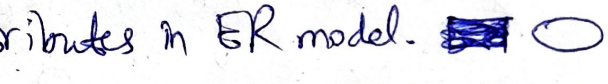
Entity Relationship Model

This model represents entities of a database i.e. helps in identifying entities to be represented in a database and also represents how those entities are related.

Rectangles → Represents strong entities in ER model that can exist by themselves. 

Double rectangles → Represents weak entities in ER model that need to be related to a strong entity to be defined, as by themselves they have no existence. 

Ellipse → Represents attributes in ER model. 

Double ellipse → Represents multi-valued attributes in ER model. 

Diamond → Represents relationships among entities. 

ER model

Entity	Attribute
Strong entity	Key attribute
Weak entity	Composite attribute
	Multi-valued attribute
Relationship	Derived attribute
One to one	
Many to many	
One to many	
Many to one	

Entity → It's an object with physical (car, house etc.) existence or conceptual (company, job etc.) existence.

Entity set → A set of entities, all of the same type.

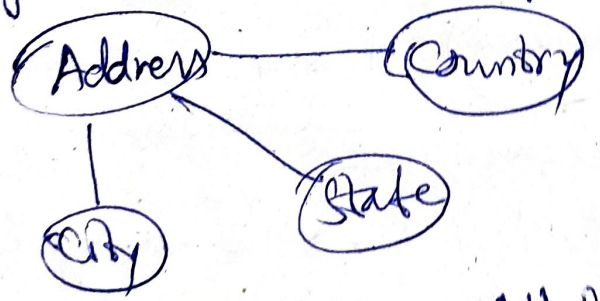
Strong entity → Eg: An employee of a company is a strong entity as he/she is directly associated with the company.

Weak entity → Eg: Employee's wife, child, parents, they are all dependants of the employee hence dependant on the company through employee.
But without the employee, they have no significance w.r.t the company, hence they won't exist.

Identifying Relationship → The relationship b/w weak entity and its identifying strong entity is called identifying relationship, shown by double diamond. ◇

Attributes → Properties that define the entity type.
Eg: Roll no; DOB, address that define student.

- Key attribute → Uniquely identifies entity.
- Composite attribute → composed of other attributes.



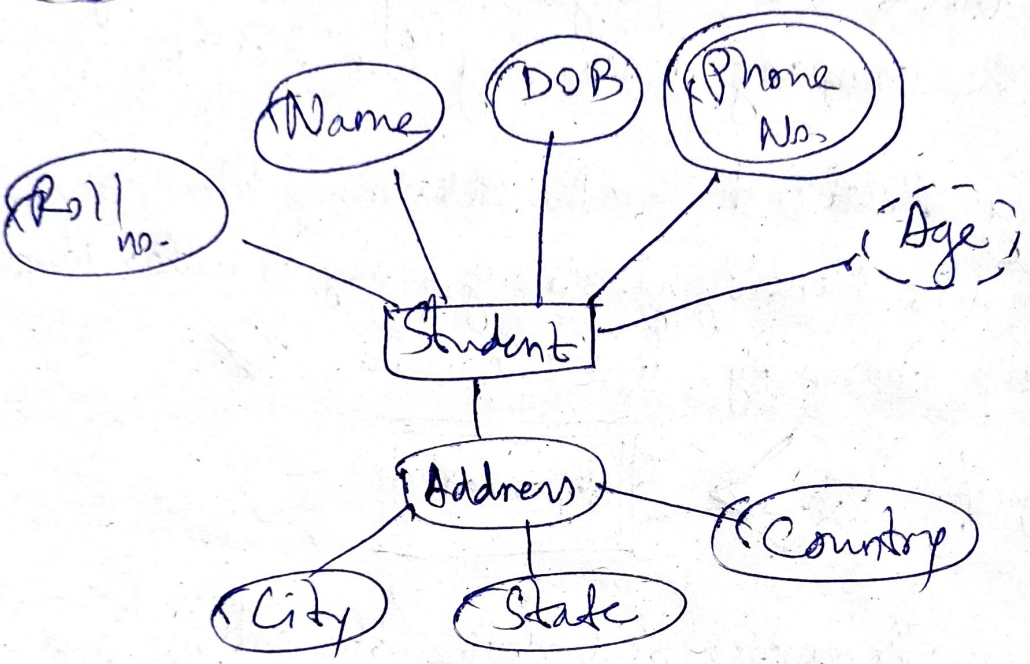
- Multivalued attribute → Attribute consisting of more than one value for an entity. Eg: a student can have multiple phone nos.

- Derived attribute :

On attribute that can be derived from another attribute of an entity. ~~Eg: DOB derived from Age~~

Eg: age derived using DOB

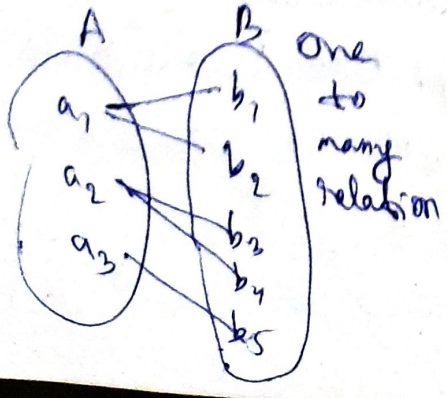
↳ Dotted ellipse



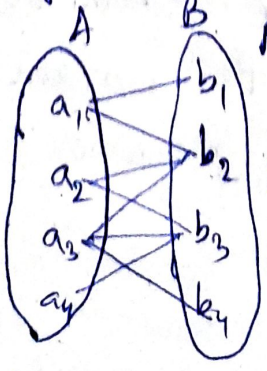
Binary relationship

Unary relationship

A no. of relationships b/w multiple entity sets is called a relationship set.



One to many relation



Many to Many relation

- Database System Concepts
  - Korth & Silberschatz
- Navathe
- Ramkrishnan Gehrke
- C J Date

Database Management System

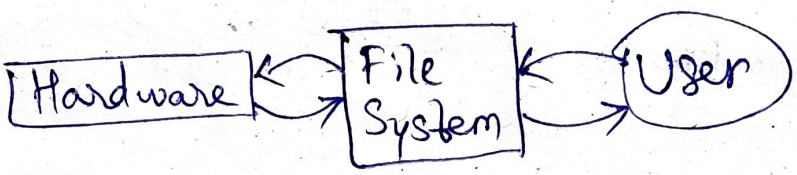
Data storage
Data retrieval (fetch)
Data updation / alteration

Structured data → Data present in fixed format

Data abstraction ←

4 levels of data abstraction :

① Physical



② Conceptual

Data is conceptualised in the form of a table with several columns for different parameters.

①, ② → Theory

③ View

It refers to the different levels of accessibility to a certain database, pertaining to who gets access to a certain data, and who's denied.

④ User

Final front-end view of data that's shown to the final user.

③, ④ → Lab

Database Design

1. ER model (Entity Relationship model)
2. Relational model

Entity → An object with physical existence, or conceptual existence.

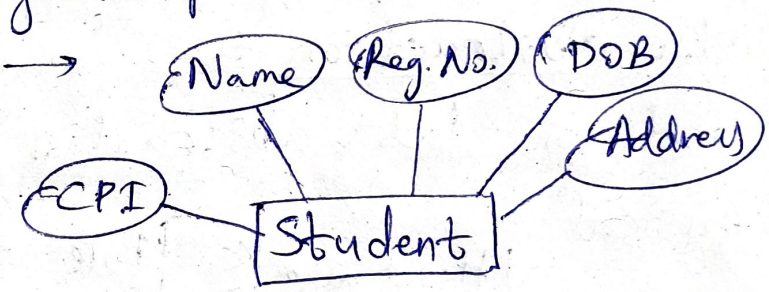
Collection of similar entities → Entity set

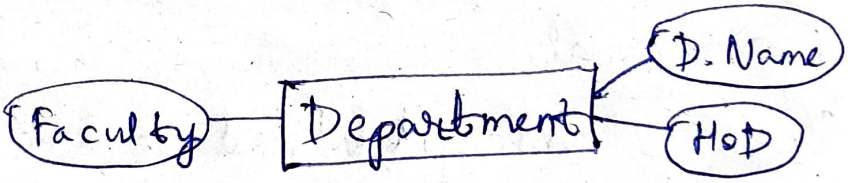
Every entity has certain distinguished features, attributes is another name for features.

Entity → □, attribute → ○

↑ rectangular shape ↑ oval shape

Dean Academic →
- Student
- Departments





Key attributes

1. Candidate key
2. Primary key
3. Foreign key
4. Super key

Key attribute → Uniquely identifies an entity.

For a student, Reg. No. is the primary key as for every student, Reg. No. is unique.

There can be several other attributes associated with Reg. No. but it'll still be a key, although the rest will be redundant.

Candidate key → Set of attributes for an entity such that it uniquely identifies the entity.

Primary key → The candidate key containing the minimum no.

Superkey → Along with essential attribute, it contains redundant attributes which can be removed, but the property of being a key still remains.

DBMS 2/8/23

Domain → The range of permissible values from which its corresponding attribute can accept any one, is called the domain of that attribute.

Foreign key → A key that relates to attributes belonging to different entities.

ER model → Entity
Relational model → Tuple
Table → Row
} All the Same

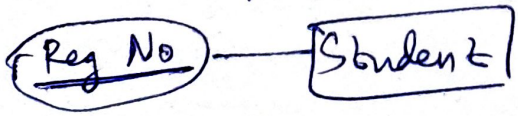
SQL Table :-

Student

varchar RegNo
varchar Name
varchar course
int sem

RegNo Primary Key

Underlined in ER model



Candidate Key :-

Sets of different attributes, where ~~every~~ all attributes in every set together uniquely identifies a student i.e. a row.

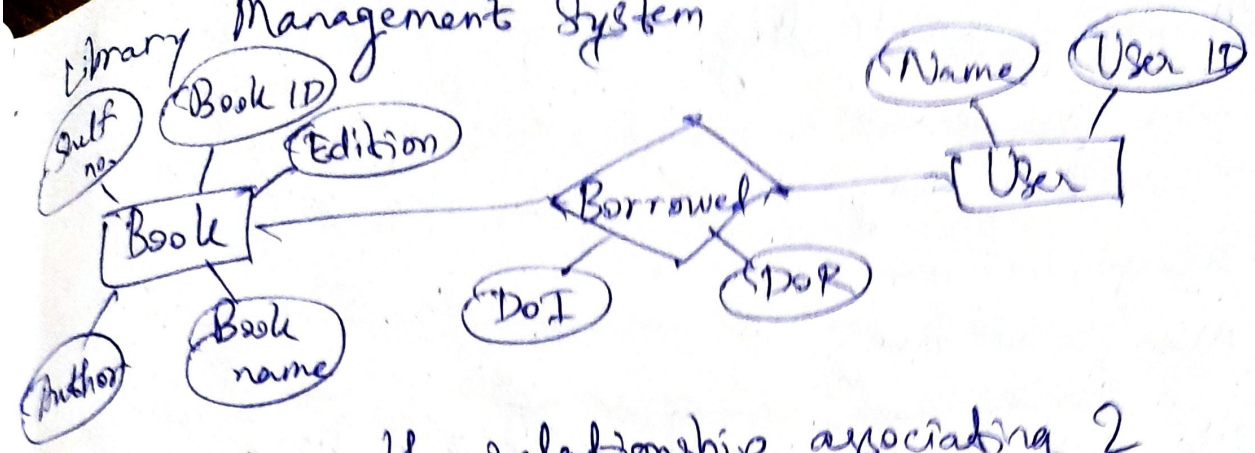
Student → {Reg. No} ← 2 such
{Name, Address} ← candidate Keys

I cannot remove any one attribute from this set, as then it'll lose its property of being a key.
If I remove address, only name is insufficient for uniquely identifying a student, ie., a row, as multiple students can have same name.

Primary key → One such set of attributes from candidate keys can be a primary key.
Usually, primary key is that set of attributes whose size is minimum.
So, here {Reg No.} is preferred.

Super Key → Set of attributes such that it's a key but it contains atleast one or more redundant attributes, upon removal of which, it still remains a key.
Eg → {Reg. No., Name} is a super key as I can remove Name from it but it'll still be a key.

Library Management System



Borrowed is the relationship associating 2 entities Book & User in the ER model.

Now, the primary key of the relationship Borrowed is the combination of primary keys of its associative entity sets.

Book → {Book ID} → {Book ID, User ID}

User → {User ID} →

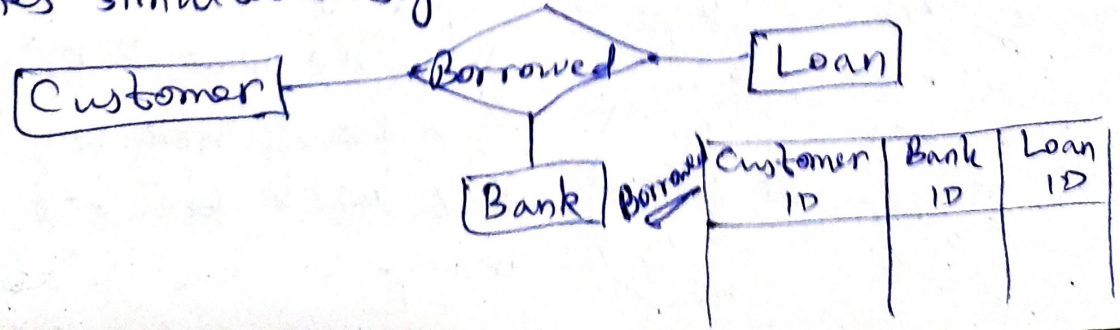
And the relationship itself has its own attributes, Date of Issue (DoI) and Date of Return (DoR).

In the table format, ~~Rela~~ Borrowed is known as a relation, not relationship.

In table format, the 'Borrowed' relationship will have 2 columns, Book ID & User ID i.e. the primary keys of the entity sets it's associated with.

A relationship can be associated with 2 or more entities simultaneously.

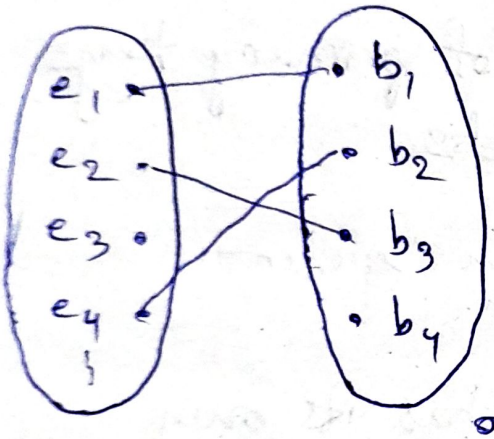
Eg:



Mapping Cardinality

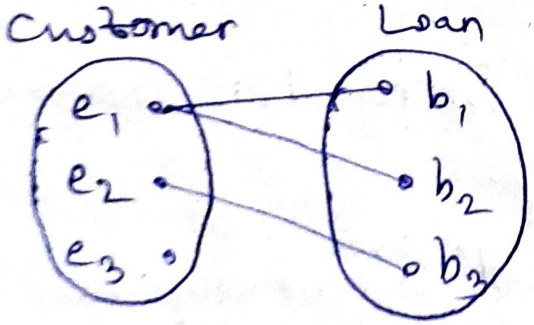
- One-to-one
- One-to-many
- Many-to-one
- Many-to-many

Mapping b/w 2 entity sets, where one or more entities of 1 set are in a relationship with one or more entities of the 2nd set.



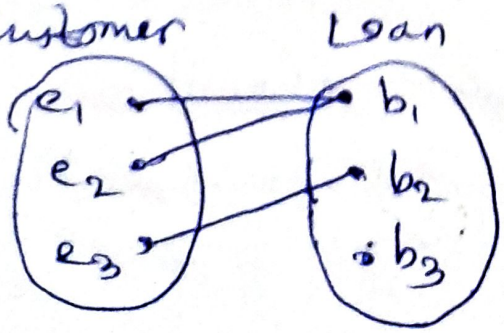
A one-to-one mapping cardinality, it is NOT mandatory that every entity in a set has to have a relationship with atleast one entity in the 2nd set.

One-to-many



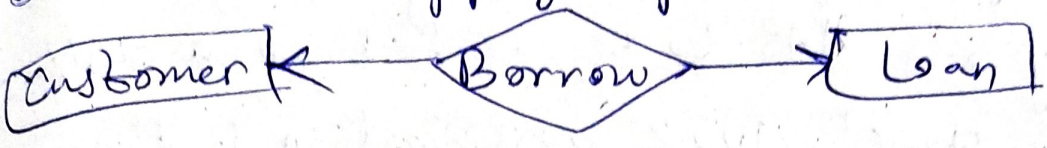
One customer can ~~take~~ several loans, but only he/she has to pay it back, and one loan ID can only be assigned to one person, so ~~many-to-one~~ it's a one-to-many mapping.

Many-to-one



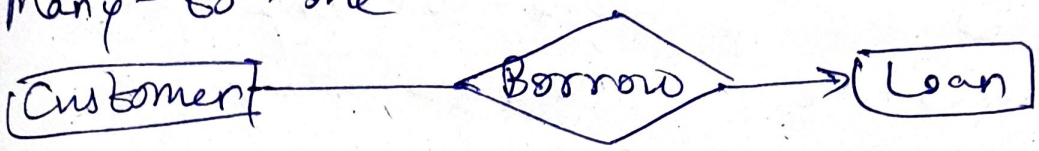
Every person can take only one loan, BUT they can have nominees to pay back the loan, so a many-to-one relationship.

One-to-one mapping depiction :-



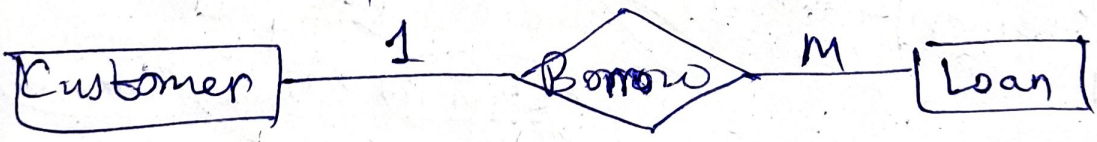
One-to-many :-

Many-to-one

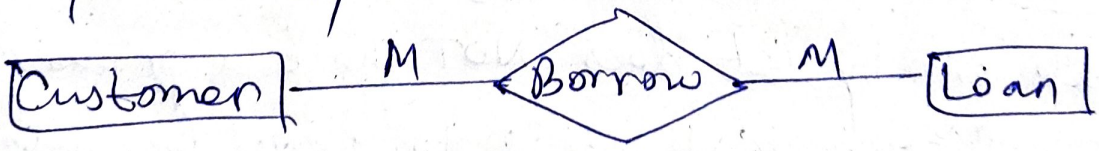


Many-to-many

One-to-many



Many-to-many

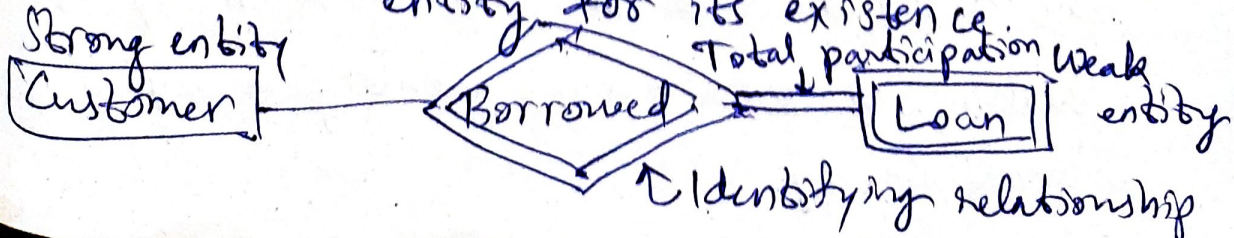


Existential Dependency

Some entities can't exist by themselves, they need to depend on another entity for their existence.

Strong entity → One that can exist on its own.

Weak entity → One that depends on a strong entity for its existence.



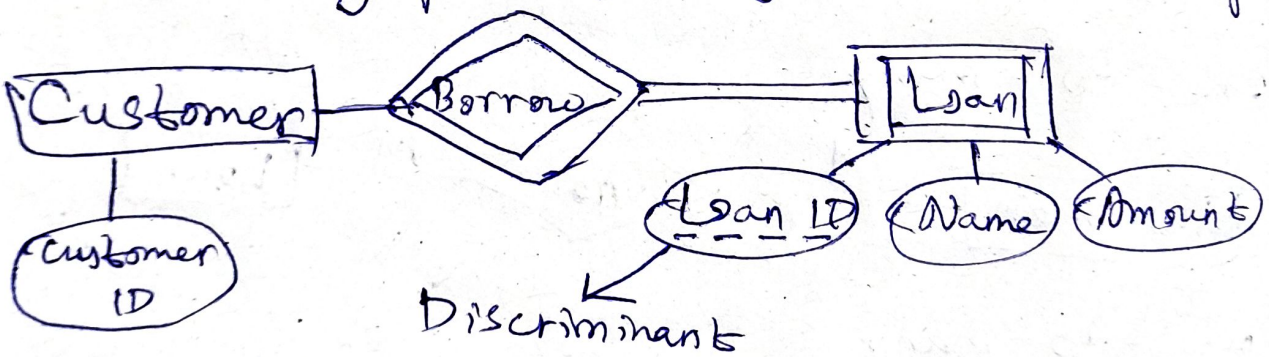
Total participation → Every loan ID in the entity set HAS to be related to a customer ID, so there's 100% participation in the loan entity set to customer entity set, hence the solid line shows total participation.

Cardinality

One-to-one | One-to-Many
Many-to-one | Many-to-Many

Mapping Cardinality →

Shows which members in each entity set is actually participating in the relationship.



Weak entity set does NOT have a primary key of its own, it needs to associate with primary key of its strong entity.

Loan

Customer ID	Loan ID	Name	Amt

While loan ID seems like the primary key of loan, it is not. It's rather called the discriminant of the weak entity as it seems to be a unique identifying key of that entity.

Customer → Strong entity has → identifying relationship
Account → weak entity

Many-to-many mapping :
One customer can have several accounts.
And one acc. can be a joint account, linked to many customers.

DBMS

8/8/23

Constraints :

1) Domain constraint
2) Key constraint
3) Entity integrity constraint
4) Referential integrity constraint

1) Value of an attribute has to be confined within the domain of the attribute.

2) No 2 tuples in a relation i.e. no 2 rows in a table can have the same values for a primary i.e. defining key. Eg: Reg. No. can't be same

Reg. No.	Name
24	Ram
25	Ram

3) The defining key attribute of a tuple can NEVER be a null value.
Primary key can never be a null value.

4)

Emp. Id	Name	Dep. No.	Addres
20	X	3	Allahabad
21	Y	5	Lucknow
22	Z	5	Kanpur

(Primary key underlined)

EMPLOYEE

Dep. No.	Dep. Name
3	Research
5	Admin

DEPARTMENT

The Dep. No. of Employee relation is a foreign key referencing to the Dep. No. of the Department relation.

Referential integrity dictates that the Dep. No. of Employee table HAS to reference to a valid Dep. Name in the Department table.

Foreign key can be NULL, as it references to the primary key in the Department table, but it can NOT be a value that does NOT exist as a primary key in the Department table, and needless to say that the primary key can never be NULL.

Also, foreign key need not always be unique.

Employee table → Referencing relation

Department table → Referenced relation

This is the referential integrity constraint.

A set of attributes FK (foreign key) in relation schema R_1 is a foreign key of R_1 that references R_2 if it satisfies 2 conditions :-

(I) The attributes in FK have the same domain as that of PK of R_2

(II) A value of FK in a tuple t_1 either occurs as a value of PK for some tuple t_2 or is null.

In the former case, we have

$t_1[FK] = t_2[PK]$

$R_1 \rightarrow$ referencing relation

$R_2 \rightarrow$ referenced relation

If (1) and (11) satisfies, we say that referential integrity holds.

$t_1 \rightarrow$ tuple (or row)

Eg:- {20, X, 3, Allahabad} is a tuple

$t_2 \rightarrow$ tuple (or row)

Eg:- {3, Research} is a tuple

1), 2), 3) are defined on single relations.

4) is defined on 2 relations.

<20, A, 3, Kanpur> is invalid insertion, as a row with employee ID 20 (primary key) already exists, So key constraint.

<X, 22, 3, Lucknow> is invalid insertion. Firstly, we insert ordered tuple, hence X ∉ domain of emp-ID, So domain constraint is violated.

Why is a table called a relation ?

Relation is basically subset of Cartesian product of several sets which form many ordered tuples.

Here, imagine cross-product given below,

Emp-ID X Name X Dep.No. X Address

and the relation (or table) hence formed R

$R \subseteq$ Emp-ID $\times$ Name $\times$ Dep. No. $\times$ Address
(set of nos. · set of strings · set of nos. · set of strings)

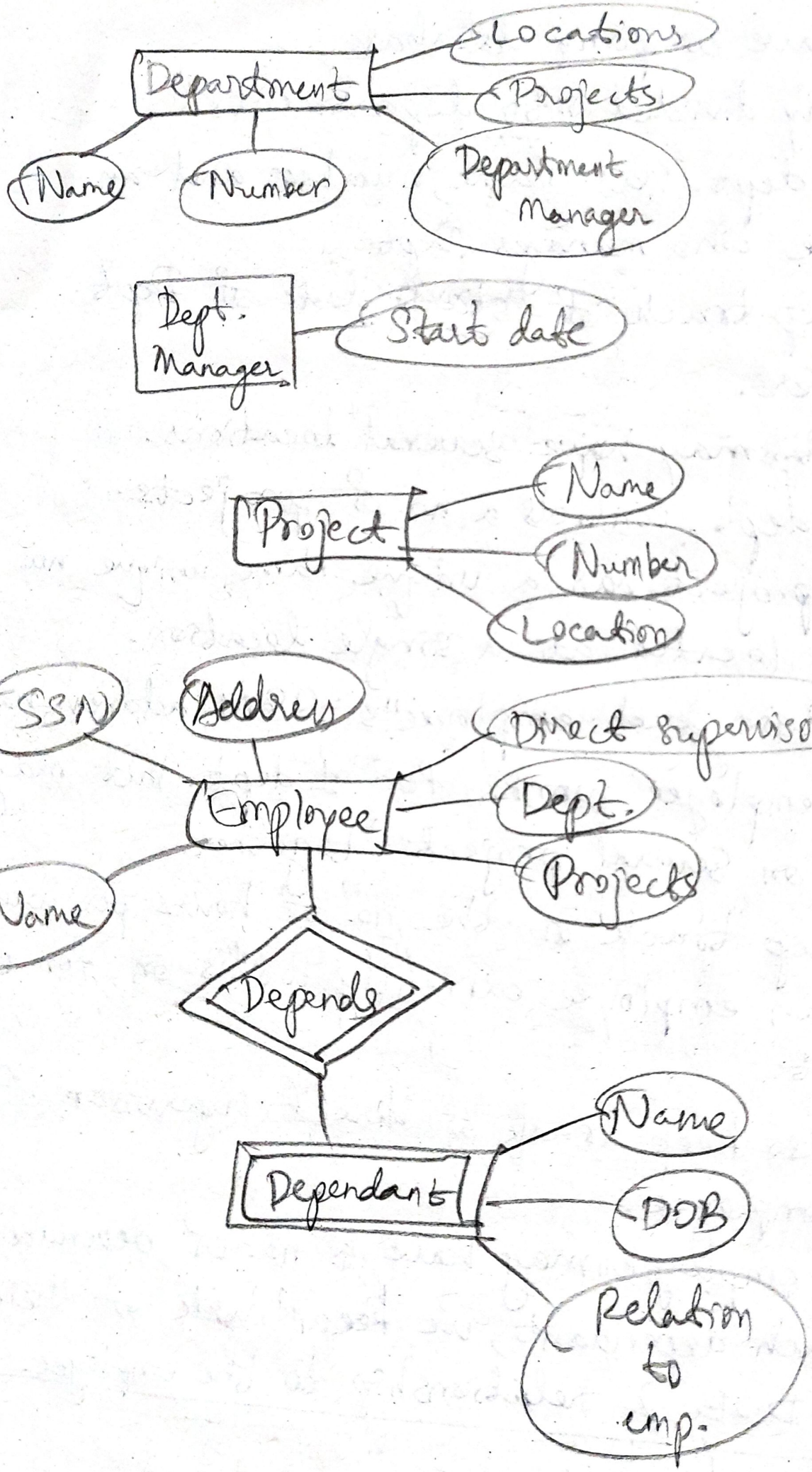
Order of Deletions →

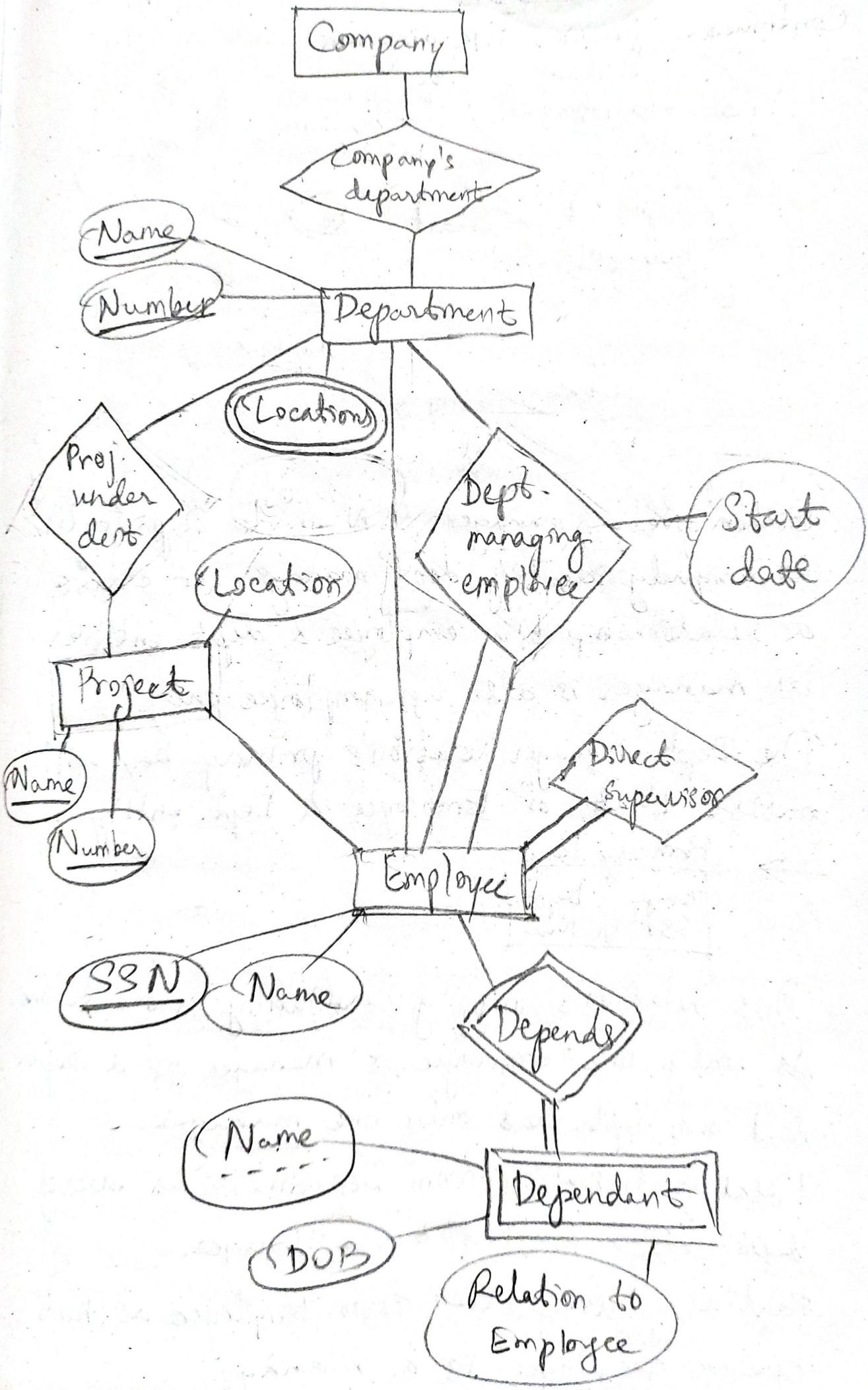
There's a certain sequence of deletion of any tuple in a table i.e. you CANNOT delete a tuple whose primary key is being referenced to by another relation i.e. if a primary key has a dependency it cannot be straightaway deleted until its dependencies are taken care of.

I can't straightaway delete <3, Research> from Department table without deleting <20, X, 3, Allahabad> in Employee table first. But I can delete the latter first, without any worry as it's not referenced to by any other table.

We have company database.
Company divided into departments.
Each dept. has name, number and an employee who manages Dept.
We keep track of start date of Dept. Manager.
A dept. may have several locations.
Each dept. controls a no. of projects.
Each project has a unique name, unique no. and is located at a single location.
We store each employee's SSN, address, name.
Each employee works for 1 dept. but may work on several projects together.
We keep track of the no. of hours per week that an employee currently works on, for each project.
We also keep track of direct supervisor of each employee.
Each employee may have a no. of dependants. For each dependant, we keep track of their name, DOB & relationship to the employee.

Entities → Dept. Manager
Dept.
Company
Projects

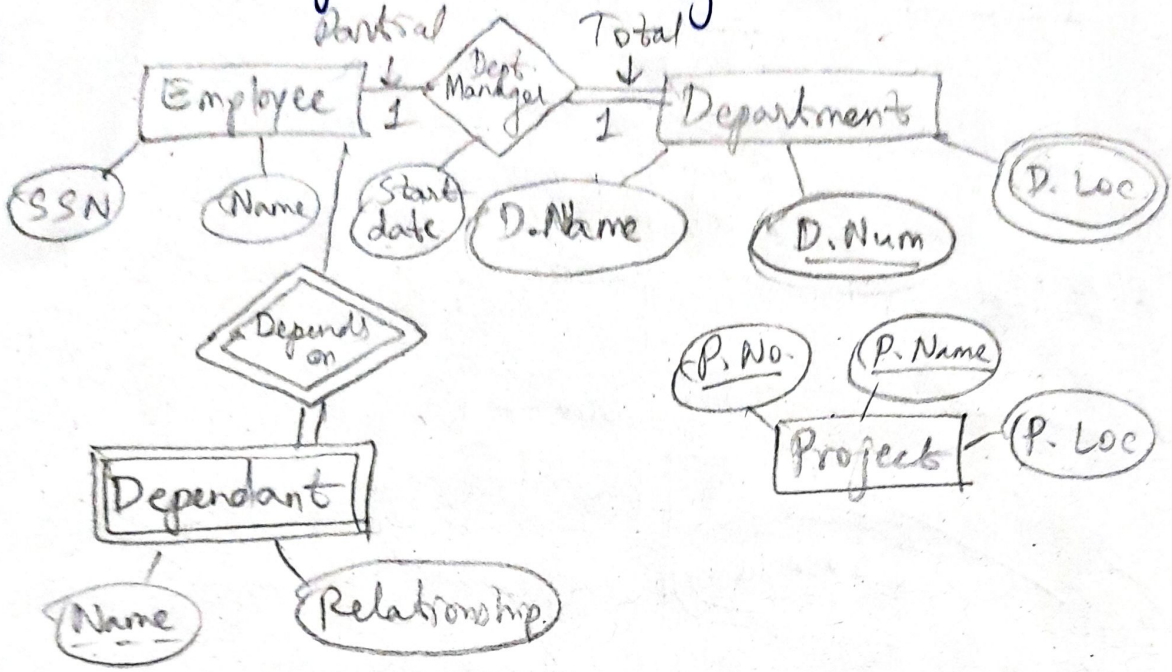

Department
Locations
Projects
Department Manager
Name
Number
Dept. Manager
Start date
Project
Name
Number
Location
SSN
Address
Direct supervisor
Employee
Dept.
Projects
Name
Depends
Name
Dependant
DOB
Relation to emp.


Company
Company's department
Name
Number
Department
Locations
Proj. under dept
Dept. managing employee
Start date
Location
Project
Name
Number
Direct supervisor
Employee
SSN
Name
Depends
Name
Dependant
DOB
Relation to Employee

DBMS

9/8/23

Continued... previous ER diagram



Either give a manager SSN no. to Dept. entity to uniquely identify dept. manager, or create a relationship b/w employee & dept. entities as manager is also an employee only.

The Dept. Manager relation's primary key will consist that of Employee & Dept. entity.

→

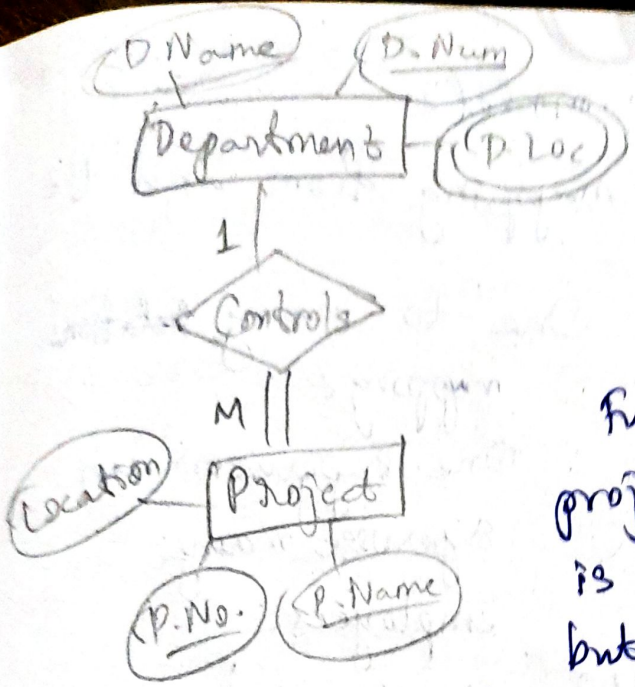
Primary key

Emp-SSN	Dept. Num

This relation's mapping cardinality is one-to-one. As only one employee is manager of a dept and one dept. has only one manager.

Total participation from Department as every dept. HAS TO HAVE a manager.

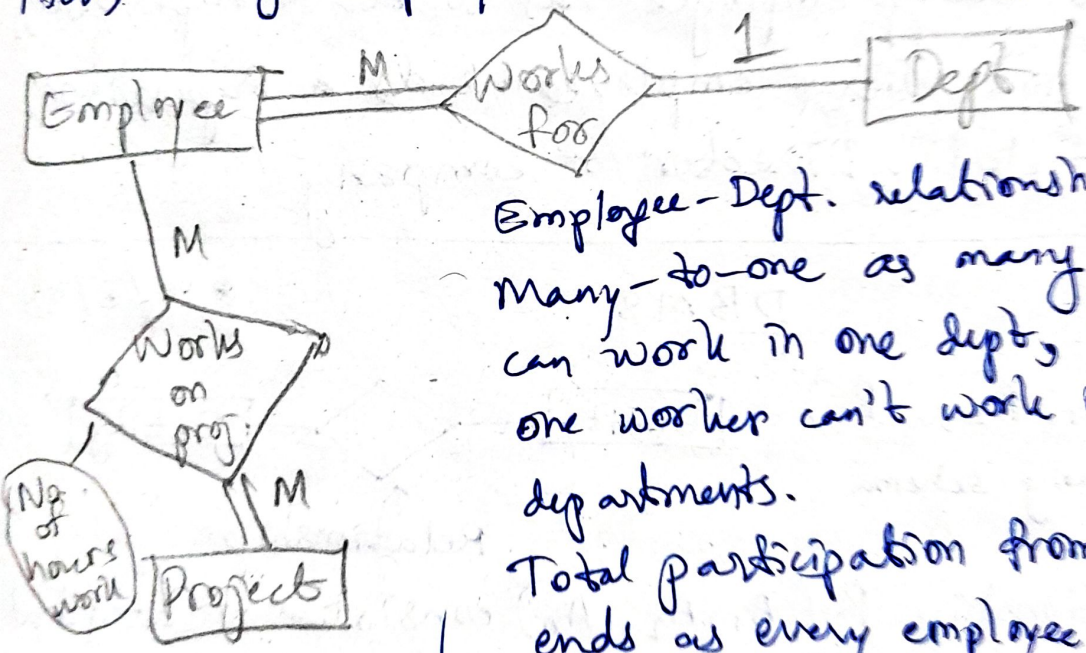
Partial participation from Employee as not every employee is a manager.



Mapping cardinality

One-to-many as one dept. may control many projects. Full participation from project entity as every proj. is under control of a dept. but not vice-versa. A dept may've finished all projects.

Now, every employee works for a dept.

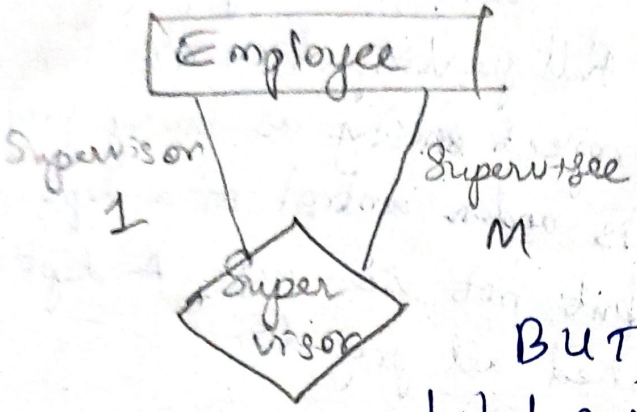


Employee-Dept. relationship → Many-to-one as many employees can work in one dept, but one worker can't work in many departments.
Total participation from both ends as every employee has to be part of a dept and every dept has to have an employee atleast.

Employee-Project → Many-to-many → as many employees can work on a project, and one employee can work on many projects.
Total participation from proj. → Every proj needs to have atleast one employee work in it, but not every employee needs to work on a proj. Eg: HOD

Employee & Supervisor model:

Unary relationship mapping from and to the employee.



One-to-many ~~relations~~ mapping:

One supervisor can supervise many employees.

BUT, neither end has total participation.

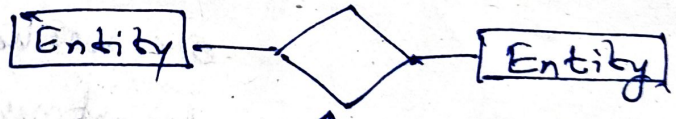
Not every employee has to be a supervisor, and not every employee needs a supervisor, eg: HoD, Director of company

---

DBMS

10/8/23

$R(R_1, R_2, .. R_n)$ Company schema



↑ Relationship

Relationship $R_1(A_1, A_2 .. A_n)$ consists of attributes

Strong entity sets are converted to relations.

Employee

SSN	Name	Addres

Department

D. No.	Dep. Name

Project

P. No.	P. Name	P. Loc.

Dependant

SSN	Name	Relationship

SSN: Primary key of strong entity

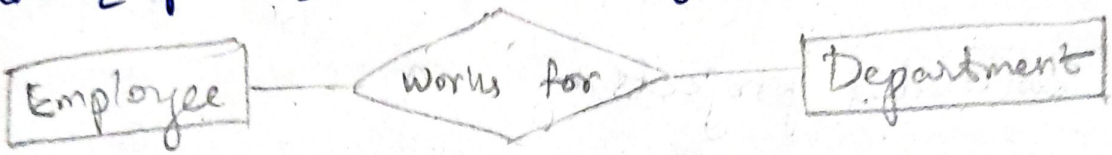
Name: Discriminant

Above 4 entities are shown as relations. Now, a multi-valued attribute will have to be shown as a separate relation like department locations.

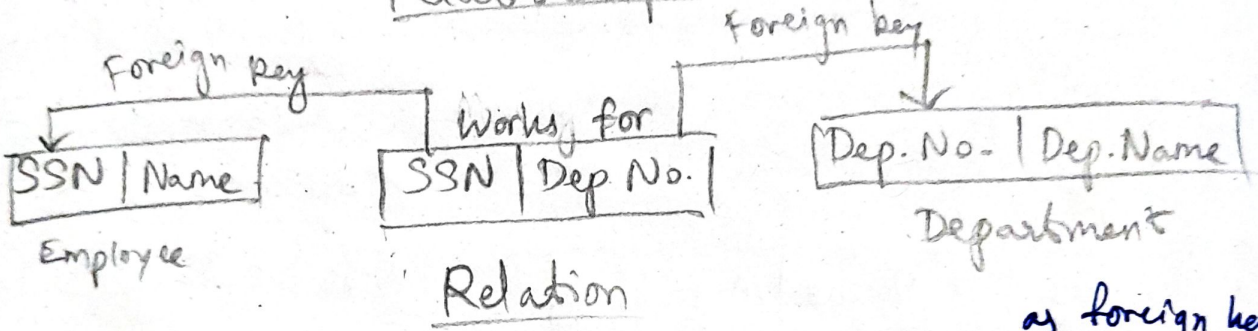
Dept. Locations

D. No.	D. Loc.

In a row, one column can accept only one entry. So, for a multi-valued attribute it'll take multiple columns, especially when you don't know the no. of columns you'll need. So, better you create a separate row for that and {dept no., location} together forms a key.



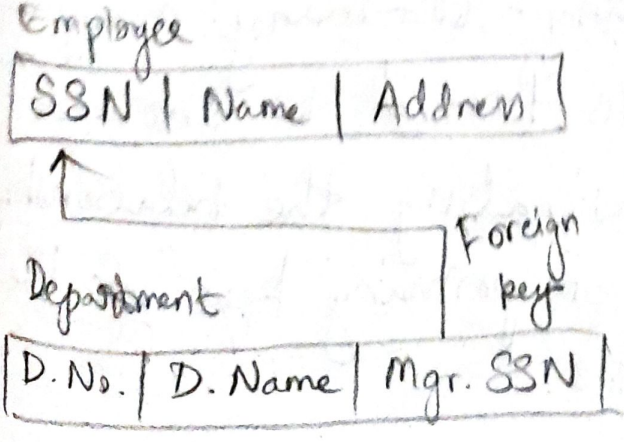
Relationship



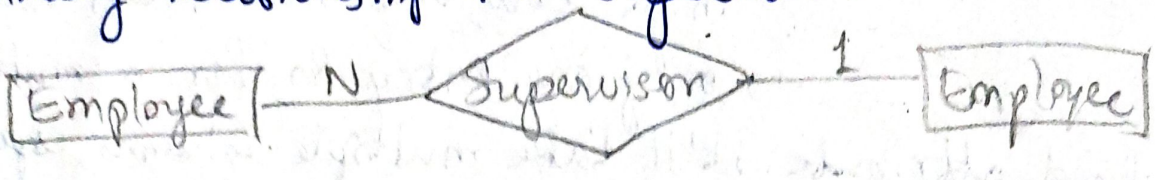
Alternative :- You can add an attribute Dep. No. as foreign key to Employee entity or add SSN of employee as foreign key to Department entity.

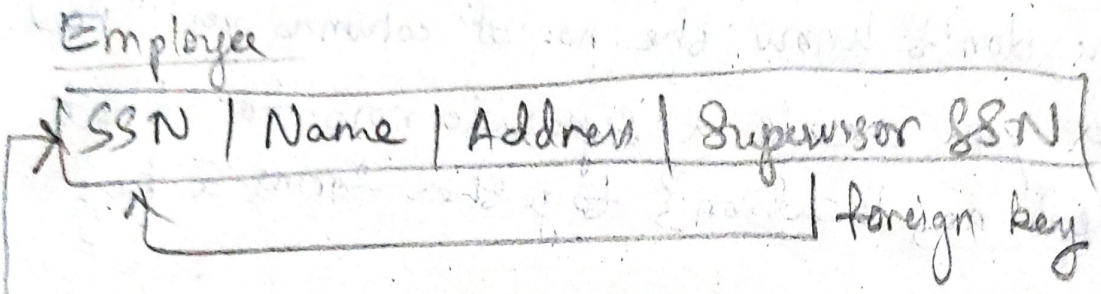
The former is a better idea as one employee works in only 1 department, but one department has many employees working under it.

Here, Dep. No. is foreign key of employee entity that ~~maps to~~ references to Department entity.

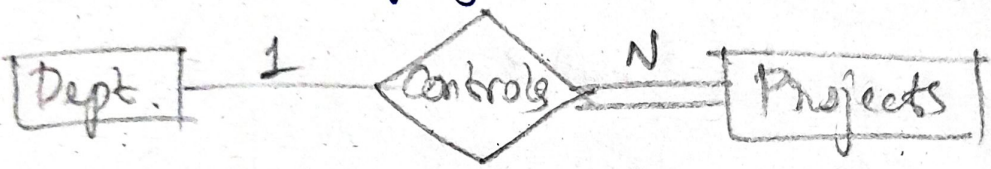


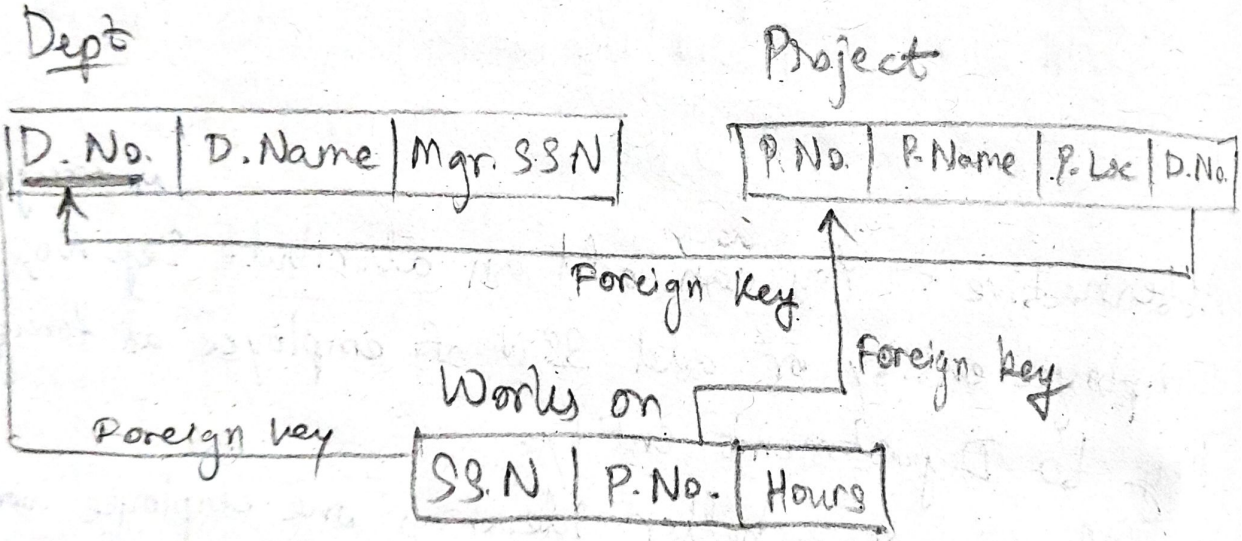
Unary relationship for supervisor :-





Dept. controlled projects :-





It's always desirable to add an attribute (as a foreign key) to a relation, referencing to another relation, whilst illustrating a relationship.

But in case it's a many-to-many mapping cardinality, we may be forced to show a separate relation illustrating the relationship consisting of collection of primary keys of the referencing to the entities.

DBMS

22/8/23

Relational Algebra:

DDL commands → Data Definition Language
also called DML → Data Manipulation Language
SQL Schema → Structure of a database

Company(Employee, Dept., Dept. Loc., Project) } together it's called a schema
Employee(SSN, Name, Gender...) }

At different instances, we can have different no of rows(tuples) in our table.

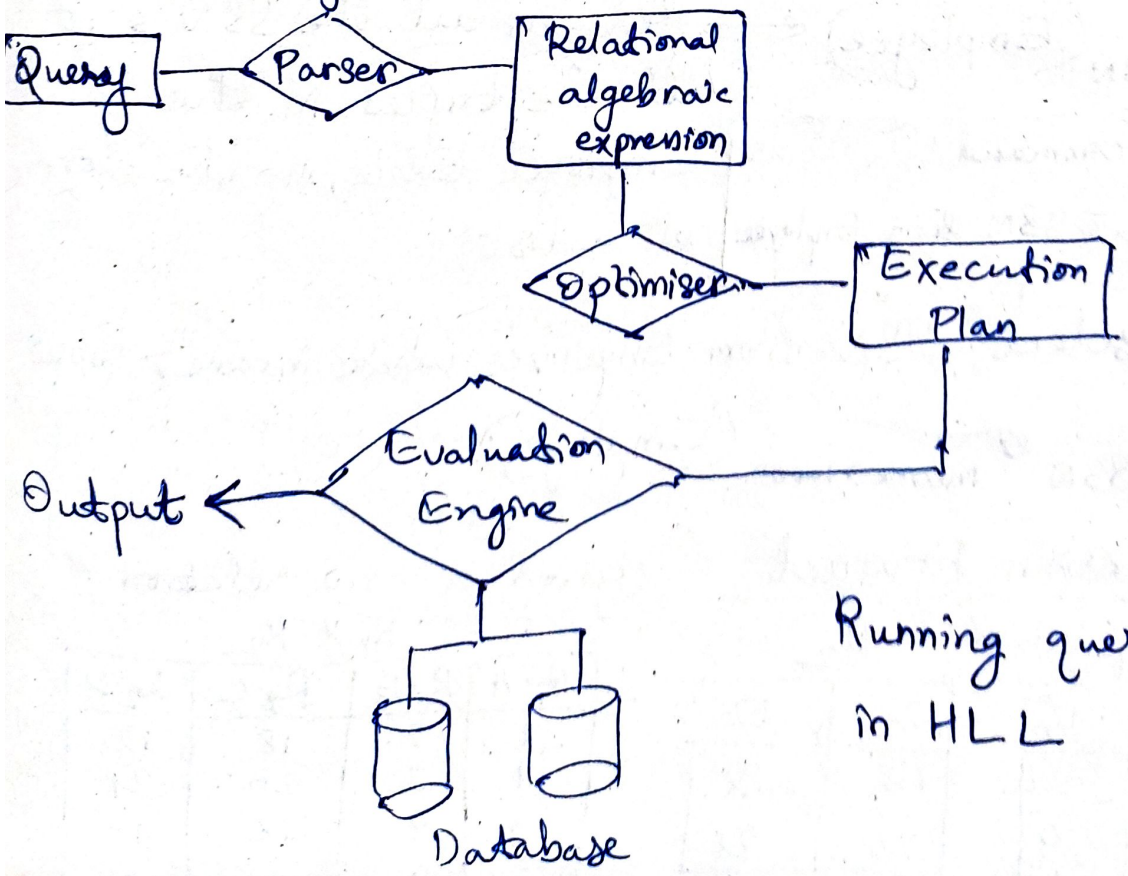
DDL commands → related to constructing the schema of the database

Eg: create, drop & delete table

DML commands → queries to get & insert data, all in all, manipulate the database

Eg: insert & select queries

Relational Algebra → Mathematical equivalent of HLL



Running query in HLL

Fundamental Operations

1. Select } unary operations
2. Project }
3. Cartesian product
4. Rename
5. Set difference
6. Union

Derived operations

1. Natural join
2. Divide
3. Theta join
4. Assignment
5. Intersection

Select Operation ≡ where clause in SQL

(Partitions table horizontally)

- Unary operation ← Accepts only 1 relation

Employee: | SSN | Name | Income |

SQL command: select * from Employee where Income > 10000

(* → all columns selected; from Employee → all tuples from Employee satisfying)

→ Equivalent relational algebraic expression

Sigma → $\sigma_{\langle Income > 10000 \rangle}(Employee)$

Project Operation

(Partitions table vertically)

$\Pi_{SSN}(Employee)$ ← Returns all the SSN's of all the tuples in the Employee table, in the form of a list

SQL command:
select SSN from Employee

→ select SSN from Employee where income > 10000

$\Pi_{SSN}\,\sigma_{Income > 10000}(Employee)$

Cartesian Product → gives a new relation

R_1

A	B
1	2
3	4

R_2

C	D
18	19
20	21

$R_1 \times R_2$

$R_1.A$	$R_1.B$	$R_2.C$	$R_2.D$
1	2	18	19
1	2	20	21
3	4	18	19
3	4	20	21

We are NOT taking cartesian product of one tuple with one attribute.
It's simply product of one tuple with that of the 2nd relation.

Book(AccNo, Year, Title)
User(CardNo, Name, Address)
Supplier(SName, SAddress)
Borrow(AccNo, CardNo, DoB)

Find the Acc. No. of all books issued by Rakesh Agrawal

select AccNo from Borrow where cardNo = that of Rakesh

We'll need the cartesian product (User x Borrow)

User

Card No	Name
12	Rakesh
13	Ram
14	Shyam

Borrow

Acc. No.	Card No.	DoB
A101	12	-
A201	12	-
A301	13	-

select AccNo from Borrow b join User u on b.CardNo=u.CardNo where u.Name = 'Rakesh'

$\Pi_{\text{Borrow. AccNO}}\ \sigma_{\text{u.cardNo = b.CardNo} \wedge \text{u.Name = 'Rakesh'}}\ (b \times u)$

User x Borrow

User Card No	User Name	Acc. No.	Card No.
12	Rakesh	A101	12
12	Rakesh	A201	12
12	Rakesh	A301	13

Name = Rakesh AND User Card No = Borrow Card No.

⋮
and so on.

In SQL, basically join the tables and then do the select operation.

Relational algebraic expression :-

$\Pi_{\text{Acc. No.}} \left(\sigma_{\text{Name = 'Rakesh'}} (\text{User} \times \text{Borrow}) \right)$

$\sigma_{\text{User Card No.} = \text{Borrow Card No.}}$

DBMS 23/8/23

Relational Algebra & Tuple Calculus

↳ Mathematical formulation of commands given in Structured Query Language (SQL)

Cartesian Product involves more than 1 relation, not a unary operation, rather a binary one.

Union : $R_1 \cup R_2$, not $R_1 \times R_2$ SQL command: union

In order for $R_1 \cup R_2$ to happen, no. of attributes of relation R_1 = no. of attributes of relation R_2 AND corresponding domains of attributes have to be the same.

Degree of a relation = no. of attributes of the relation

Book(Acc No, Yr of Pub, Title)

User(Card No, Name, Address)

Supplier (SName, Address)

Borrow (Acc No, Card No, DoI) ← date of issue

Supplied By (Acc No, SName, Price, DoS) ← date of supply

Q. Find all the books which are either issued or have been supplied by a supplier.

All books issued → given by relation Borrow

All books supplied → given by Supplied By

Borrow ∪ SuppliedBy can't be done directly, so we just need to take the AccNo attribute, which is what we need. ← done using project operation.

$\Pi_{AccNo}(Borrow) \cup \Pi_{AccNo}(SuppliedBy)$

Q. Find AccNo of all books available in library.

All books whose AccNo is NOT in Borrow.

So, Set difference operation.

Book − Borrow, take the equal attributes only.

$\Pi_{AccNo}(Book) - \Pi_{AccNo}(Borrow)$ ← on the basis of this

Cartesian Product of 2 tables on the basis of some condition ← SQL Query

Select Book.AccNo from Book, Borrow where Book.AccNo = Borrow.AccNo

↑ Gives list of AccNo of all borrowed books

Name of users who've borrowed books

$\Pi_{Name}(\sigma_{User.CardNo = Borrow.CardNo}(User \times Borrow))$ ← Cartesian product done internally

Query ← Select User.Name from (User, Borrow) where User.CardNo = Borrow.CardNo

For SQL query practice → Korth, Evan Bayross

Rename: $\rho_{newrelation}(oldrelation)$

Eg ← $\rho_{BookNew}(Book)$

Renaming attributes as well ← new names of attributes, old name of relation

$\rho_{BookNew}(AccNoNew, YrofPubNew, TitleNew)(Book)$

BookNew ← new name of relation

5/9/23

SQL Operators

Distinct keyword → Takes only unique values under column/s specified, rejects repetitions.

count(...) → Displays no. of concerned elements from column/s specified

State	City
X	A
X	B
X	C
Y	D
Y	E

Select distinct state from table

X
Y

Select count(distinct state) from table
↳ 2

Select distinct city, state from table

AX
BX
CX
DY
EY

Select count(distinct city, state) from table
↳ 5

Wildcard → '_' → one character
'%' → zero/multiple characters

like operator → for string matching

Comparison operators don't work on null values.
So, there you have to use IS NULL, IS NOT NULL

DBMS

5/9/23

Simple Cartesian product is seldom used because we associate it with several conditions.

Natural Join → It's a combination of project, select and cartesian product (binary operation).

Say, a natural join of tables R_1 and R_2 would be $R_1 \bowtie R_2$.

Eg:

R_1

A	B
1	2
3	4
5	6

R_2

B	C
2	22
4	82

$\pi_{R_1.A, R_2.C}\ \sigma_{\langle R_1.B = R_2.B \rangle}\ (R_1 \times R_2) \equiv R_1 \bowtie R_2$

$R_1 \bowtie R_2$ ← this meaning is by default implemented by $\bowtie$, on the basis of common columns

A	B	C
1	2	22
3	4	82

R_1

A	B	C
1	2	22
3	4	86
5	6	34

R_2

B	C
2	22
4	82

As 2 columns are common, instead of similarity on the basis of element in 1 column, now it will be on the basis of an ordered tuple of 2 intersecting (common) columns.

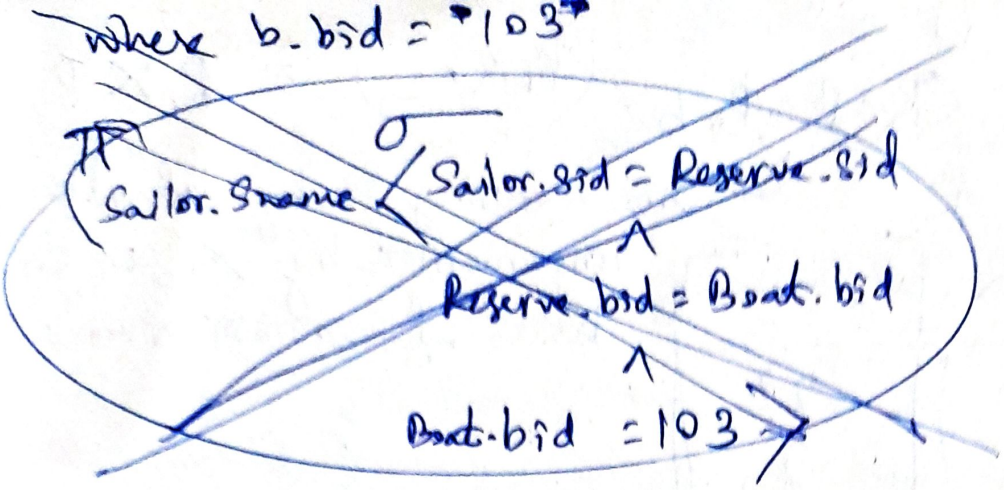
$R_1 \bowtie R_2$

A	B	C
1	2	22

Sailor(<u>Sid</u>, Sname, Age, Rating)
Reserve(<u>Sid, bid</u>, day)
Boat(<u>bid</u>, bname, color)

Q1. Find names of sailors who've reserved boat "103".

Select s.Sname from Sailor s inner join Reserve r on s.Sid = r.Sid inner join Boat b on r.bid = b.bid



$\pi_{Sailor.Sname}\ \sigma_{<Reserve.bid=103>}\ (Sailor \bowtie Reserve)$

$\pi_{Sname}\ \sigma_{<Sailor.sid=Reserve.sid\ \wedge\ Reserve.bid=103>}\ (Sailor \times Reserve)$

~~π_{Sname}~~ $\pi_{Sname}\ (\sigma_{<bid=103>}\ (Reserve) \bowtie Reserve)$

(Sailor ⋈ Reserve) is going to give a much larger table on which you'll perform conditional select.

But this is better as this truncates the Reserve table so natural join product gives a smaller table, hence taking lesser time.

Q2. Find Sailor names who reserved a red boat.

Select S.Sname from Sailor s inner join Reserve r on S.Sid = r.sid inner join Boat b on r.bid = b.bid where b.color = 'Red'

Finding $\sigma_{color='Red'}(Boat)$ will be better to optimise

$\pi_{Sname}(((\sigma_{color='Red'}(Boat)) \bowtie Reserve) \bowtie Sailor)$

```
select s.Sname from Sailor s
inner join Reserve r on s.Sid = r.Sid
inner join Boat b on r.bid = b.bid
where b.color = 'Red'
```

DBMS 6/9/23

Q3. Find names of sailors who've reserved atleast one boat.

```
select distinct s.Sname from Sailors s inner join
Reserve r on s.Sid = r.Sid
```

$\pi_{distinct\ Sname}(Sailor \bowtie Reserve)$ (bid)

$\pi_{Sname}(Sailor) - \pi_{s.Sname}(\sigma_{<s.Sid = r.Sid,\ r.bid = NULL>}(Sailor\ s \times Reserve\ r))$

$\pi_{Sname}(Sailor) - \pi_{s.Sname}(\sigma_{<r.bid = NULL>}(Sailor(s) \times Reserve(r)))$

Q4. Find sailor names who've reserved a 'red' boat or green boat

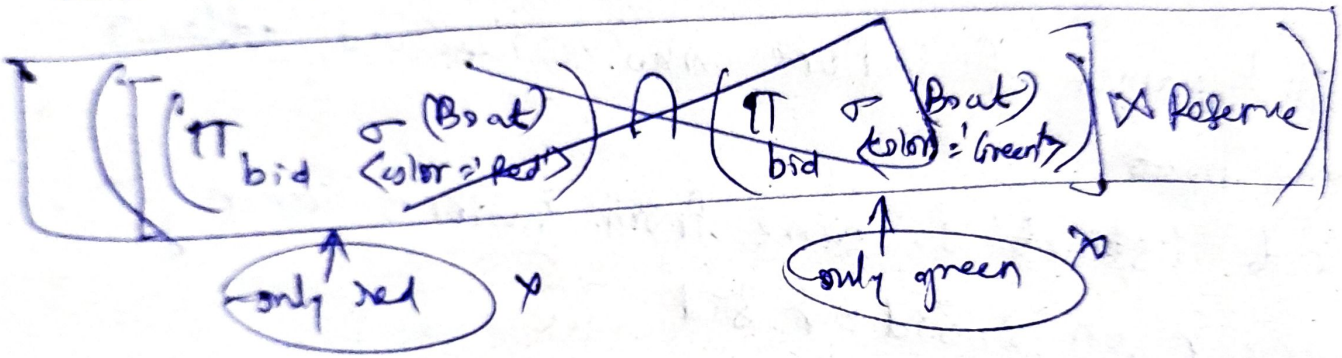
$\pi_{Sname}([(\sigma_{<color='Red' \vee color='Green'>}(Boat)) \bowtie Reserve) \bowtie Sailor]$

~~$(Boat) \cup (color='Red')$~~ Columns are unionised

$\pi_{sname}\Big(\big(\pi_{bid}(\sigma_{color='Red'}(Boat)) \cup \pi_{bid}(\sigma_{color='Green'}(Boat))\big) \bowtie Reserve\Big) \bowtie Sailor$

Generally union is preferred over joined boolean conditions

Q5. Find sailor names who've reserved a Red boat and a Green boat.




```
select distinct s.sname from Sailor s
  inner join (select r.sid from Reserve r inner
    join Boats b on r.bid = b.bid where
      b.color = 'Red' or b.color = 'Green') x
  on s.sid = x.sid
```

sid → bid1 → Green
sid → bid2 → Red
sid → bid3 → Red

$\pi_{sname}\Big(\big(\pi_{sid}(\sigma_{color='Red'}(Boat) \bowtie Reserve) \cap (\sigma_{color='Green'}(Boat) \bowtie Reserve)\big) \bowtie Sailor\Big)$