INTRODUCTION

Basic Definitions -

1.) **Data** - Known facts that can be recorded and have some meaning.

2.) **Database (DB)** - Collection of related data which is designed and stored for a specific purpose.

3.) **Miniworld / Universe of Discourse (UoD)** - Some part of the real world about which data is stored in a database.

4.) **Database Management System (DBMS)** - It is a software which is used for defining, constructing and manipulating databases for various applications.

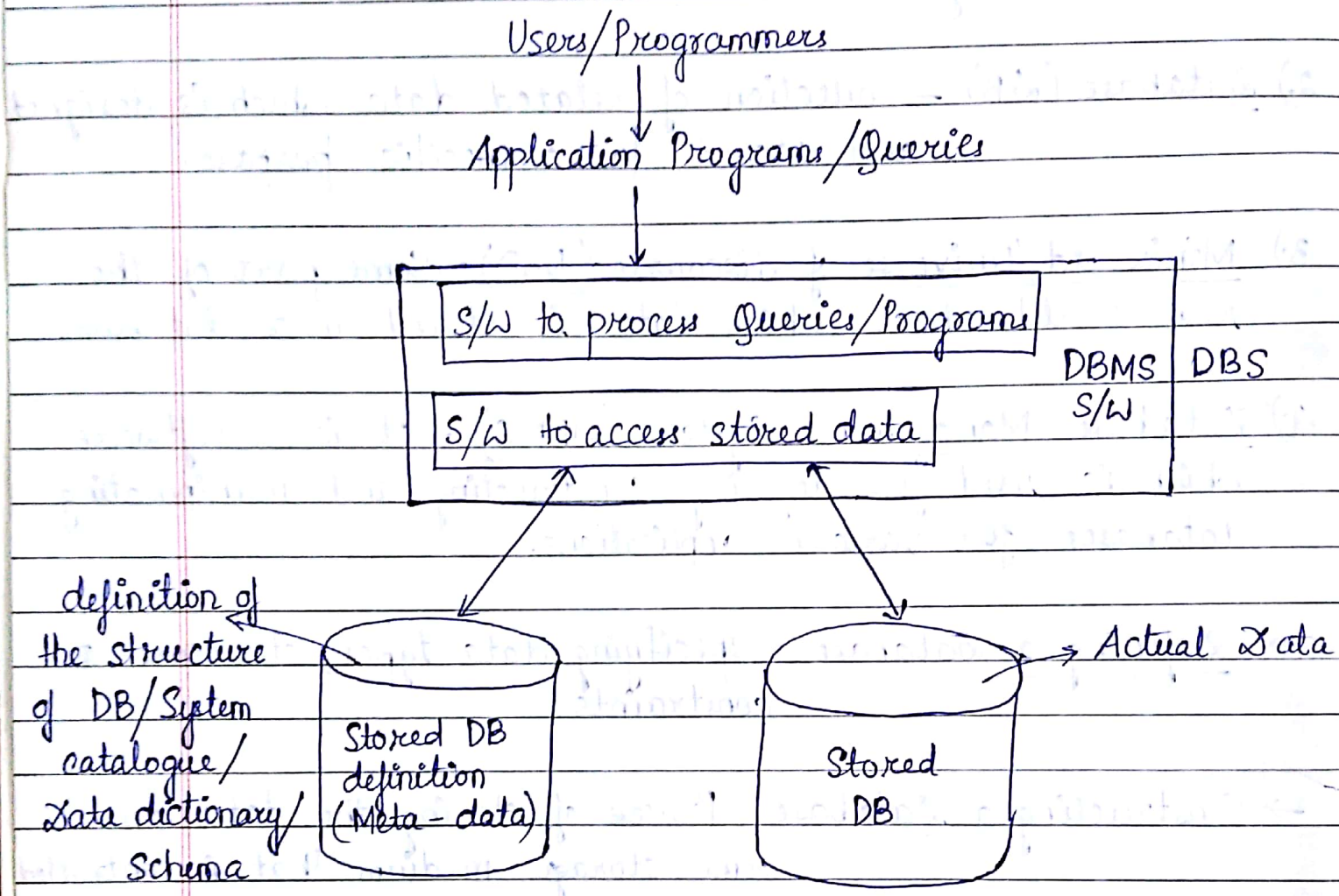
→ **Defining a Database** - specifying data types, structures & constraints

→ **Constructing a Database** - Process of storing the data on some storage medium that is controlled by DBMS.

→ **Manipulating a Database** - Retrieval
Modification : Insertion
Deletion
Updation

DATABASE SYSTEM (DBS) -

Simplified Database Environment -



★ Except users/programmers, everything is called DBS.

Database + Database Management System software are together called Database System.

DBMS software is a collection of programs to create & maintain a database.

System Catalogue / Data Dictionary / Metadata - Data about the database used for describing the structure of data base (DB schema).

Schema contains the name of tables; name of attributes, datatypes of attributes.

Database can be of any length & complexities.

Characteristics of a Database Approach

→ Self-describing nature of database
→ Insulation b/w programs & data and data abstraction.
→ Support of multiple views of the data
→ Sharing of data and multi-user transaction processing.

Eg. of a simplified meta-data -

Relations -

Rel_Name	No. of columns
Student	4
Course	4
Section	5
Grade	3
Pre-requisite	2

Columns -

id_name	datatype	belongs to_relation
S_name	char (50)	student
⋮	⋮	
⋮	⋮	
prerequisite_ID	char (30)	Pre-requisite

★ Insulation b/w programs and data & data abstraction -

Program-Data Independence - Allowing change in data storage structures and operations without having to change the DBMS access programs.

Program Operation Independence - Application programs/queries can operate on data stored on databases through their names & arguements regardless of how the operations are implemented.

Data Abstraction provides a conceptual view of the DB to the user that does not include the storage details of the data & how the operations are implemented in the database.

★ Support of multiple views of the data -

virtual table → view

★ Categories of Database Users

★ Advantages of using the DBMS approach

Ch-2

DATABASE SYSTEM CONCEPTS & ARCHITECTURE

- A data model is a collection of concepts that can be used to describe the structure of a database & the constraints that the database should obey.

- Structure of a database : data types, relationships, constraints the data should hold

- Data Model Operations are of two types -

(a) Basic Data Model operations : Generic operations like insert, select, update, delete

(b) User-defined Data Model operations : Database designer is allowed to specify a set of valid user-defined operations that are applied on the database objects.

Categories of Data Models —

They are categorized based on the types of concepts a model use to describe the structure of a database.

- High-Level / Conceptual Data Model — It uses concepts like entities, attributes, relationships. It is for common users.

- Low-Level / Physical Data Model — It describes the details of how data is stored in the database. It is meant for computer specialists / designers, record format (information of a particular entity stored in a database), record orderings, access paths (specifies the way to access a particular data of an entity in a faster way).

- Representational / Implementational Data Models : It provides concepts that balances user views with some storage details. This is most frequently used in commercial DBMS. There are diff. types of this data model -
  - → Relational Data Model
  - → Network Data Model } outdated
  - → Hierarchial Data Model }
  - → Object Data Models

BASIC DEFINITIONS :

→ Database Schema - describes the structure of a database. is / CDS

For eg- University Database

Student (Name, Roll_No, Course, Major)
Course (Course_No, Course_name, Credit_Hrs, Dept)
Prerequisite (Course_No, Pre_No)
Sections (Sec_ID, Course_No, Sem, Year, Instructor)
Grade_Report (Roll_No, Secti_ID, Grade)

→ Schema Diagram - A diagrammatic display of the database schema is called a schema diagram.

Student

Name	Roll_No	Course	Major

→ Schema Construct - A component of the schema is called schema construct. Eg - Student → Course, etc are schema construct of the university database schema.

→ Database Instance, State, Snapshot -

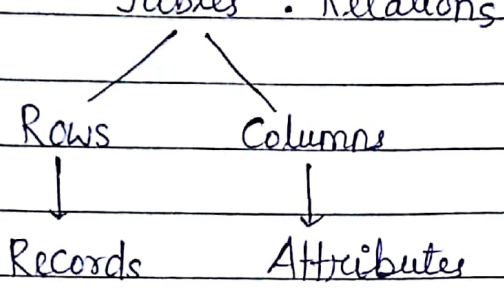
Difference between Database Schema & Database State

DB Schema	DB State
1.) Schema refers to the basic structure.	1.) DB refers to the data currently loaded to the database.
2.) Schema changes infrequently/ rarely.	2.) Changes everytime the data is manipulated.
3.) Called as intension.	3.) Called as extension.

Schema Evolution - DB schema is changed once in a while as the application requirement changes.

DATA MODELS

- **Relational Data Model** - Proposed in 1970 by E.F. Codd
  - First used commercially in 1981-82
  - Data storage & processing



- Single row → Tuple
- Finite set of tuples → Relation Instance

Relation Schema : Relation name & its attributes
Relation Key : combination of one or more attributes that help in identifying the row in the relation uniquely.

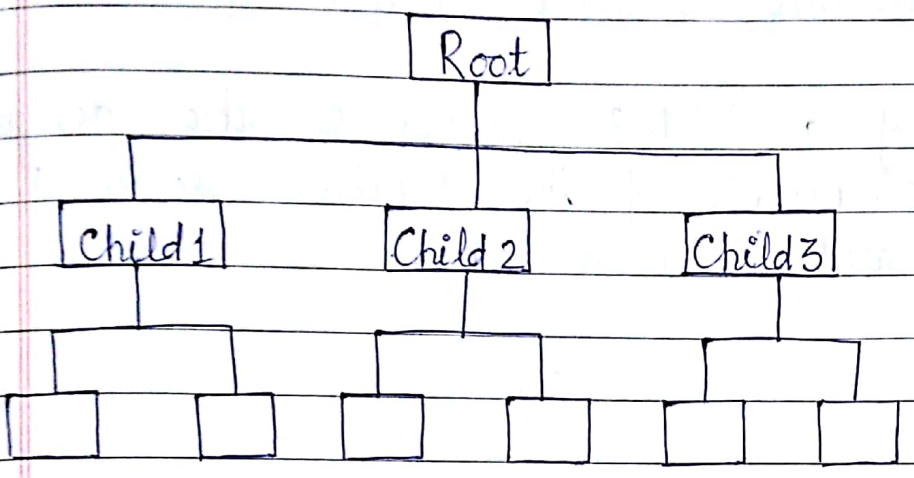
Banking Schema -
Loan (Loan_No, Payment_No, Payment_Date, Cust_id, Payment_Amt)

Here, relation key = Loan_No + Payment_No

Attribute Domain : range of values which the attribute is allowed to take.

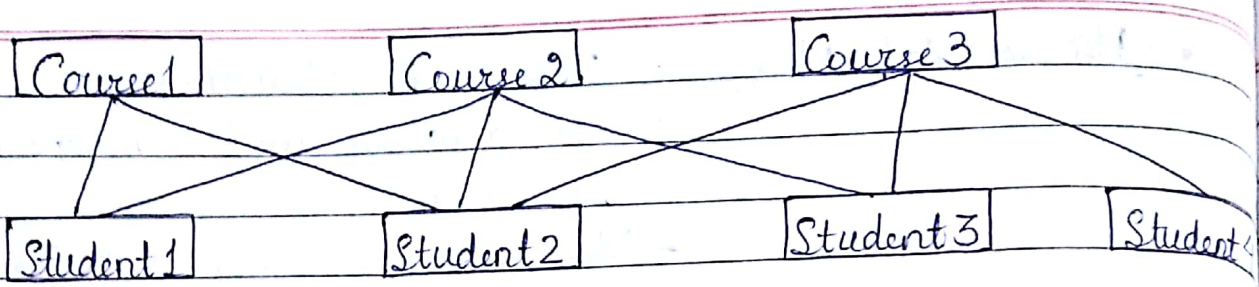
Relational Integrity Constraint :
Each relation has some constraints or relations that must hold. These conditions are called relational integrity constraint.

- **Hierarchial Model** – Tree-like structure
  - 1st database model proposed by IBM in 1960s.
  - Model: one to many relationship (parent-child relationship)



- Data is stored as records connected through links.
- A record here corresponds to a row/tuple in relational model and entity type corresponds to tuple/relation.
- ★ This model restricts that one parent record can have more than one children records but a child record can have only one parent.
- In order to retrieve a particular info, the whole level of tree has to be searched.

- **Network Data Model** – Developed in 1971 during the Conference on Data System Languages (CODASYL)
  - Depicts many-to-many relationships
  - Eg – University Schema: Student-Course relationship.

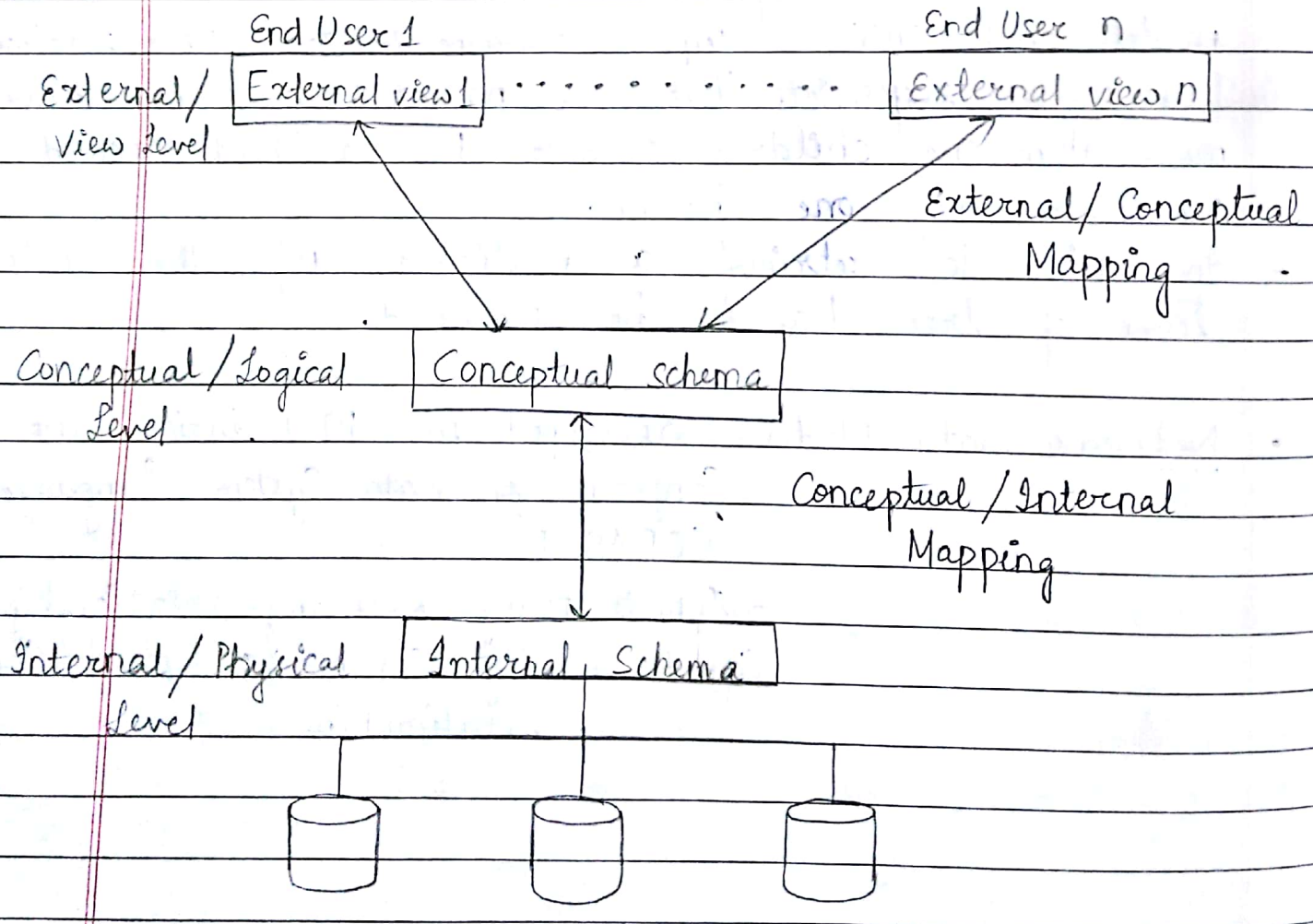


DBMS Architecture & Data Independence

- Architecture of a DBMS refers to the design of the major components of the database system & how they are related to each other.

- **The Three-Schema Architecture**

⇒ Characteristics • Data Independence
• Support of multiple user views



* Logical level means entities, attributes & their relationships.

- Internal Level – It has an internal schema that describes the physical storage structure of the database on the storage medium.

This particular model describes the physical data model.

- Conceptual Level – It has a conceptual schema that describes the logical structure of the schema DB. (It hides the physical / storage details). It uses a high level / implementational data model.

* If N diff. users give N diff. queries, so accordingly N diff. responses are given back to the users. N different views (virtual table) are given as a result.

Mapping :-

There are two levels of transformations of requests & response when a user requests data by writing database query, the user's request must be converted from one of the external views to the conceptual view of the database (external / conceptual mapping).

The process of finding the actual record in physical storage that constitute a logical record in the conceptual schema is called conceptual / internal mapping.

This process of expressing and transforming user request and result at one level in terms of another level in DBMS architecture is called mapping.

Data Independence - It is the ability of changing the schema definition at one level of DBS without having to change the schema definition at other levels or next higher level.

There are two levels of data independence -

1) Logical Data Independence - Ability to change the conceptual schema without having to change the external schema is called logical data independence.

Eg - adding a new data item, removing a data item, change of constraint supply changes the logical data independence.

2) Physical Data Independence - Ability to change the internal/physical schema without having to change the conceptual schema is called physical data independence. Eg - change in storage structure.

★ Achieving logical data independence is more difficult than physical data independence because application programs are dependent on the logical structure of the database.

The physical schema can be usually changed without affecting application programs.

Example - Online Book Database

(Primary Key)

→ ISBN (char) - 15
→ Book_Title (char) - 50
→ Category (char) - 15
→ Price (number) - 4
→ Page_Count (number) - 4
→ Year (number) - 4
→ Copyright_Date (number) - 4

At the internal level –

Data item name	Starting Position in Record	Length in bytes
ISBN	1	15
Book_Title	16	50
Category	66	15
Price	81	4
Page_Count	85	4
Year	89	4
Copyright_Date	93	4

Length of a record for a single book = 96 bytes

At the conceptual level –

→ ISBN ⇒ char(15)
→ Book_Title ⇒ char(50)
→ Category ⇒ char(15)
→ Price ⇒ number(4)
→ Page_Count ⇒ number(4)
→ Year ⇒ number(4)
→ Copyright_Date ⇒ number(4)

At the external level –

View 1

Book-Title
Category
Price

View 2

ISBN
Book_Title
Page_Count
Year

DDL & DML → in Lab

Entity - Relationship Model

- Based on perception of real world scenario that consists of a collection of objects called entities, their attributes and relationships among these entities.

- **Entity** - It is an object in the real world that is distinguishable from other objects. Eg - Person, company, etc. (by their attributes)
- **Entity Set** - It is a set of entities of the same type that share the same attributes. Eg - set of all customers of a bank, set of all students of a branch.
- **Attributes** - They are descriptive properties possessed by each member of an entity set. Eg - for a customer, his attributes can be cust_id, cust_name, cust_street, cust_city, cust_contactno
- **Relationship** - It is an association among several entities usually taken from different entity sets.
- **Relationship Set** - A set of similar relationships can be grouped under a relationship set. We can mathematically define it as -

no. of entities ← $n \geq 2$ entities, each taken from entity sets

If $E_1, E_2, \cdots E_n$ are entity sets, then a relationship set R is a subset of -

$$R \subseteq \{(e1, e2, \cdots en) \mid e1 \in E_1, e2 \in E_2 \cdots en \in E_n\}$$

where $(e1, e2, e3, \cdots e_n)$ is a relationship.

e - element of entity set
E - entity set

If $E_1 \rightarrow$ customer & $E_2 \rightarrow$ loan, then (customer1, loan1) is a relationship set

Descriptive Attributes

Borrower

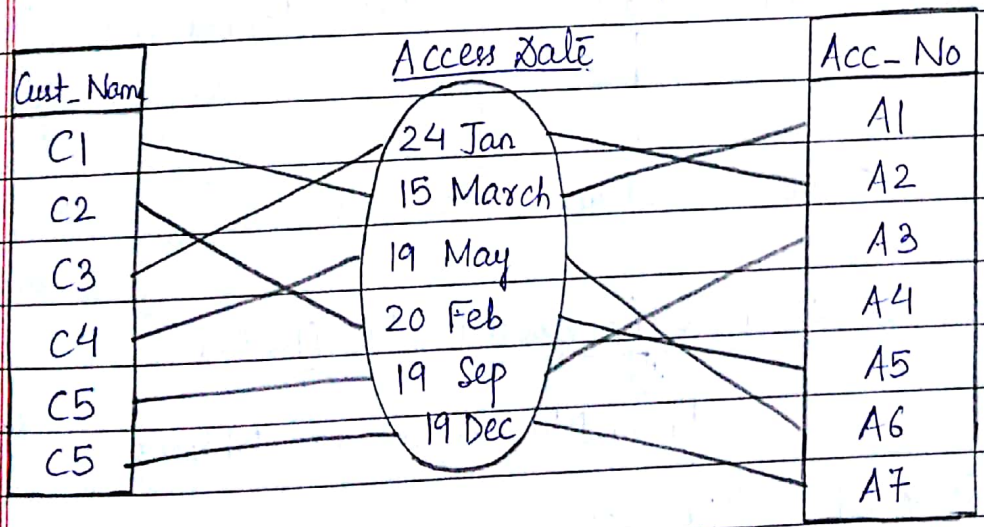
Cust_Id	Cust_Name	Cust_Street	Cust_City
321	A	S1	C1
119	B	S2	C2
227	C	S3	C3
106	D	S4	C4
150	E	S5	C5.
291	F	S6	C6

Customer

Loan_No	Loan_Amt
L-6	10k
L-7	20k
L-8	30k
L-9	5k
L-4	17k
L-3	50k

Loan

- A single ~~ino~~ association among two entity sets is a relationship instance. Eg - Cust_name A borrows 20k ~~fro~~ with loan no. L-7.



⇒ Access date specifies the most recent date on which a customer accessed his account.

- The attribute associated with a relationship is called descriptive or relationship attribute.

Degree of a Relationship Set

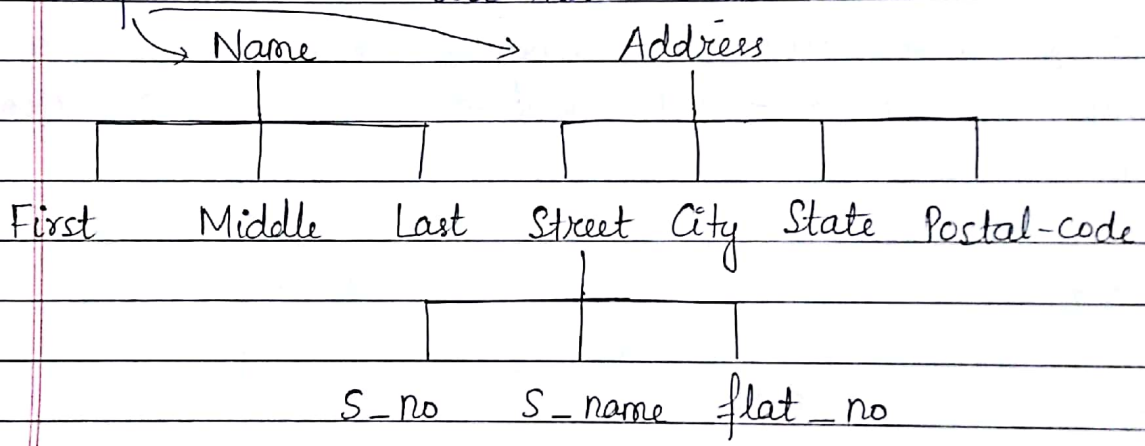
- It refers to the no. of entity sets that participate in a relationship set.
- 2 entity sets → Binary (Degree 2)
  3 entity sets → Ternary (Degree 3)
  4 entity sets → Quarternary (Degree 4)

Types of Attributes -

1) Simple & Composite Attributes -

→ Roll_No, Cust_ID, PNR_no.

Atomic/Simple attributes are those which cannot be divided further.
Composite " are those which can be divided further



2.) Single-valued & Multi-valued Attributes -

⇓

Phone_No, Email_ID, Names

3.) Base & Derived Attributes -

⇓ ⇓

DOB Age

Derived attributes are those whose values can be derived from the value of base attribute. Eg - if DOB is known, then the age can be computed.

- Constraints - rules that the model should obey.

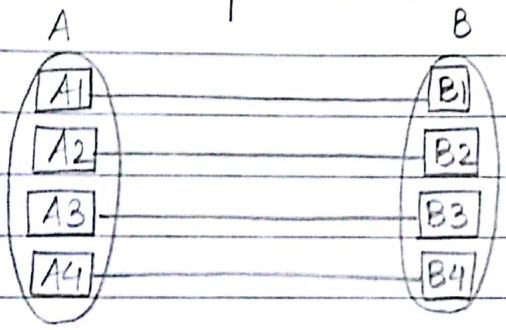
Types of constraints :-

(i) Mapping cardinalities
(ii) Key cardinalities
(iii) Participation cardinalities

(i) It expresses the no. of entities to which another entity set can be associated via a relationship set. They are most useful in describing binary relationship sets.

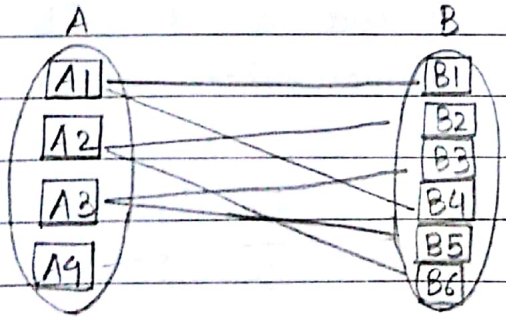
Four types of mapping cardinalities -

(a) One to One - Consider two entity sets A & B and R as the relationship set



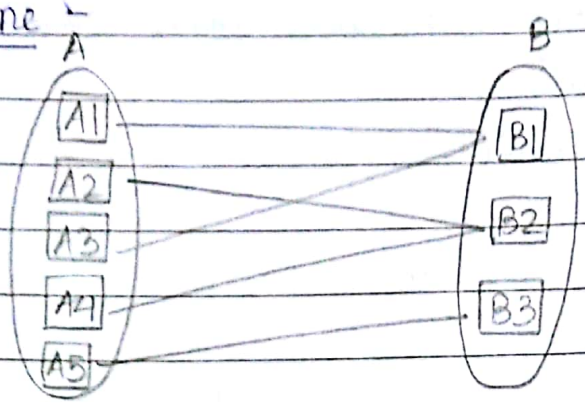
An entity in A is associated with atmost one entity in B and vice-versa.

(b) One to Many -



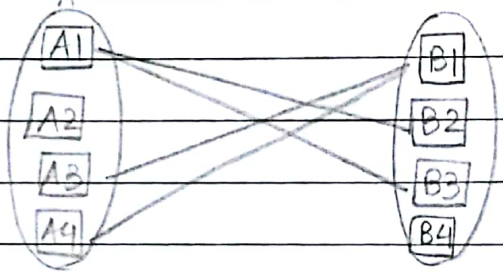
An entity in A is associated with 0 or more entities in B and an entity in B is associated with atmost one entity in A.

(c) Many to One –



An entity in A is associated with atmost one entity in B and an entity in B can be associated with 0 or more entities in A.

(d) Many to Many –



In both the sets A and B, one entity of a set can be associated with 0 or more entities in the other set.

NOTE – Some elements in A and B may not be mapped to any element in the other side.

(ii) Key Constraints – A key allows us to identify a set of attributes that is sufficient to uniquely identify entities from each other.

Types of Keys –

• Super Key • Candidate Key • Primary Key

– A super key is a set of one or more attributes that, when taken collectively allows us to uniquely identify an entity in the entity set.

If K is a super key, then any superset of K is also a super key.
Eg - Roll_No ⇒ superkey
Roll_No + Name ⇒ superkey
Roll_No + Name + Branch ⇒ superkey

- Minimal super keys (super keys for which no proper subset is a super key)
Eg - If its a customer relation and assumption is that a single customer from single address is there, the keys can be -
{cust_id}, {cust_name, cust_address}

{cust_id, cust_name} is not a candidate key as cust_id alone is a candidate key.

- The candidate key which is chosen by the database designer as a principal means of identifying entities within an entity set is primary key. Its attributes never or rarely change. Eg - roll_no, emp_id, social_security_no., etc.

(iii) Participation Constraints - The participation of an entity set E in a relationship set R could be of 2 types -
- Total participation
- Partial participation

Total → every entity in E participates in atleast one relationship in R. Eg - Loan → Borrower

Partial → if only some entities in E participates in atleast one relationship in R. Eg - Customer → Borrower

Entity - Relationship Diagrams - express the overall logical structure of a database graphically.

- entity set
- attributes
- relationship set
- link attributes to entity sets and entity sets to relation sets
- multi-valued attribute
- derived attributes
- represents identifying relationship
- total participation of an entity in a relationship
- weak entity set
- Primary key

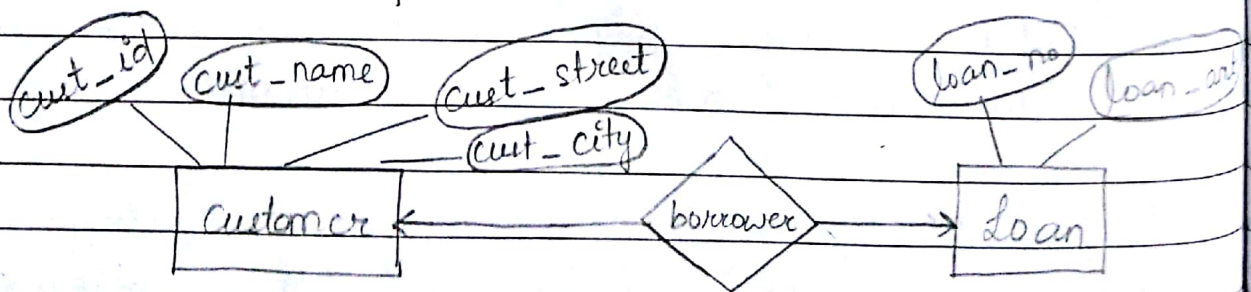
Attribute

Cardinality Constraints in an E/R diagram

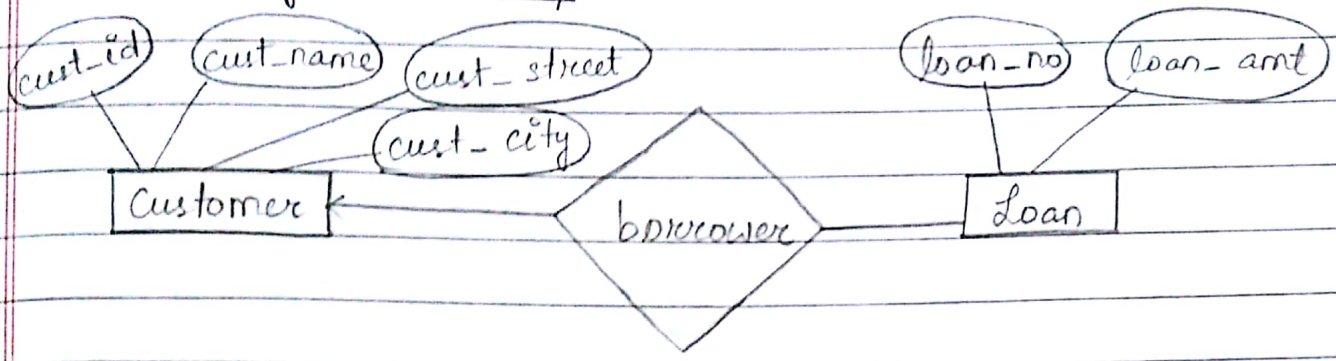
There are 4 types of cardinality constraints -

- One : represented by directed line (→)
- Many : represented by undirected line (—)

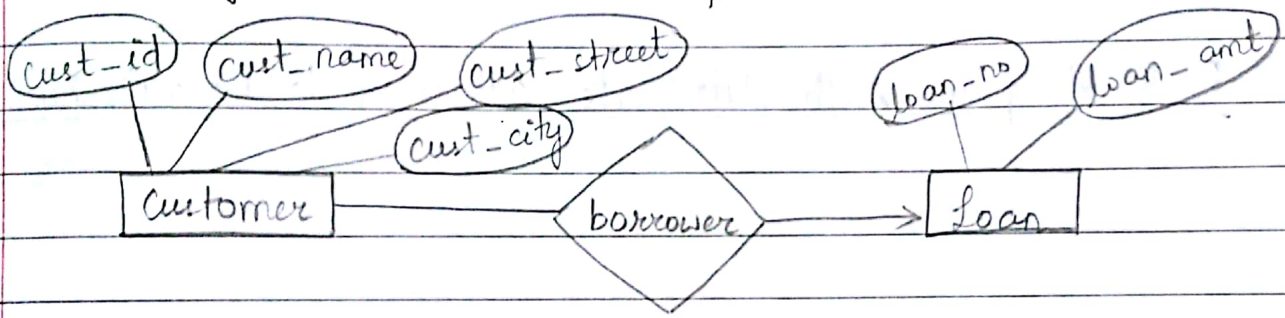
One to One Relationship :



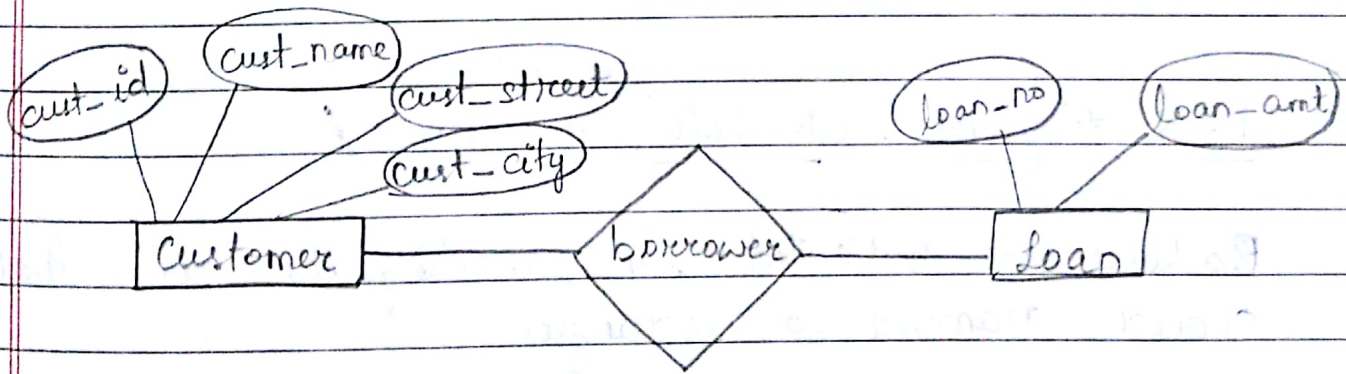
One to Many Relationship



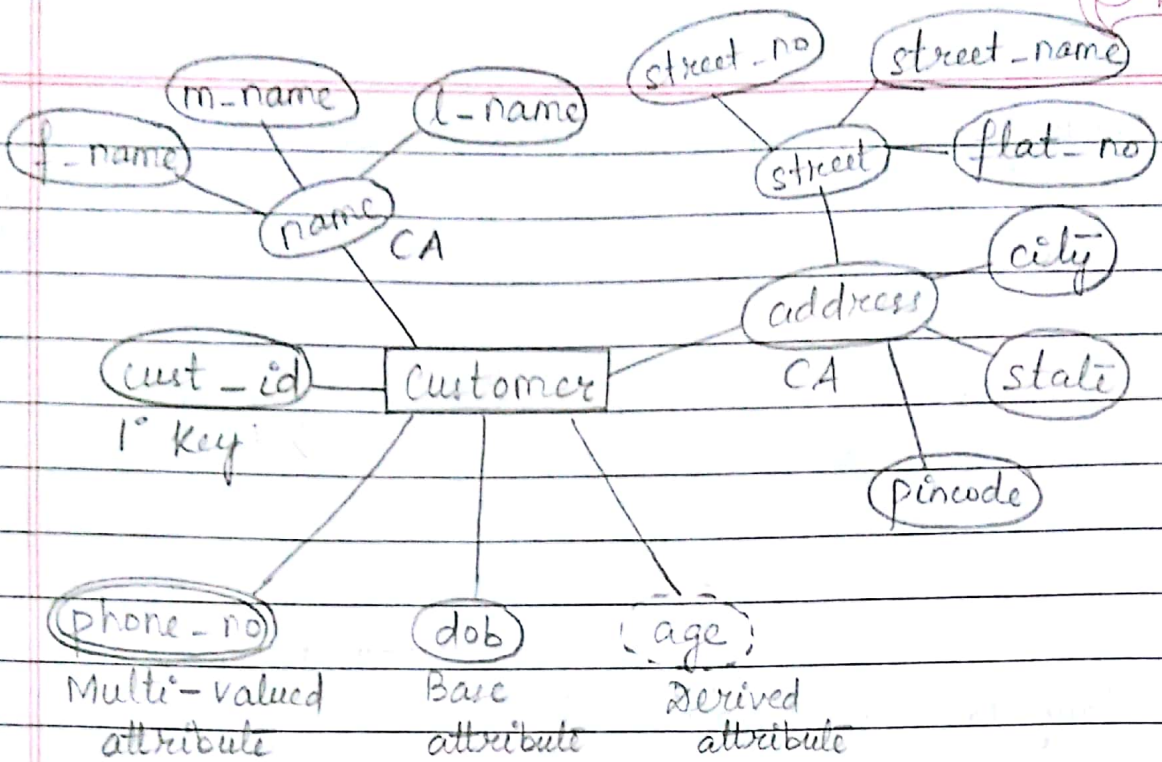
Many to One Relationship



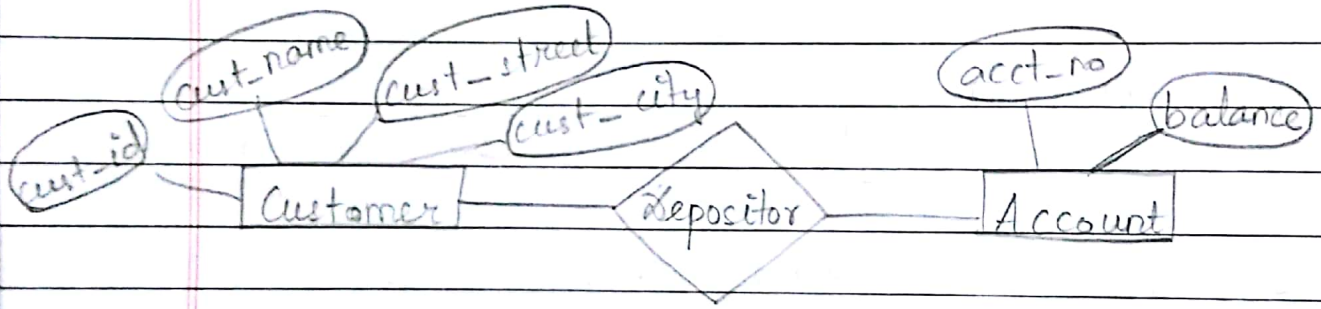
Many to Many Relationship



E/R diagram with composite, base & derived attributes

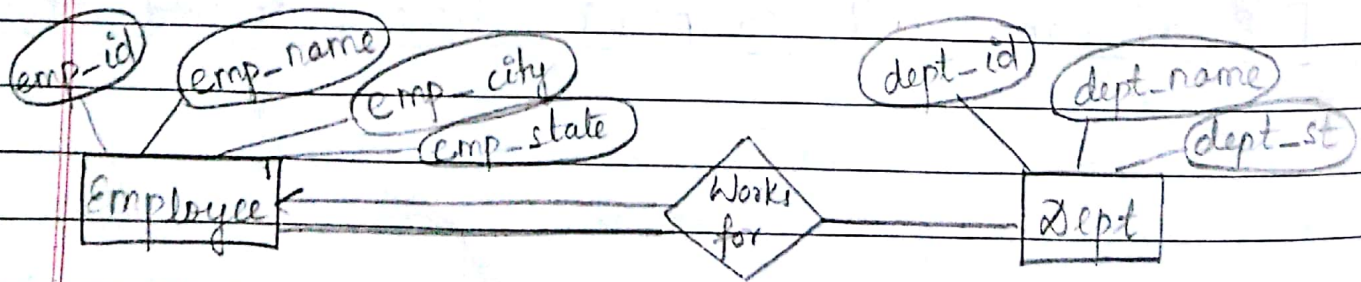


E-R Diagram with Attribute Attached to Relationship Set



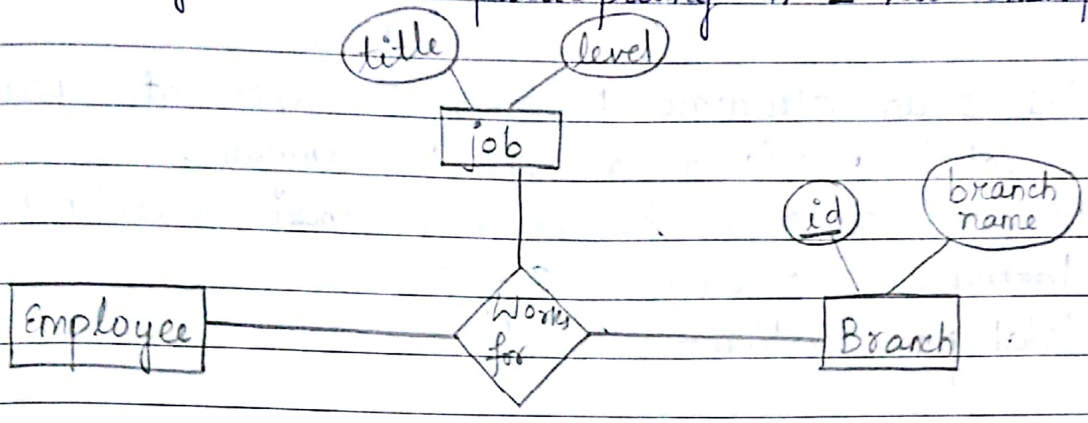
E-R diagram with Role Labels or Indicators

Roles are indicated by labelling the lines that connect diamond to rectangles.

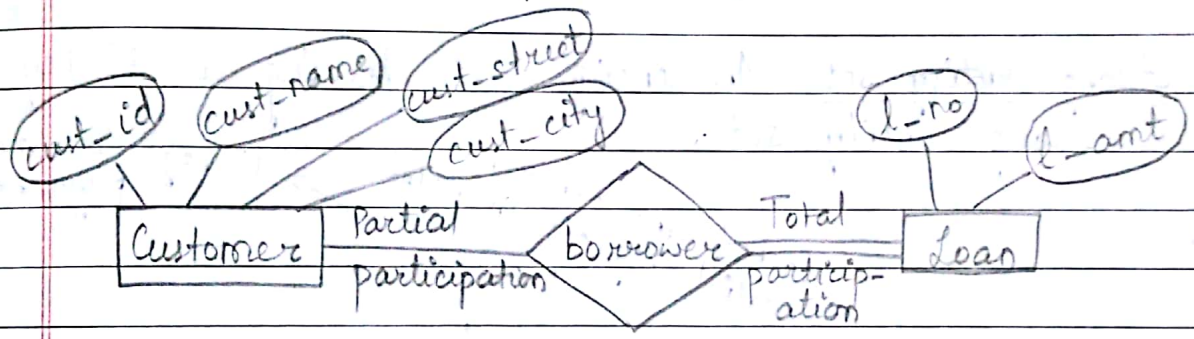


E-R Diagram with a Ternary Relationship

3 entity sets are participating in 1 relationship.



Total participation & Partial participation of an entity set in a Relationship Set



Cardinality Limits or Constraints on Relationship Set

E-R diagrams provide a way to indicate more complex constraints on a no. of times each entity participate in a relationship in a relationship set & edge b/w an entity set & binary relationship set can have an associated min^m cardinality & max^m cardinality value in the form of $l..h$

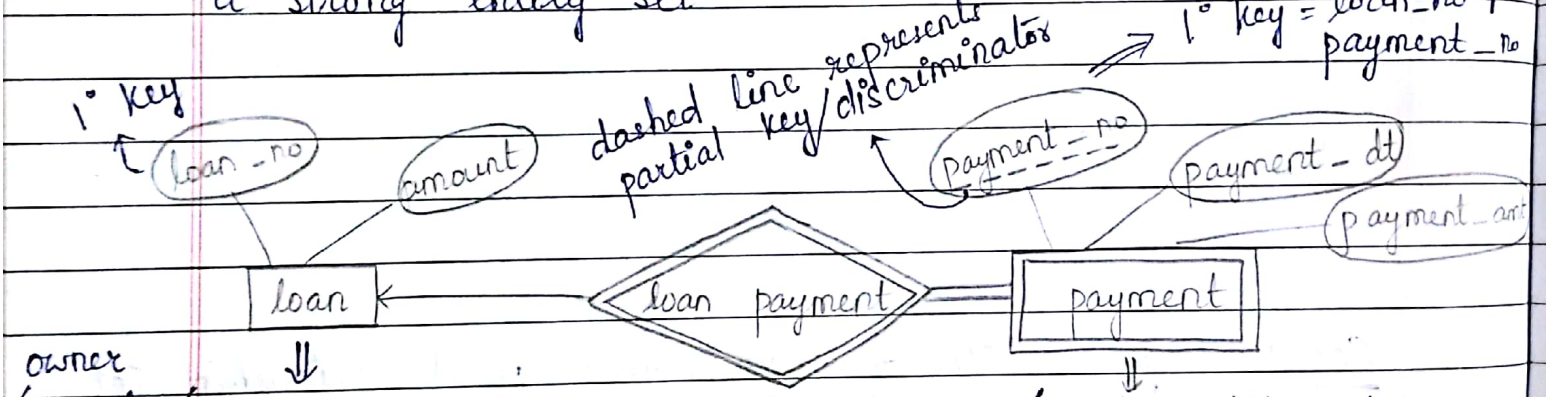
Both l & h are +ve & $l <= h$.

☆ If l=h i.e. 1..1 → total participation
☆ 1..☆ (asterisk indicates no limit)
☆ 0..☆ → partial participation.

This is an alternate notation to represent partial & total participation in E-R diagram.
For the customer, borrower & loan relationship.
Partial participation → 0..☆
Total participation → 1..1

Weak Entity Set - An entity set may not have sufficient attributes to form a primary key. Such entity set is called weak entity set.

Strong Entity Set - An entity set having a 1° key is a strong entity set.



owner/identifying Strong ES

existence dependent on the owner ES / Weak Entity Set

Payment_no is a sequential no. starting from one generated for each loan

- The relationship b/w a Strong ES and a Weak ES is called identifying relationship.

⇒ Identifying relationship is many to one from the Weak ES to the identifying ES and the participation of the

Weak ES in the relationship is total (as every loan payment is associated with some loan)

- The discriminator of a Weak ES is a set of attributes that allows the distinction to be made for various payments for a particular loan. It is also called partial key & represented by dashed line.

1° key of Weak ES = 1° key of owner ES ∪ Discriminator of Weak ES

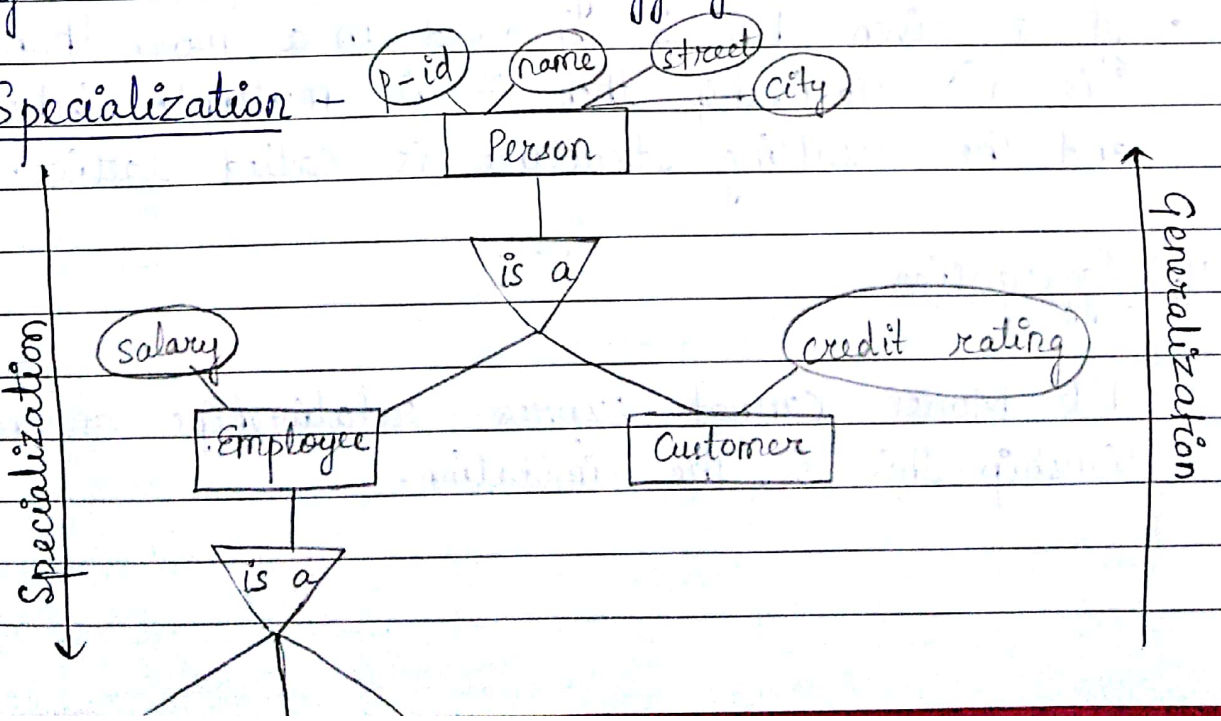
1° key = { loan_no, payment_no }

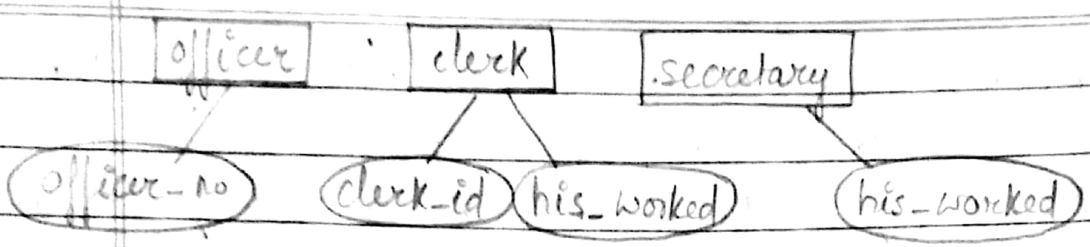
* It is also possible to have a weak ES with more than one identifying ES. Then the 1° key is the union of 1° keys of all identifying ES + the discriminator of the weak ES.

Extended E-R Features –

1.) Specialization
2.) Generalization
3.) Attribute Inheritance
4.) Aggregation.

1.) Specialization –





The process of designating sub-groupings within an ES is called specialization. It is a top-down design process depicted by ER diag. by a triangle labelled 'is a'. This represents a sub-class, super-class relationship.

The refinement from initial ES. into successive levels of the entity sub-groupings represent specia-

2.) lization. The design process may also proceed in a bottom up where multiple lower level entity sets are syntheseized / combined to a higher level ES. This is called generalization.

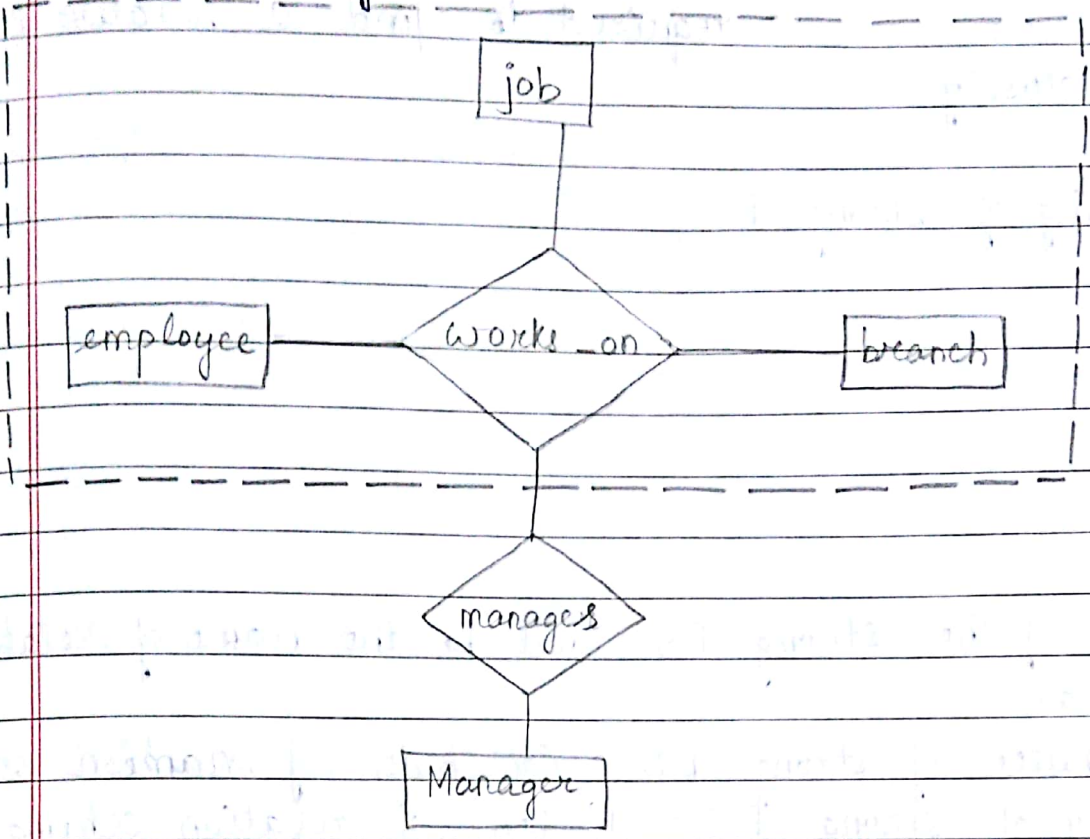
3.) A lower level ES inherits all the attributes & relationship participation of the higher level ES to which it is linked.

- If a given ES is involved in a single 'is a' relationship, it is single inheritance.
- If a given ES is involved in more than one 'is a' relationship, then it is multiple inheritance and the resulting structure is called lattice.

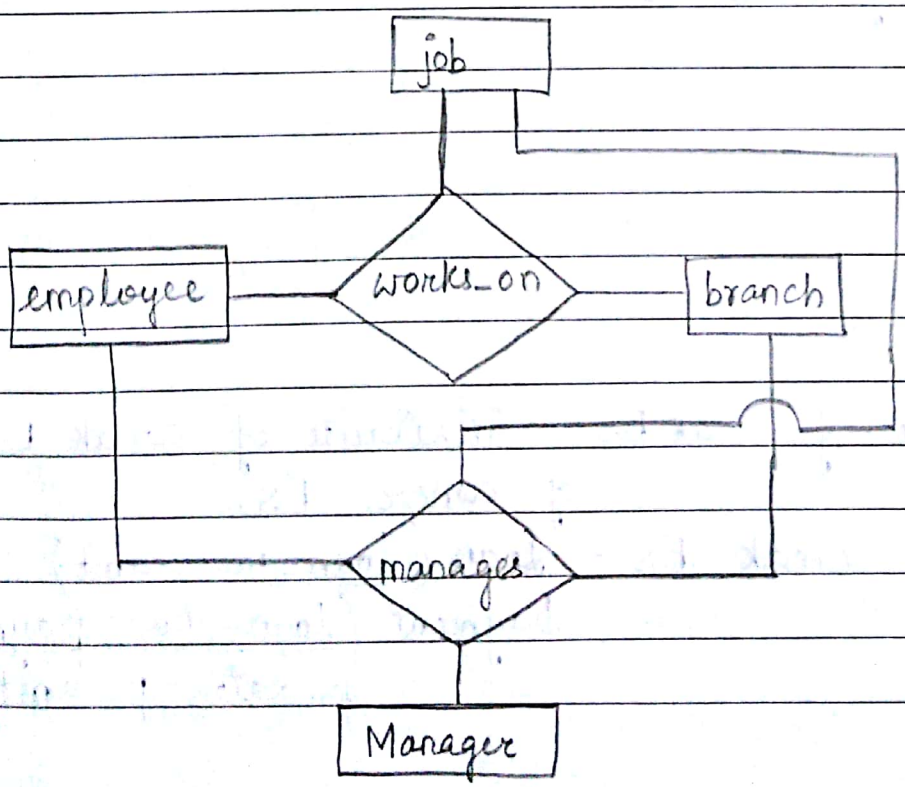
4.) Aggregation -

ER Model cannot express relationships among relationship. This is the limitation.

- Let us consider a quarternary relationship - employee, job, branch, manager.



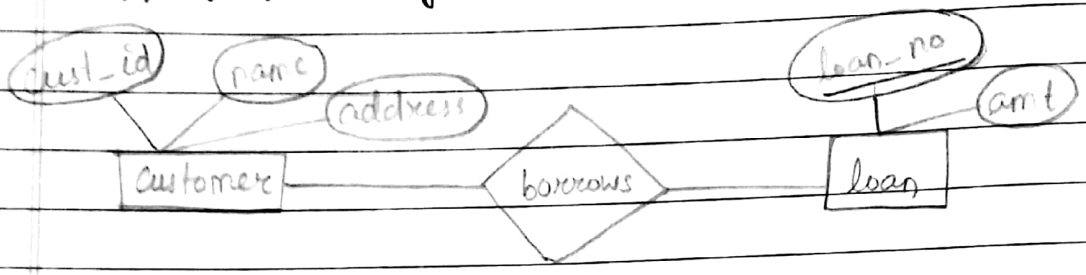
Aggregation is a process in which relationship b/w ES is treated as an abstract higher level ES & this can participate in relationship.



Mapping of ER diagram to Relation schema -

- required to find a relation schema that closely.

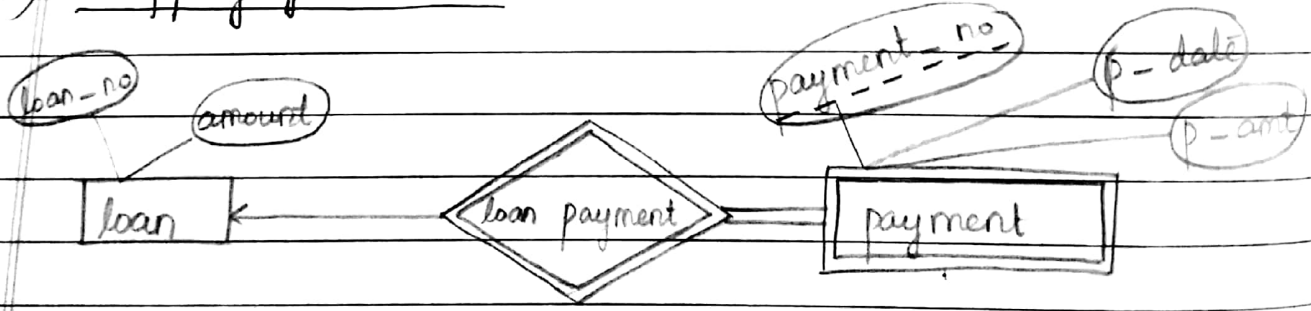
1.) Mapping of strong ES



- Name of the strong ES will be the name of relation schema.
- Attributes of strong ES = Attributes of relation schema
- 1° key of strong ES = 1° key of relation schema

customer (cust-id, name, address)
loan (loan-no, amt)

2.) Mapping of weak ES

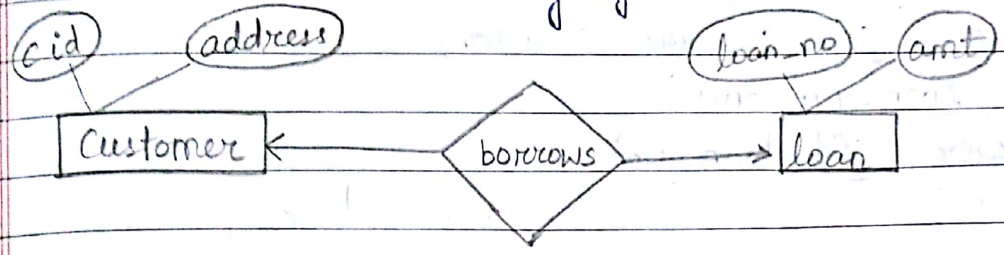


- Attributes of weak ES = Attributes of weak ES U 1°key of owner ES.
- 1° key of weak ES = loan (loan_no , amt)
  payment (loan-no , payment-no, p-date , p-amt)

3.) Mapping of Relationship Sets

- One-to-One :-

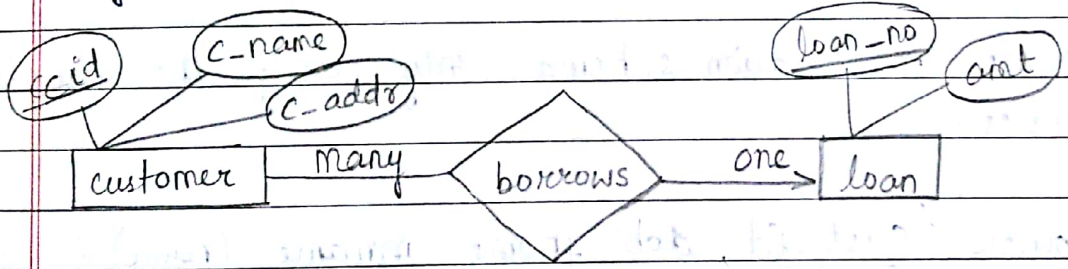
- Create 2 no. of relation schema. The 1° key of one ES is added to the 1° key of the 2nd relation schema.



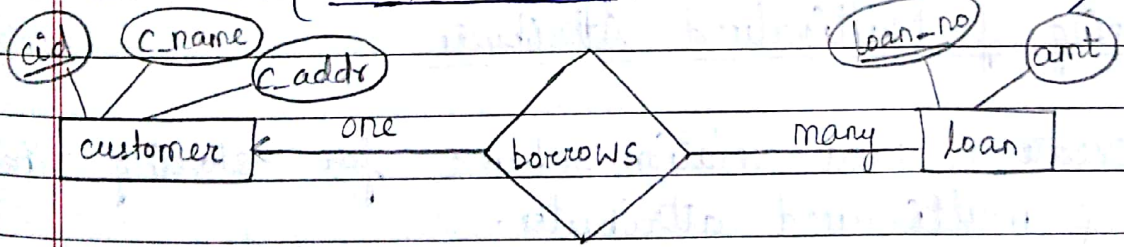
customer (cid , address)
loan (cid, loan_no, amt)
↳ for identifying which customer has taken which loan

- One-to-Many & Many-to-one :-

These type of sets can be represented by adding one extra side to the Many side containing the 1° key of the One side.



loan (loan_no, amt)
customer (cid, loan_no , c_name, c_addr)

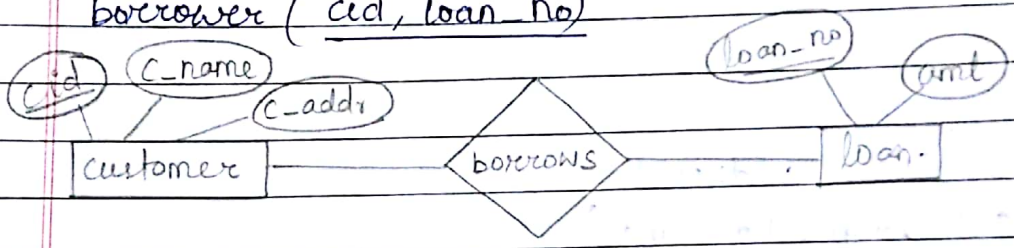


customer (cid , c_name, c_addr)
loan (loan_no, cid , amt)

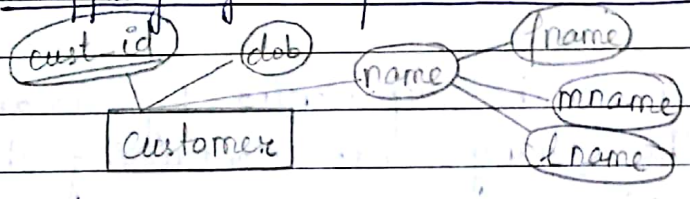
- **Many-to-Many -**

- We construct an additional schema with the same name as the relationship set & its attributes include the P° keys of the participating E.S.

customer (cid, c_name, c_addr)
loan (loan_no, amt)
borrower (cid, loan_no)



4.) **Mapping of Composite Attributes**

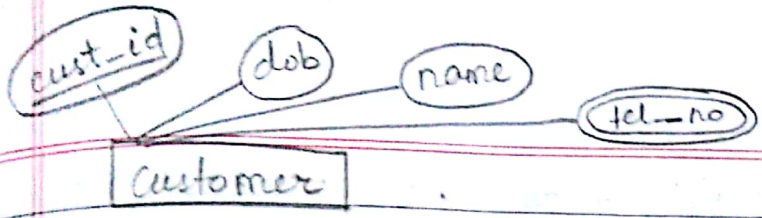


- We create the relation schema only using the simple attributes.

customer (cust_id, dob, fname, mname, lname)

5.) **Mapping of Multivalued Attributes**

We create a new relation schema for storing the key & multivalued attributes.



customer (cust_id, dob, name)
customer_tel (cust_id, tel_no)

To minimize the redundancy, this mapping is used.

HW Mapping of generalization & specialization.

QUERY LANGUAGES

- Users to request information from the database.
- It is of two categories -

(i) Procedural Query Language - User instructs the system to perform a sequence of operations on the database to compute the desired result. Eg ⇒ Relational Algebra

(ii) Non-Procedural Query Language - User asks for the desired info without specifying the procedure for obtaining the info. Eg ⇒ Tuple Relational Calculus (TRC)
⇒ Domain Relational Calculus (DRC)

Relational Algebra -

Fundamental RA operations -

(i) Select Operation (σ)
(ii) Project Operation (Π)
(iii) Rename Operation (ρ)
(iv) Union operation ($\cup$)
(v) Set - difference ($-$)
(vi) Cartesian - Product ($\times$)

(i) Select - It selects tuples that satisfies the selection predicate (p).

$$\sigma_p(r) = \{t \mid t \in \quad \wedge P(t)\}$$

where P = selection predicate containing operation. We can combine conditions using AND ($\wedge$), OR ($\vee$) and NOT ($\neg$)

Comparison - <attribute> op <attribute> or <constant>

Eg-

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

$\sigma_{A=B \wedge D>5}(r)$ ⇒

A	B	C	D
α	α	1	7
β	β	23	10

(ii) project - It is defined as the relation of K columns obtained by removing the columns that are not specified in the projection operation.

Let the no of columns be n.

$\Pi_{A1, A2, \ldots\ldots A_K}(r)$, $K \leq n$

- Duplicate rows are deleted.

r.

A	B	C
α	10	1
α	10	1
β	30	1
β	40	2

$\Pi_{A,C}(r)$, K = 2

A	C
α	1
β	1
β	2

The results of the array operations do not have a name that can be referred to them. ∴ Rename operation is used.

(iii) rename - $\rho_x(E)$

E : RA expression

X : name of result relation.

- The result of RA expressions don't have a name that can be used to refer to them. So, rename operation is used for this purpose. Eg-

A	B	C	D
α	α	1	7

$$\rho_X(\sigma_{A=B \wedge D>5}(r))\ X$$

Binary Operations

(iv) Union

Notation = $r \cup s$

$r \cup s \{ t \mid t \in r \text{ or } t \in s \}$

Union compatible :- 1) r & s must have same arity (same no. of attributes).

2.) Corr. attribute domains must be compatible.

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

$r \cup s$

A	B
α	1
α	2
β	1
β	3

(v) Set-difference - Notation : $r - s$

Def^n - $r - s = \{ t \mid t \in r \text{ and } t \notin s \}$

r & s must be compatible relations.

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

r − s

A	B
α	1
β	1

s − r

A	B
β	3

(vi) Cartesian Product operation :-

r, s r × s

Def^n - $r \times s = \{ tq \mid t \in r \text{ and } q \in s \}$

- As the same attribute may appear in both r & s, a naming schema is devised to distinguished b/w the attributes.

same attribute may be there

- The attribute name is preceded by relation name separated by (·) operator.
- This is done by attaching the name of the relation to the attribute name where the attribute name matches.

Eg- loan (loan_no, loan_name, amt)
borrower (c_name, loan_no)

Resultant relation- result (c_name, borrower·loan_no, loan·loan_no, b_name, amt)

Result = borrower × loan

Eg- r

A	B
α	1
β	2

s

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

$r \times s$ –

A	B	C	D	E
α	1	α	10	a
α	1	β	10	
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Duplicates won't be removed here. For that, another operation is there.

Additional RA operations –

1.) **Set - Intersection** – $r \cap s$ – all the tuples that are in both r and s.

☆ For this, r & s must be compatible.

Eg – $r \cap s = \{ t \mid t \in r \text{ and } t \in s \}$

$$r \cap s = r - (r - s)$$

S_1 =

sid	sname	rating	age
22	John	7	45
31	Smith	8	55
58	Joyce	10	35

S_2 =

sid	sname	rating	age
28	Frank	9	55
31	Smith	8	55
44	Will	5	35
58	Joyce	10	35

$S_1 \cap S_2$

2) Join operation :- ⋈

- Used to combine related tuples from two relations into single tuple in one relation.
- General Form : $R(A_1, A_2, A_3, \cdots A_n)$, $S(B_1, B_2, B_3, \cdots B_m)$

$$R \bowtie_{\text{<join condition>}} S$$

- Result of join :- Q with (n+m) attributes.
  $(A_1, A_2, \cdots A_n, B_1, B_2, B_3, \cdots B_m)$
- Diff. b/w Cartesian product & Join operation :-

(i) In join, only combination of tuples satisfying the join condition appear in the result whereas in Cartesian product, all combinations of tuples are included in the result.

General Join condition -
<condition> AND <condition> AND .... AND <condition>
where each condition is of the form :

$A_i \; \theta \; B_j$

A_i = attribute of R (1st relation)
B_j = attribute of S (2nd relation)
θ = any of the comparison operators $\{=, <, >, <=, >=, \neq\}$

Theta Join - A join operation such as the general join condition is called θ-join.

Equi Join - In join condition, comparison with equality operation only.

In the result of equi join, always one or more pair of attributes have identical values in every tuple. Such

attribute values are unnecessary or they are called superfluous. So, another form of join called Natural Join is used to get rid of 2nd duplicate attribute values resulting from equi-join.

- Equi join is a special case of natural join.

Natural Join :-

- Join condition of natural join is same as equi-join.
- It is denoted by $R *_{x=y} S$
- Eg -

R

NR	Tel
Joe	1234
Jill	1244
Bill	2244

S

NS	Addr
Joe	NA1
Jill	NA3
Bob	NG5

Equi Join - $Q \leftarrow R \bowtie_{NR=NS} S$

(← : assignment operator)

NR	Tel	NS	Addr
Joe	1234	Joe	NA1
Jill	1244	Jill	NA3

Natural Join - $Q \leftarrow R *_{NR=NS} S$

NR	Tel	Addr
Joe	1234	NA1
Jill	1244	NA3

∵ NS appears later, it is not appearing in the table

Q Express the expression of equi join & natural join from Cartesian product.

Equi-Join - $R \bowtie_{x=y} S = \sigma_{x=y}(R \times S)$

Natural-Join - Apply project operation to remove a column.

$R *_{x=y} S = \Pi_{A=y}(\sigma_{x=y}(R \times S))$

↳ It will project all attributes except y

$R * S$ = combination of all attributes

Outer Join :- It is an extension of join operation operation to deal with missing information.

emp (e)

e_name	street	city
A	S1	C1
B	S2	C2
C	S3	C3
D	S4	C4

works (w)

e_name	b_name	salary
A	B1	15k
B	B1	13k
CC	B2	53k
D	B2	15k

Natural join

emp $*_{e.e\_name = w.e\_name}$ works

e_name	street	city	b_name	salary
A	S1	C1	B1	15k
B	S2	C2	B1	13k
D	S4	C4	B2	15k

In the above process, the info about C & CC is lost in the natural join. ∴ Outer join is used.

There are 3 types of outer join :-

(i) Left Outer Join - It takes all tuples in the left relation that did not match with any tuple in the right relation. (⟕)

- Null values are inserted for the attributes from the right relation wherever there is a mismatch.

e_name	street	city	b_name	salary
A	S1	C1	B1	15k
B	S2	C2	B1	13k
D	S4	C4	B2	15k
C	S3	C3	NULL	NULL

Left outer Join (⟕) :- Employee ⟕ works

(ii) Right Outer Join :-

e_name	street	city	b_name	salary
A				
B				
D				
Cc	NULL	NULL		

Right Outer Join - (⟖) - Employee ⟖ works

3.) Division operation - Let R have two fields x and y and S has one field y.

R/S or R ÷ S is mathematically defined as the set of all tuples

$$R/S = \{\langle x \rangle \mid \exists \langle x, y \rangle \in R \ \forall \ \langle y \rangle \in S\}$$

Relation R -

S no	P no
S1	P1
S1	P2
S1	P3
S1	P4
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4

S1

P no
P2

R ÷ S1

S no
S1
S2
S3
S4

* Whebre there is P2, write the sno in R ÷ S1

S2

P no
P2
P4

Combination of P2 & P4 is seen & not the individual ones.

R ÷ S2

S no
S1
S4

S3	pno
	P1
	P2
	P4

R ÷ S3

sno
S1

Extended RA operations

1) Generalized Projection – This operation extends the projection operation by allowing arithmetic of expressions or functions to be used in the projection list. So, it is of the form –

$\Pi_{F1, F2, F3, \ldots Fn}(E)$ where E = any RA expression

$F_1, F_2, \ldots F_n$ = arithmetic expressions involving constants & attributes of the schema of E.

For Eg – credit_info –

cust_name	c_limit	c_balance (amt spent)
Will	20K	17.5K
Hey	15K	15K
Jon	60K	7K
Smith	20K	4K

If we want to know how much more each person can spend, what would be the RA expression –

$\Pi_{cust\_name, c\_limit - c\_balance}(credit\_info)$

Here, two columns will be there but there won't be a name for the 2nd column. We can apply

the rename operat^n to the result of generalized operat^n projection to give a name to the column. So, we have to modify the query as –

$\Pi_{cust\_name, (c\_limit - c\_balance)\ as\ credit\_avl}\ (credit\_info)$

O/P –

cust_name	credit_avl
Will	2.5K
Hey	0
Jon	53K
Smith	16 K

2.) Aggregate Functions – Take a collection of values & return a single values as the result. The functions are – SUM, AVG, MIN, MAX, COUNT

– The general form of aggregate operation –

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

where E = any RA expression

$G_1, G_2, \ldots G_n$ = list of attributes on which to group (this list could be empty)

F_i = an aggregate func.

A_i = attribute name

Emp

Name	Office_Id	Dept	Salary
Smith	400	CSE	45K
Jones	220	ECO	35K
Adams	100	ECO	50K
Brown	422	CSE	65K
Smith	500	Fin	60K

★ Query to find the min^m salary –

$$\mathcal{G}_{MIN(salary)}(Emp)$$

Min (salary)
35k

★ Query to find the avg. salary –

$$\mathcal{G}_{AVG(salary)}(Emp)$$

★ Query to count the no. of Employees in the CSE dept –

$$\mathcal{G}_{COUNT(Name)}(\sigma_{Dept=CSE}(Emp))$$ O/P = 2

OR

$$\mathcal{G}_{COUNT(Dept.)}(\sigma_{Dept=CSE}(Emp))$$

★ Query to find the total salary of the Economics dept –

$$\mathcal{G}_{SUM(salary)}(\sigma_{Dept=Eco}(Emp))$$

– If we want to eliminate duplicates while counting, we use the word distinct at the end of func. name.

Now, COUNT is replaced by COUNT_DISTINCT

★ Query to count the distinct employee names –

$$\mathcal{G}_{COUNT\_DISTINCT(Name)}(Emp)$$

- Suppose we want to find the sum of salaries of employees at each dept.

$$Dept\ \mathcal{G}_{SUM(salary)}\ (Emp)$$

o/p = Dept	SUM (salary)
CSE	
ECO	
Fin	

Modification of the Database

The content of the DB get modified using -
(i) Deletion (ii) Insertion (iii) Updation.

(i) Deletion - selected tuples are removed from the DB.
In RA, a deletion operat^n is expressed as -

$$r \leftarrow r - E$$

(E → RA query; r → relation)

For eg - depositor (c_name, acct_no)
loan (loan_no, b_name, amt)

★ Delete all the records of Smith's account from depositor.

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

★ Delete all the loans with amt. in the range of 50K to 1 lakh.

$loan \leftarrow loan - \sigma_{amt >= 50000 \wedge amt <= 100000}(loan)$

(ii) Insertion - To insert data into a relation either a tuple or set of tuples to be inserted is specified.

RA expression - $r \leftarrow r \cup E$

account (acct_no, b_name, balance)
depositor (c_name, acct_no)

★ Suppose, we want to insert the fact that Smith has 12K rupees in account A973 at Downtown branch

account ← account ∪ {(A973, "Downtown", 12000)}
depositor ← depositor ∪ {("Smith", A973)}

(iii) Updation - We use generalized projection operator to change a value in a tuple without changing all the attribute values in the tuple.

$r \leftarrow \Pi_{F1, F2, \ldots Fn}(r)$

where F_i = ith attribute of r if not updated or F_i is an expression that gives the new value of the attribute.

★ All balances in accounts are to be increased by 5%. What is the RA expression ?

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

Tuple Variable – A single or set of tuples of a particular relation are taken as values in these variables.

* Accounts with balances over 10K receive 6% interest and all others receive 5% interest. Write the RA expression.

account ← $\Pi_{acct\_no,\ b\_name,\ balance * 1.06}(\sigma_{balance > 10000}(account)) \cup \Pi_{acct\_no,\ b\_name,\ balance * 1.05}(\sigma_{balance < 10000}(account))$

OR

General Formula –

$r \leftarrow \Pi_{F1, F2, \ldots Fn}(\sigma_p(r)) \cup (r - \sigma_p(r))$

Other Relational Languages

Non-procedural Query Language –

It is of two types –
(i) Tuple Relational Calculus (TRC)
(ii) Domain Relational Calculus (DRC)

(i) TRC – Creates a new relation that is based on specifying a no. of tuple variables that range over rows of the stored DB relations.

General Form of a TRC Query – $\{t \mid COND(t)\}$ or $\{t \mid P(t)\}$

Where t = tuple variable
COND(t) or P(t) = conditional expression involving the variable t

The result of this query is the set of tuples P that satisfies the condition given in the RHS of the vertical bar.

$t[A]$ - signifies value of tuple t at attribute A where $t \in$ relation schema r.

Eg - We consider a banking example with the following relation schema.

branch (b_name, b_city, assets)
customer (c_name, c_street, c_city)
account (acc_no, acc_name, balance)
loan (loan_no, b_name, amt)
depositor (c_name, acct_no)
borrower (c_name, loan_no)

- Query 1 - Find the branch name, loan no. & amount for loans of over $1200.

$$\{t \mid \underbrace{t \in loan \wedge t[amt] > 1200}_{\text{condition}}\}$$

- Query 2 - Suppose we want only the loan_no attribute rather than all the attributes of the loan relation.

$$\{t \mid \exists s \in loan\ (t[loan\_no] = s[loan\_no] \wedge s[amt] > 1200)\}$$

#

★ For answering these type of queries, we need existential quantifier ($\exists$ (there exists)).

s : tuple variables in the loan relation
t : tuples in result relation $t[loan\_no]$

Generalization – $\exists\, s \in r\,(C(s))$

It means that there exists a tuple variable s in relation r such that C(s) is true.

The result is a relation on known no. that satisfies the condition.

- Query 3 – Find the names of all customers who have a loan from Delhi branch.

In RA, natural join is accompanied by select operation.

Here, 3 tuple variables are used.

s & u : tuple variables in borrower & loan relation respectively.
t : tuple variable of result relation.

$$\{t \mid \exists\, s \in borrower\,(t[c\_name] = s[c\_name] \wedge \exists\, u \in loan(\underbrace{u[loan\_no] = s[loan\_no]}_{\text{join condition}}) \wedge u[b\_name] = \text{"Delhi"})\}$$

The result is a relation of customer name that satisfies the condition.

- Query 4 – Find all customers who have a loan or account at the bank (its union operation in RA).

We have to take the union of borrower & depositor.

s : tuple variable for borrower
u : tuple variable for depositor
t : tuple variable for the result

$\{t \mid \exists s \in borrower\ (t[c\_name] = s[c\_name] \vee \exists u \in depositor\ (t[c\_name] = u[c\_name])\}$

- Query 5 - Find the names of those customers who have both an account & a loan at the bank.

$\{t \mid \exists s \in borrower\ (t[c\_name] = s[c\_name] \wedge \exists u \in depositor\ (t[c\_name] = u[c\_name])\}$

- Query 6 - Find all customers who have an account but do not have a loan at the bank.

Here, we need to eliminate those customers who appear in borrower relation.

$\{t \mid \exists u \in depositor\ (t[c\_name] = u[c\_name]) \wedge \neg \exists s \in borrower\ (t[c\_name] = s[c\_name])\}$

Free & Bound Variables

- A variable is said to be free unless it is quantified by $\forall$ (universal quantifier) or $\exists$ (existential quantifier)

Eg - $\{t \in loan \wedge \exists s \in customer\ (t[b\_name] = s[b\_name])$

t : free
s : bound

Tuple Calculus Formula

- It is made up of atoms where the atom is of the following form.
- If s - tuple variable & r - relational instance, then an atom is of the form -

$$s[x] \ \theta \ u[y] \rightarrow \text{atom}$$

x = attribute on which s is defined
y = " " " u " "
θ = comparison operator ($<, >, <=, >=, =, \neq$)

* Domains of x & y must have values which can be compared by θ.

OR

$$s[x] \ \theta \ c$$

c = constant in the domain of x.

Rules for building formula from Atoms

- An atom is a formula.
- If P_1 is a formula, then $\neg P_1$ or $(P1)$ is also a formula.
- If P_1 and P_2 are formulae, then -
  $P1 \vee P2$, $P1 \wedge P2$, $P1 \Rightarrow P_2$ are also formulae.

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

Safety of Expressions - A TRC expression is said to be safe if it takes values from the ^ domain of the formula P, then dom (P) must produce a

finite no. of tuples as its result.

Eg - $\{t \mid \neg(t \in loan)\}$
It will include all tuples ($\neg t \in r$)
- There are infinitely many tuples which are not in loan.

Domain Relational Calculus

Domain Variable - They range over single columns from domain of attributes.

Definition of DRC -

A DRC expression is written as -
$\{\langle x_1, x_2, \ldots x_n \rangle \mid P(x_1, x_2, \ldots x_n)\}$

TRC → $\{t \mid P(t)\}$

- Query 1 - Find the loan no, branch name & amount for loans over $1200.

$\{\langle l, b, a \rangle \mid \langle l, b, a \rangle \in loan \wedge a > 1200\}$

- Query 2 - Find all loan nos. for loans with an amount greater than $1200.

Rules - Specify the requested attribute before the vertical bar by the free domain variables. The condition is specified following the vertical bar.

$\{\langle l \rangle \mid \exists b, a (\langle l, b, a \rangle \in loan \wedge a > 1200)\}$

- Query 3 – Find the names of all customers who have a loan from Delhi branch and also find the loan amounts.

c_name → amt

c, a : free variables

l, b : bound variables

l (loan_no) → b_name

$\{ \langle c, a \rangle \mid \exists\, l (\langle c, l \rangle \in \text{borrower} \wedge \exists\, l, b (\langle l, b, a \rangle \in \text{loan} \wedge b = \text{"Delhi"}))\}$

- Query 4 – Find the names of all customers who have a loan and account or both at the Delhi branch.

Join loan – borrower, depositor – account.
Then apply union.