup$$

$$account \leftarrow \Pi_{acc\_no,\ b\_name,\ balance * 1.05}(\sigma_{balance \leq 10000}(account))$$

▷ Generalized projection

Credit_Info.

C_Name	Limit	Credit_balance
Curry	2000	1750
Hayes	1500	1500
Jones	6000	200
Smith	2000	400

1- Generalized Projection Operation extends the projection operation by allowing arithmatic functions to be used in the projection list.

2- The Generalized projection operation has the following form:

$$\Pi_{F_1, F_2 \ldots F_n}(E)$$

↳ Any relational Algebra Expression.

Represent arithmatic expression involving Constants and attributes in the schema of E.

1- Find how much more each person can overdraw.

$\pi_{\text{cname, limit} - \text{credit\_balance}}$ (credit info)

▷ Aggregate Fn's & Group by Clause

pt_works

e_name	b-name	Salary
Adams	Perryridge	1500
Brown	Perryridge	1300
Gopal	Perryridge	5300
Johnson	Downtown	4500
Peter	Downtown	2500
Rao	Austin	1500
Sato	Austin	1600

1- Find out the total sum of salaries of all part time employees in the bank.

Calligraphic G

$\mathcal{G}_{\text{sum(salary)}}$ (pt_works)

1- Calligraphic G signifies that aggregation is to be applied, the subscript specifies the aggregation function to be applied.

2- Find number of branches appearing in the part time works relation.

$\mathcal{G}_{\text{count(b-name)}}$ (pt_info) ×

$\mathcal{G}_{\text{count Distinct(b-name)}}$ (pt-info)

3. Find total salaries of all part time employees at each branch of the bank separately.

→ $_{b\_name}\mathcal{G}_{sum(salary)}$ (pt_works)

4. Find maximum salary of part-time workers of each branch in addition of sum of salaries in each branch.

$_{b\_name}\mathcal{G}_{sum(salary) \text{ and } MAX(salary)}$ (pt_works)

5. General form of aggregate function

$$_{G1, G2, \ldots Gn}\mathcal{G}_{F1(A1), F2(A2) \ldots Fm(An)}(E)$$

E - E is any relational algebra expression
G - Represents attributes on which to apply group by
Fn - Represent aggregate func.
An - Represent attribute.

D. Customer {C_ID, C_Name, Address, City, Country}

1. $\sigma_{country = 'germany' \cup country = 'UK'}$

2. Fetch customer address, city and country whose name is amit

$$\pi_{Address, City, Country}(\sigma_{C\_Name = 'Amit'}(Customer))$$

3. Fetch all detail of customers having C_ID 19

$$\sigma_{C\_ID = 19}(Customer)$$

4. Fetch all customer information having C_ID till 19.

$$\sigma_{C\_ID < 20}(Customer)$$

D Supplier {S_ID, S_Name, Country}

1. Find all customers that are from those countries where there is no supplier.

$$\pi_{Country}(Customer) - \pi_{Country}(Supplier)$$

D Student {R_No, S_Name, Address}
Teachers {T_ID, T_Name, T_Subject}
College {RNO, T_ID}

1- Find the name of the teacher who teaches CO to John.

→ $\pi_{T\_name}(\sigma(S \bowtie C \bowtie T))$

$\pi_{T\_name}(\sigma_{S\_Name='John' \text{ AND } T\_subject='CO'}(S \bowtie C \bowtie T))$

2- Find the name of teacher who teaches who teaches compiler

$\pi_{T\_name}(\sigma_{subject='compiler'}(Teacher))$

3- Insert a new tuple into teacher table.

Tuple : T10, XY2, CO

$Teachers \leftarrow Teachers \cup ('T10', 'XY2', 'CO')$

- Passenger (pid, pname, pgender, pcity)
- agency (aid, aname, acity)
- bus (bid, bdate, time, src, dest)
- booking (pid, aid, bid, bdate)

1. Give the details of buses from dehradun to delhi.

→ $\sigma_{src='Dehradun' \text{ And } dest='Delhi'}(bus)$

2. Find the bus_id for the passenger with P_id 'P04' for bus to Delhi before 20/5/24.

$\gamma_1 = (Passenger \bowtie booking \bowtie bus)$

$\pi_{b\_id}(\sigma_{P\_id='P04' \text{ AND } DEST='Delhi' \text{ AND } bdate<'20/5/24'}(\gamma_1))$

▷ Multirow Sub-queries

Any, or, exist

In Sql the operator Any, ALL, EXISTs are used in combination with sub-queries to perform comparison and logical comparison.

1- Any: The any operator is used in combination with comparison operator to compare a value with a set of values returned by subquery

It returns true if the comparison holds true for atleast one value in the set.

Syntax of any operator:

Select col Names
From Table Names
where col Name . operator ANY (Sub Query)

Emp

Id	name	Salary	Deptt
101	Jack	2000	HR
103	Mark	6000	Developer
104	Peter	4000	Tester
105	Tom	2000	HR
107	Roger	5300	Account
108	Mike	2000	NULL
109	Paul	4000	Developer
110	Hon	2000	Account

1- Write an SQL to get all employees from the emp table where salary is equal to any salary in the HR department.

select *
from emp
where Salary = ANY (select Salary from Emp where dept = 'HR');

2- ALL: The all operator is used in combination with comparison operators to compare a value with a set of values returned by a sub Query.

- It returns true if the comparison holds true for all values in the set

Syntax: Select col_Name
From Table_Name
Where col_Name operator ALL (SubQuery)

2- Write an SQL query to retrieve all employees whose salary is greater than the salary of all employees in the account department.

Select *
from emp
where Salary > ALL(Select Salary from Emp where dept = 'Account')

3. Exists Operators:
a. The exist operator is used to test the existance of any record in the subquery. If the subquery returns true, the main query is evaluated.
Syntax: Select col Names
From Table Name
where exists (Sub Query);

Date:
Page:

▷ Cust

Cust_id	Name
C_1	John
C_2	Jane
C_3	Michael
C_4	Emily
C_5	David

Order

id	Cust_id	Total amt.
1	C_1	150
2	C_1	75
3	C_2	200
4	C_3	350
5	C_3	100
6	C_5	125
7	C_1	50

1- Write an SQL query to retrieve the names of the customers who have placed an order.

→ Select Name
from Cust
where EXIST (Select *
from Orders
where order.cust_ID = cust.cust_ID;

4. NOT EXIST → Reverse of EXIST
5. IN → More than 1 answer
NOT IN → Reverse of In

Sub Queries (Nested Queries)

=> Select col names
from Table Name
where expression operator (Select col Names from Table name)

Outer Query — Inner Query

MAX. 255 Inner Queries

1. A Sub Query is a select statement that is embedded in clause of another select statement.
2. The sub Queries or inner queries executes once before the main query.
3. The result of the Sub Query is used by the main query or the outer query.
4. You can fill ~~from~~ outer statement out of simple ones by using sub. Queries.
5. You can place the sub query in ~~as~~ number of SQL clauses like where clause of having clause.

☆ Here can single or multi row operator.

D. Keys :

Emp_id	Name	Aadhar	Salary	Phone	Email
101	Hari	0000	45000	123456	ABCEDEF
102	Ram	1111	50000	654321	NULL
103	Hari	2222	45000	NULL	AEDEFGH
104	Raju	3333	42000	145632	AFG EED A

1. Super key : is like a superset (of attributes), uniquely identify the tuples, can contain NULL values, can contain extraneous attributes (composite key), super key contains all possible combination of attribute

2. Candidate Key: Minimal Super key is called candidate key
Emp_id, aadhaar, email, {emp_name, phone}

3. Primary Key: No duplicate values, no NULL value, only one primary key
Empid, aadhar

4. Alternate Key: Used in case first primary key fails.
Aadhar can be used as alternate for emp_id.

5. Unique Key: Multiple unique keys, no duplicate values, NULL allowed.
- Phone, Email

6. Composite primary key: {emp_name, phone} can be used as primary key if it is ensured phone cannot be null.

▷ PK constraint defined at col. level.

Syntax:

Colname datatype size Primary key

```
Create Table Cust_mstr(
    Cust_No  Varchar(10)  Primary key,
    Frome    Varchar(20),
);
```

▷ PK constraint defined at table level.

Syntax:

Primary key (<col_name1>, col name2 . . .);

1- Create a table FD_Master where there is a composite primary key mapped to columns FD_SERNO, CORP_CUST_No

```
Create Table FD_MSTR (
    FD_SER_No  Varchar(10),
    Branch_No  Varchar(20),
    CORP_CUST_No  VARCHAR(20)
    :
    PRIMARY KEY(FD_SER_NO, CORP_CUST_NO));
```

▷ Passenger { pid, pname, pgender, pcity }
Agency { aid, aname, acity }
Bus { bid, bdate, time, src, dest }
Booking { pid, bid, aid, bdate }

$\sigma_{src = \text{'D. Bus'} \wedge dest = \text{'Delhi'}}$ (bus)

$\sigma_{pid = \text{'p04'} \& dest = \text{'Delhi'} \wedge bdate < \text{''}}$ (bus ⋈ booking)

Q. Find details of all male passenger with agency known as Amit Travels.

$\sigma_{pgender = \text{'male'} \wedge aname = \text{'Amit Travels'}}$ (passenger ⋈ booking ⋈ agency)

▷ Emp

E_name	L_Name	SSN	D_No
Josh	Smith	123	3
Alice	Brown	456	3
Jeremy	Jones	789	2
Olivia	William	999	3

Project

Pname	PNo	DNo	P_Loc
X	10	3	Houston
Y	20	3	Houston
Z	30	2	Stafford

Dept

DName	DNo	Mgr_SSN
Research	3	999
Headquarter	2	789

D location

DNo	Location
3	Houston
2	Stafford

Works_On

SSN	PNo
123	10
789	30
456	20
9989	10
999	20

1- Retrieve the names of all employees who work headquarter department.

$\pi_{F\_name \text{ and } Lname}$ ~~(Emp ⋈ Dept)~~ $\sigma_{D\_Name = \text{'Headquarter'}}$ (Emp ⋈ Dept)

2. For every project located in Stafford list project no, manager name.

1- Set details of project that are located in stafford. ~~so project~~

$S\_Project \leftarrow \sigma_{P\_loc = \text{'Stafford'}}$ (Project).

2. $Ctrl\_Dept \leftarrow (Dept \bowtie S\_Project)$

3. $Dept\_Mgr \leftarrow (Ctrl\_Dept \bowtie Emp)$

4. $\pi_{PNo, DNo, FName, LName}$ $(Dept\_Mgr)$

3. Find the names of employees who work on all the projects controlled by DNo.3.

1. $Dept\_Proj \leftarrow \pi_{P\_No}$ $(\sigma_{DNo=3}$ (Project))

2. Find list of employees working on various projects.

$E\_Proj \leftarrow \pi_{SSN, PNo}$ (Works_On)

3. Since we have keyword all in the query we divide.

$K\_Proj \leftarrow E\_Proj \div Dept\_Proj$

4. $\pi_{Ename \text{ and } Lname}$ $(Emp \div K\_Proj)$

D. Foreign Key:

1. Foreign key represents Relationship between Tables.
2. Foreign key of the column whose values are derived from primary key of some other table.
3. The table in which foreign key is defined is called a foreign table or detail table.
4. A table that defines a primary key and is referenced by foreign key is called a primary table or a master table.
5. This constrain establishes a relationship between records across master and retail table.
6. This relationship ensures:
   a- Records cannot be inserted in retail table if corresponding record in master table donot exists.
   b- Records of the master table cannot be deleted if records in corresponding table actually exists.

Syntax:

```
Col Name Datatype Size REFERENCES <Table Name>
                                  [<Col Name>]
Create Table Emp_MSTR
(Emp_No Varchar(10) PRIMARY KEY,
Branc_No Varchar(10) REFERENCES Branch-MSTR
                                (Branch-No),
);
```

FK defined at Table Level:

Syntax: Foreign Key(Col Name1, Col Name2) REFERENCES
Table Name [(Col Name 1, Col Name 2)]

eg: Create Table Acct-IDC
ACCT_FD_NO Varchar(10),
Cust-No Varchar 2(10),
:
FOREIGN(Cust-no) REFERENCES ~~Select~~ Cust-MSTR (Cust-No);

▷ CHECK Constraints:

Create Table CUST_MSTR
CUST_No varchar (10) CHECK (CUST_No LIKE '_CY%');

▷ Functional Dependencies (FD's)
Functional Dependencies tell us how the two elements are related to each other.
Functional Dependency is a relationship between one attribute with another.

$X \rightarrow Y$

Determinant Dependents

	X	Y	$X \rightarrow Y$	
t_1	10	A	$t_1.x = 10$	$t_1.y = A$
	20	B	$t_2.x = 10$	$t_2.y = A$
	30	C		
t_2	10	A		

Rule:
$X \rightarrow Y$ is FD if it satisfies the following constraint
if $t_1.x = t_2.x$ then $t_1.y = t_2.y$ where
t_1 & t_2 are tuples in a relation

Types I-Partial FD
$X \rightarrow Y$ is partial FD.
If we remove any attribute from X, it doesn't violate FD rule.
$\{A, B\} \rightarrow C$
$B \rightarrow C$
$A \rightarrow C$

Date:
Page:

eg: {s_id, course} → S Name
s_id → S Name
Course → S Name

II- Fully Functional FD

X → Y is full FD.
if we remove any attribute of X it violates the FD rule.

{A, B} → C
B → C Y
A → C Y

Ex: S_id → S_name

III- Trivial FD

The dependency of an attribute on a set of attribute is known as trivial FD if the set of attribute include that attribute.

A → B is trivial FD if B is a subset of A.
A → A

Example: {S_id, S_name} → S_id
{S_id, S_name} → S_name

Student

Sid	Sname	Addr	Course
101	Sandeep	D. Dun	Python
102	Sandeep	D. Dun	Java
102	Saradhi	Vijayawada	Python
103	Saradhi	Delhi	C
104	Satvik	Hyderabad	Java
105	Ram	D. Dun	Python

1- Sid → Sname (Valid FD)
2- Sid → Address (Valid FD)
3- Sid → Course (Not Valid)
4. (Sid, Sname) → Course (Not Valid)
5. (Sid, Course) → SName (Valid FD)
~~6. (Sid, Course) →~~
6. (Sname, Course) → Sid (Fully Functional FD)

▷ Armstrong's Axioms

① Reflexivity

if B is a subset of A then
$A \rightarrow B$

② Augmentation

if $A \rightarrow B$
then $AC \rightarrow BC$

③ Transitivity

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

4. Self determination:
$A \rightarrow A$

5. Decomposition

if $A \to BC$

then $A \to B$

$A \to C$

6. Union

if $A \to B$ & $A \to C$

then $A \to BC$

7. Composition

if $A \to B$ & $C \to D$

then $AC \to BD$.

8. Pseudotransitive Rule

if $A \to B$ holds

& $CB \to D$ holds

then $CA \to D$ holds

Q- Suppose we are given relation

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

FD's (given)

$A \to BC$

$B \to E$

$CD \to EF$

Show that $AD \to F$ holds for R.

① $A \to BC$ (Decomposition)

then $A \to B$

$A \to C$

② ~~$B \to E$~~ $AD \to CD$ (Augmentation)

$A \to E$

③ $CD \to EF$ (Decomposition)

$CD \to F$

(2) $A \to P$ (Transitivity) holds for R.

D. Irreducible Set of Dependencies

$P = \{P\#, PName, Colour, Weight\}$

1- We define a set of FDs to be irreducible if and only if it satisfies the following 3 properties:

1- Right hand side of every FD in S ~~[illegible]~~ involves just one attribute.

2- The left hand side of every FD is irreducible, meaning ~~[illegible]~~ that no attribute can be discarded from the determinant without changing the closure set S+.

3. No FD can be discarded without changing the closure set.

I - $P\# \to PName$ ✓
$P\# \to Colour$ ✓
$P\# \to Weight$ ✓

II - $P\# \{PName, colour\}$ ✗
$P\# \to \{weight\}$
~~$P\# \to$~~

III - $\{P\#, PName\} \to Colour$ ✗
$P\# \to Weight$
$P\# \to PName$

D. Given relation R with attributes A, B, C, D.

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

$A \to BC$ $AB \to C$ FD's (Given)
$B \to C$ $AC \to D$
$A \to B$

Date:
Page:

→ Compute an irreducible set of FD's that is equivalent to the given set.

1- First step is to rewrite the FD's such that each has a singleton right hand side.

$A \to B$
$A \to C$
$B \to C$
$A \to B$ ✗
$AB \to C$
$AC \to D$

2- Attribute C can be eliminated from the left hand side of the FD

$AC \to D$ (given)

By we have FD

$A \to C$ (given)

By Augmentation

$A \to AC$

we are given

$AC \to D$

$\therefore A \to D$ (By Transitive rule)

3. We observe that the FD

$AB \to C$ can be dropped.

$AB \to CB$ (By augmentation)

$AB \to C$ (Decomposition)

4. $A \to C$ is implied.
can be derived
$A \to B$
$B \to C$
$A \to C$ {By transitive}

Final Set : $A \to B$
$B \to C$
$A \to D$

▷ Attribute Closure :

S S^+

If we want to find closure of S, it is represented by

$X^+ = \{ \ldots \}$
↳ Set of Attributes.

Q - $R = \{A, B, C, D, E\}$
$A \to B$ $C \to D$
$B \to C$ $D \to A$ } Given

Find the closure of attribute A

$A^+ = \{A, B, C, D\}$

$D^+ = \{D, A, B, C\}$

$E^+ = \{E\}$

$BC^+ = \{B, C, D, A\}$

Date:
Page:

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

$$\left.\begin{array}{l} A \to BC \\ BC \to DE \\ D \to F \\ CF \to G \end{array}\right\} \text{Given}$$

▷ $A \to B$

$A \to C$

$BC \to D$

$BC \to E$

$D \to F$

$CF \to G$

1- Closure of A

$$A^+ = \{A, B, C, D, E, F, G\}$$

$$D^+ = \{D, F\}$$

$$BCD^+ = \{B, C, D, E, F, G\}$$

▷ ~~Example~~ Canonical Cover

A ~~can~~ canonical cover is a set of functional dependencies that is minimum, irreducable and equivalent to orginal set of dependencies.

$R = \{x, y, z\}$

$x \to y$	Given
$x \to z$	
$z \to y$	

Find canonical cover.

$\cancel{X^+ = \{X, Y\}}$

① Find X^+ from given FD.

$X \to Y$

$X^+ = \{X, Z, Y\}$

Thus, $X \to Y$ can be dropped.

② Find X^+ from given FD

$X \to Z$

$X^+ = \{X\}$

Thus, $X \to Z$ cannot be dropped.

③ Find Z^+ from given FD

$Z \to Y$

$Z^+ = \{Z\}$.

Thus, $Z \to Y$ cannot be dropped.

▷ $R = \{A, B, C\}$

① $A \to B$
② $B \to C$
③ $A \to C$

Given

① For $A \to B$, find A^+

$A^+ = \{A, C\}$

② For $B \to C$, find B^+

$B^+ = \{B\}$

③ For $A \to C$, find A^+

$A^+ = \{A, B, C\}$

Thus $A \to C$ can be dropped.

Date:
Page:

▷ $R = \{A, B, C\}$

$A \to C$ | given.
$AB \to C$

1- For $AB \to C$, find AB^+
$AB^+ = \{A, B, C\}$. ✱ Minimal Cover
Thus it can be dropped.

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

$AC \to E$ | Given Minimal Cover.
$E \to D$
$A \to B$
$AC \to D$

1- For $AC \to E$, find AC^+
$AC^+ = \{A, C, D, B\}$

2- For $E \to D$, find E^+
$E^+ = \{E\}$

3. For $A \to B$, find A^+
$A^+ = \{A\}$

4. For $AC \to D$, find AC^+
$AC^+ = \{A, C, E, D, B\}$
Thus it can be dropped.

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

$ABC \rightarrow CD$

$BC \rightarrow D$

$A \rightarrow B$

$C \rightarrow D$

$ABC \rightarrow C$

$ABC \rightarrow D$

$BC \rightarrow D$

$A \rightarrow B$

$C \rightarrow D$.

1- $ABC \rightarrow C$, find ABC^+

$\{ABC\}^+ = \{A, B, C, D\}$ Can be dropped.

2. $ABC \rightarrow D$, find ABC^+

$\{ABC\}^+ = \{A, B, C, D\}$. Can be dropped.

3. $BC \rightarrow D$, find BC^+

$\{BC\}^+ = \{B, C, D\}$ can be dropped.

4. $A \rightarrow B$, find A^+

$\{A\}^+ = \{A, \}$

5. $C \rightarrow D$, find C^+

$\{C\}^+ = \{C\}$ can be dropped.

Date:
Page:

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

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

$A \to B$ ✗
$A \to C$ ✗
$B \to A$ ✗
$B \to C$
$C \to A$
$C \to B$.

1. $A \to \underline{B}$, A^+
$A^+ = \{A, C, B\}$ Can be dropped.

2. $A \to C$, A^+

$A^+ = \{B, C, A\}$ Can be dropped.

3. $B \to A$, B^+ $B^+ = \{B\}$

4. $B \to C$, B^+ $B^+ = \{B\}$

5. $C \to A$, C^+ $C^+ = \{C, B\}$

6. $C \to B$, C^+ $C^+ = \{C, A\}$

⇒ NORMALISATION

S = { S#, Sname, Status, city }

P = { P#, Pname, Colour, Weight, City }

SP (shipment) = { S#, P#, QTY }

SP =

S#	CITY	P#	QTY
S1	London	P1	300
S1	London	P2	200
S1	London	P3	400
S2	Paris	P1	200
S2	Paris	P2	400

1- The normalization procedure is reversible.
Reversibility is important because it means that normalization process is information preserving.

▷ Non-loss Decomposition and FD's.

S

S#	STATUS	CITY
S3	30	Paris
S5	30	Athens

S# → STATUS
S# → CITY

SST

S#	STATUS
S3	30
S5	30

SC

S#	CITY
S3	Paris
S5	Athens

(D) SST

S#	STATUS
S3	30
S5	30

Status	CITY
30	Paris
30	Athens

▷ Heath's Theorem:

Let R {A, B, C} be a relation where A, B, C are sets of attributes if R ~~stis~~ satisfies FD that

$A \to B$

& $A \to C$

then R is equal to the join of its projection on {A, B} & {A, C}

▷ More on FD's:

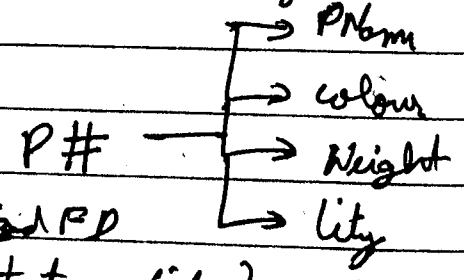
1- Left Irreducible FD's:

FD is said to be left irreducible if its left hand side is not too big.

2- P = {P#, PName, Color, Weight, City}

P# → PName
P# → Color
P# → Weight
P# → City

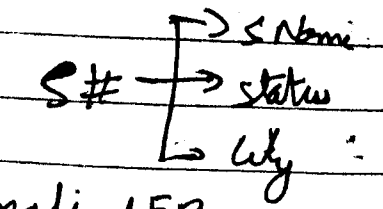
FD Diagram



Normalized FD

S = {S#, SName, Status, City}

S# → SName
S# → Status
S# → City



Normalized FD

$SP = \{S\#, P\# \ QTY\}$

$\{S\#, P\#\} \rightarrow QTY$.

$$\left.\begin{array}{l} S\# \\ P\# \end{array}\right\} \rightarrow QTY$$

~~Our FDs are normalized~~
Arrows should ~~go~~ originate from left hand side and nowhere else.

▷ Normalization :

1- Third Normal Form : (Informal Definition)

* If and only if elements are :

1 - mutually independent.

2 - irreducibly dependent ~~of~~ on primary key.

2- First Normal Form :

	P#	P Name	Color	Weight
✓	1	A	Red	10
✓	2	B	Blue	20
X	1, 3	A, C	B, G	200, 300

First Normal Form : A relation is FNF if and only if in every legal value of that relation every tuple contains exactly one value for each attribute.
This defination states that relations are always in first normal form.

First = { S#, STATUS, CITY, P#, QTY }

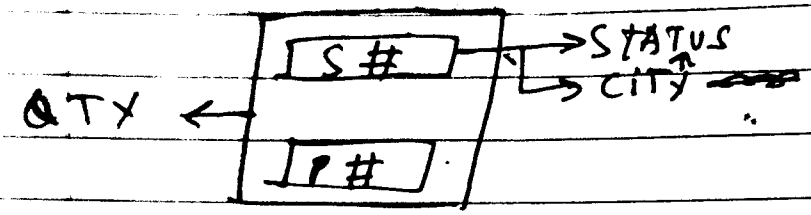
PK = { S#, P# }

S# → STATUS

S# → City

CITY → STATUS

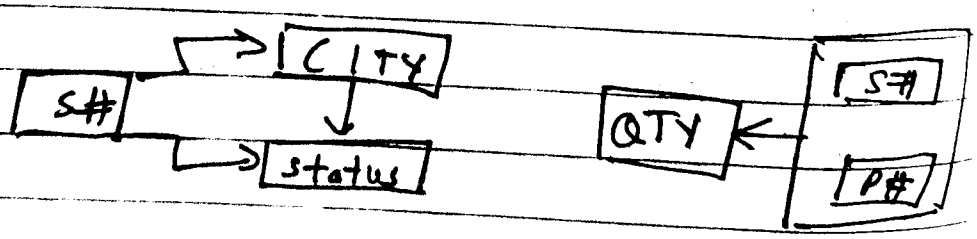
{ S#, P# } → QTY



S#	STATUS	CITY	P#	QTY
S_1	20	London	P_1	200
S_1	20	London	P_2	300
S_1	20	London	P_3	400
S_2	10	Paris	P_1	600
S_2	10	Paris	P_2	700
S_3	10	Paris	P_1	400
S_4	20	London	P_1	500
S_4	20	London	P_2	400

Second = { S#, Status, City }

S1 = { S#, P#, QTY }



Second

S #	Status	City
S1	20	London
S2	10	Paris
S3	10	Paris

SP

S#	P#	QTY
S1	P1	200
S1	P2	300
S1	P3	400
S2	P1	600
S2	P2	700
S3	P1	400

▷ Second Normal Form

A relation is in second normal form if and only if it is in first normal form and every non key attribute is irreducibly dependent on the primary key.

For table to be in second normal form
1- It should be in first normal form and it should not have partial dependency

▷ Second to Third normal form:

SC = { S#, CITY }

CS = { City, STATUS }

[S#] → [CITY]

[CITY] → [STATUS]

SC		CS	
S#	City	City	Status
S1	London	Athens	30
S2	Paris	London	20
S3	Paris	Paris	10
S4	London	Rome	50

A relation is in third normal form if and only if relation is in second normal form and every non key attribute and is not transitively dependent on primary key.

2. The second step in the normalization procedure is to take projections to eliminate transitive dependencies.

▷ Views :

⇒ Reason why views are to be created:

1. When data security is required.
2. When data duplicacy is to be kept to the minimum by maintaining data security.

Create view view name
Select col1, col2
From table name
Where col name = exp list.
Group by clause
Having clause

⇒ Select * from Vw- employee.
⇒ Drop View Vw- employee.

▷ BCNF Boyce Codd NF

A relation is in BCNF if and only if every determinant is a candidate key.

1- The difference between Third normal form and BCNF is that for every FD i.e. A->B 3rd normal form allows this dependency in a relation. If B is a candidate attribute and A is not a candidate key.
Whereas BCNF insists that for this dependency to remain in a relation, A must a candidate key.

BCNF is a stronger form of third normal form such that every relation in BCNF is also in third normal form.
However a relation in third normal form is not necessarily in BCNF.

Student	Subject	Teacher
ABC	DBMS	MNO
ABC	C	PQR
XYZ	DBMS	MNO
XYZ	C	STU

- {Student, Subject} → Teacher
  - Teacher → Subject

R {Student, Subject, Teacher}

R1 (Student, Teacher) FK

R2 (Teacher, Subject) PK

▷ Fourth Normal form: ☆ First make table BCNF then check for multi valued dependencies to convert into M.V.D IVth NF.

MCTX

Course	Teachers	Texts
Physics	Prof Green Prof Brown	Basic Mechanics Optics
Math	Prof Green	Basic Mechanics Vector Analysis Trigonometry

Date: _____
Page: _____

CTX

Course		Teacher	Text
Physics	C	Prof Green t_1	Basic Mechanics x_1
Physics	C	Prof Green t_1	Optics x_2
Physics	C	Prof Brown t_2	Basic Mechanics x_1
Physics	C	Prof Brown t_2	Optics x_2
Maths		Prof Green	Basic Mechanics
Maths		Prof Green	Vector Analysis
Maths		Prof Green	Trigonometry

The meaning of relation CTX is as follows:
A Tuple { Course : C, Teacher : T, Text : X }
appearing in CTX if & only if course : C can be taught
by teacher : t & uses text : x as reference.

Relation CTX satisfies a constraint:

If Tuple : (c, t_1, x_1) (c, t_2, x_2) both appear
in CTX then tuples (c, t_1, x_2) (c, t_2, x_1) both
also appear in CTX.

CT

Course	Teacher
Physics	Prof. Green
Physics	Prof. Brown
Maths	Prof. Green

CX

Course	Text
Physics	Basic Mechanics
Physics	Optics
Maths	Basic Mechanics
Maths	Vector Analysis
Maths	Trigonometry

MVD (CTX)
Course $\twoheadrightarrow$ Teacher
Course $\twoheadrightarrow$ Text

$A \twoheadrightarrow B$: B is multi dependent on A or A multi determine B.

⇒ Multi Valued Dependencies :
Let R be a relation and let a, b, c be subsets of the attributes of R then we say that B is multi dependent on A ~~and~~ ($A \twoheadrightarrow$) if and only if in every possible legal value of R the set of B values matching a given AC pair depends only on A value and is independent of C value
($A \twoheadrightarrow B | C$)

▷ IVth Normal Form :
Relation R is in IVth N.form if and only if whever there exists subset A and B of the attributes of R such that the non trivial MVD ($A \twoheadrightarrow B$) is satisfied then all attributes of R are also functionally dependent on A.