E-R Model

E-R stands for Entity Relationship.

An E-R model describes the overall logical structure of a database.

Basic terminology

Entity :- An entity is a thing or object that is distinguishable from all other objects.

E.g. Person

Entity Set :- An entity set is a set of entities of the same type that share the same properties or attributes.

E.g. The set of all persons who are customers at a given bank can be defined as entity set customer.

An entity is represented by a set of attributes.

Attribute :- An attribute is a descriptive property possessed by each member of an entity set.

Attributes can be classified as

1. Simple attribute
2. Composite attribute

Simple attribute :- An attribute which cannot be divided into subparts is known as simple attribute.

E.g. customer_id

Composite attribute :- An attribute which can be divided into subparts is known as composite attribute.

E.g. customer_name

first_name middle_initial last_name.

Attributes can also be classified into

1. Single-valued attribute
2. Multivalued attribute.

Single-valued attribute :- An attribute which possess a single value for a particular entity is known as single-valued attribute.

E.g. customer_id

Multi-valued attribute :- An attribute which possess multiple values for a particular entity is known as Multivalued attribute.

E.g. phonenumber.

Derived attribute :- An attribute whose value can be derived from the value of other attributes is known as Derived attribute.

E.g. customer_age

$$customer\_age = currentdate - customer\_DOB$$

(DOB = Date of Birth)

Relationship :- A relationship is an association among several entities.

Relationship set :- A relationship set is a set of relationships of the same type.

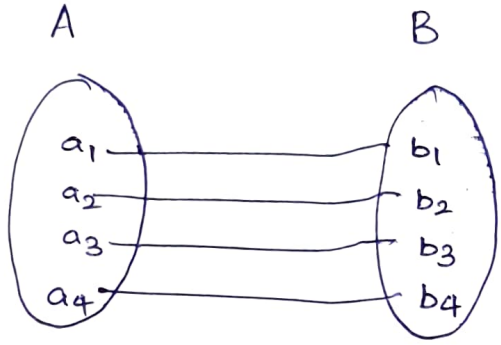
Mapping cardinalities (cardinality ratios) :-

It express the number of entities to which another entity can be associated via a relationship set.

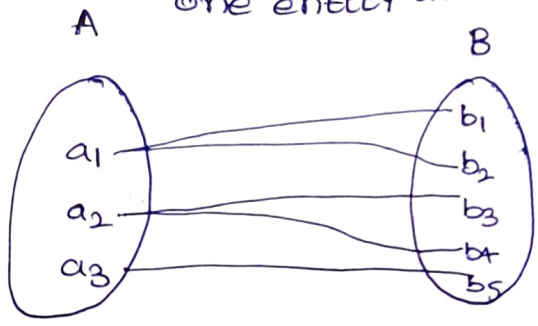
For a binary relationship set R between entity sets A and B, the mapping cardinality must be one of the following :

* one-to-one
* one-to-many
* Many-to-one
* Many-to-many.

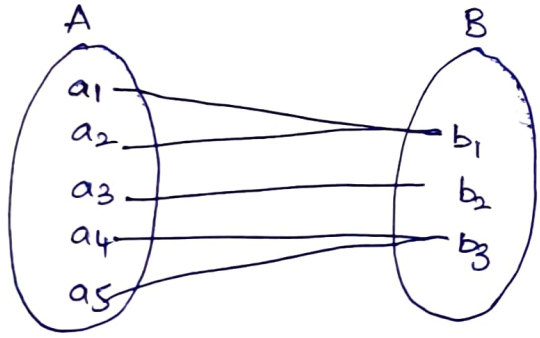
1. One-to-one :- An entity in A is associated with atmost one entity in B and an entity in B is associated with atmost one entity in A.



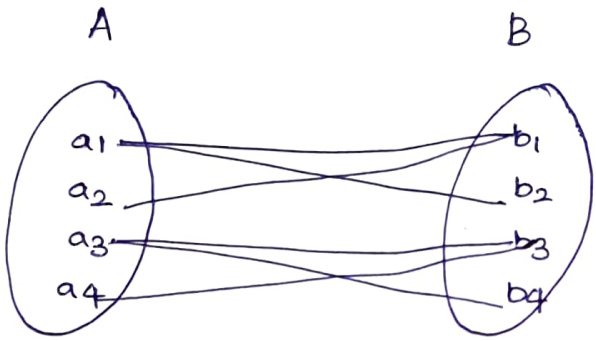
2. One-to-many :- An entity in A is associated with zero or more entities in B. An entity in B can be associated with atmost one entity in A.



3. Many-to-one :- An entity in A is associated with atmost one entity in B. An entity in B can be associated with zero or more entities in A.



4. Many-to-many :- An entity in A is associated with zero or more entities in B and an entity in B is associated with zero or more entities in A.

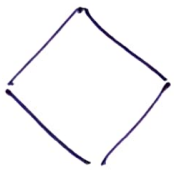


Notations Used in E-R Diagrams

An E-R diagram consists of the following symbols

 → Rectangle — Entity Sets

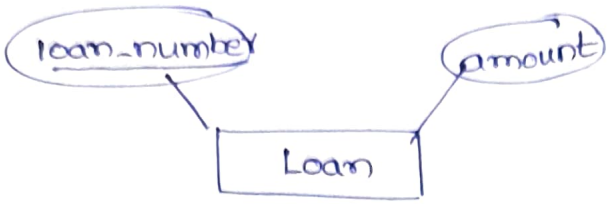
 → Ellipse — Attribute.

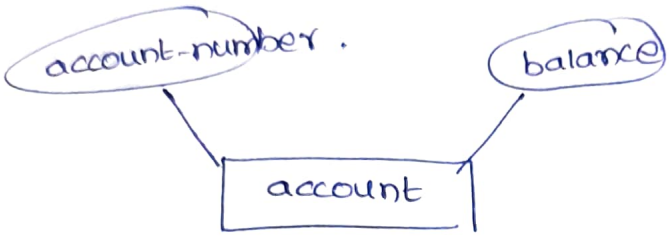
 → Diamond — Relationship Set

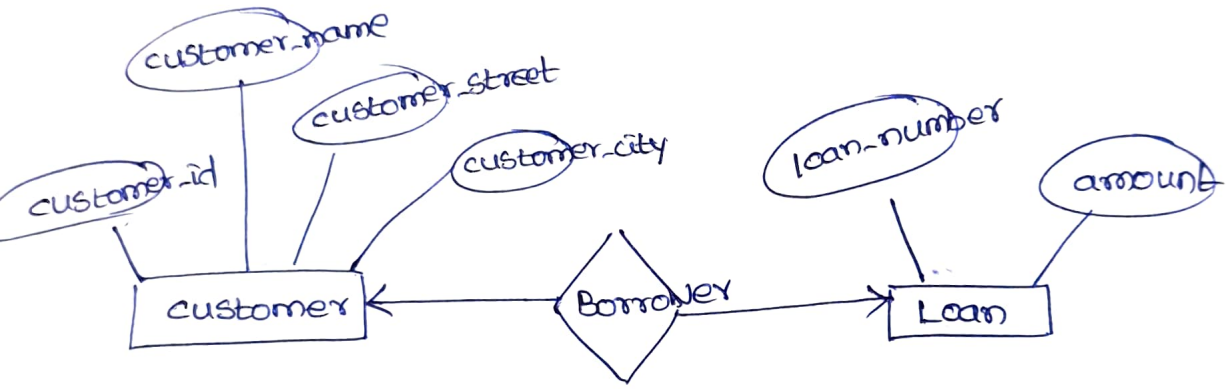
 → Line — links attributes to entity sets and entity sets to relationship sets.

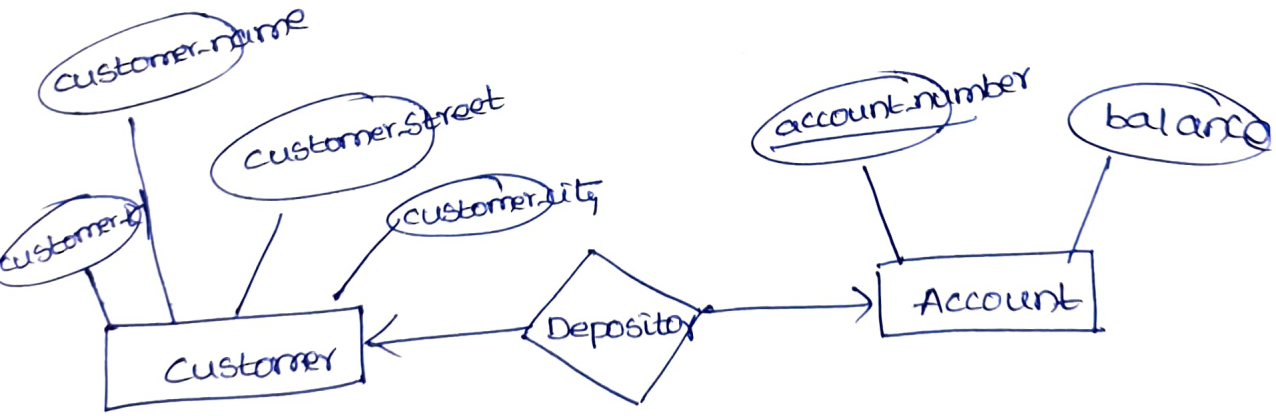
 → Double Ellipse — Multivalued attributes.

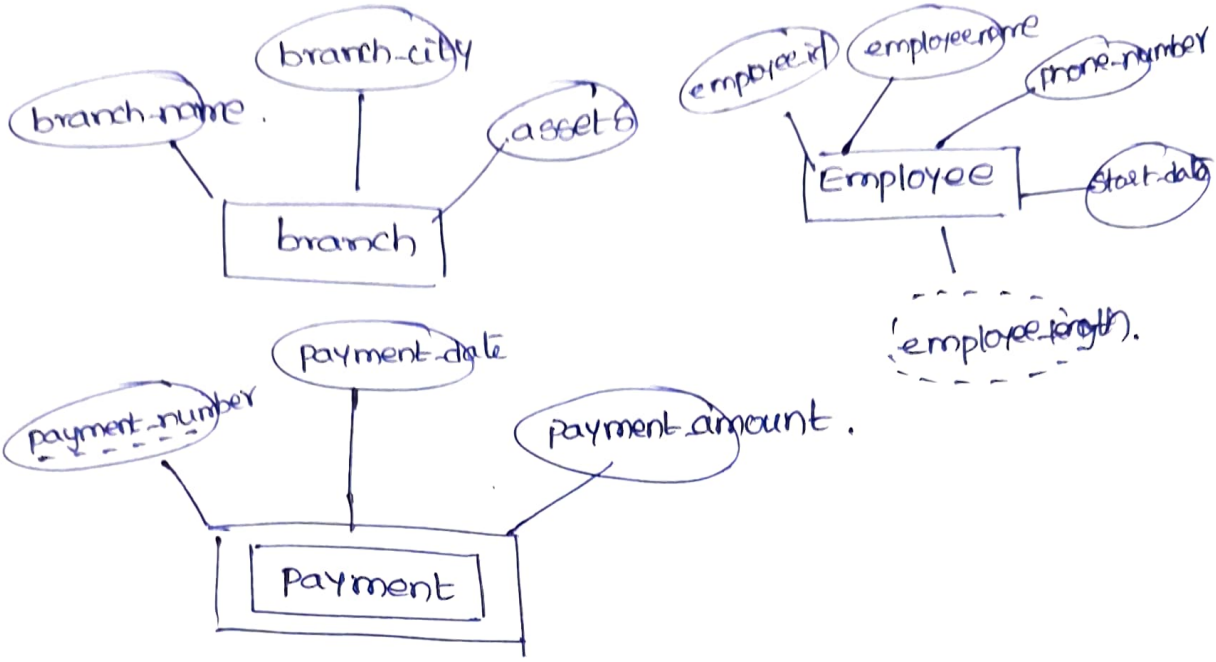
An E-R diagram with simple, composite, multivalued and derived attributes.

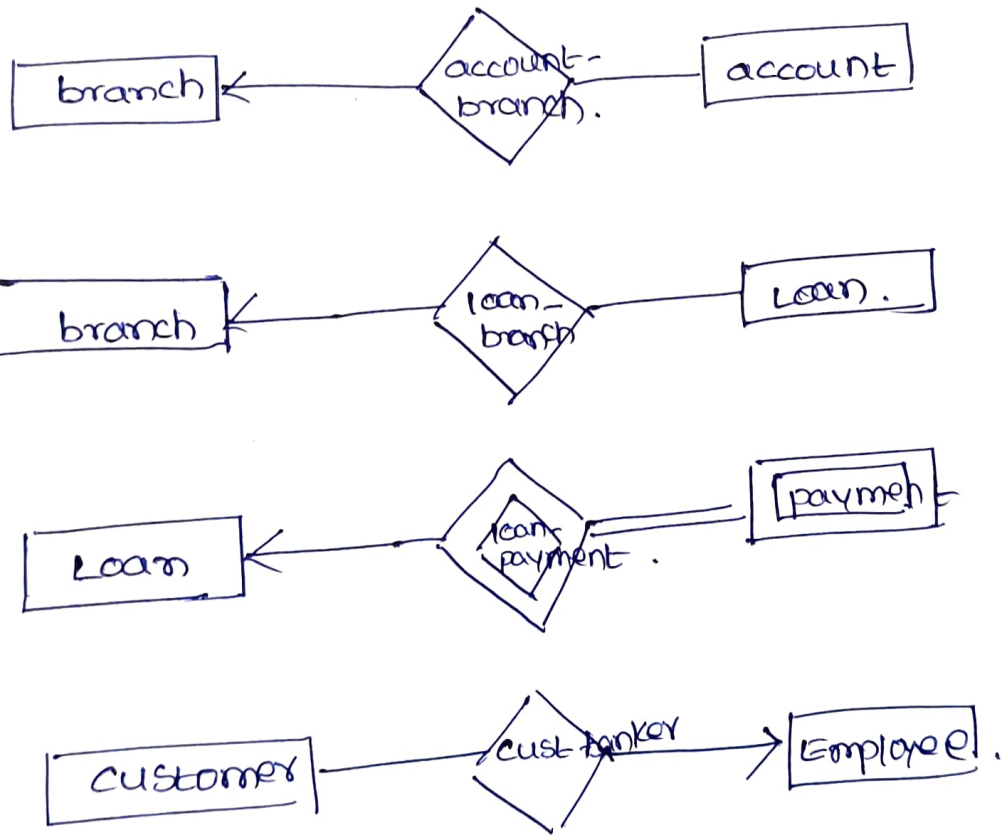

loan_number
amount
Loan


account-number
balance
account

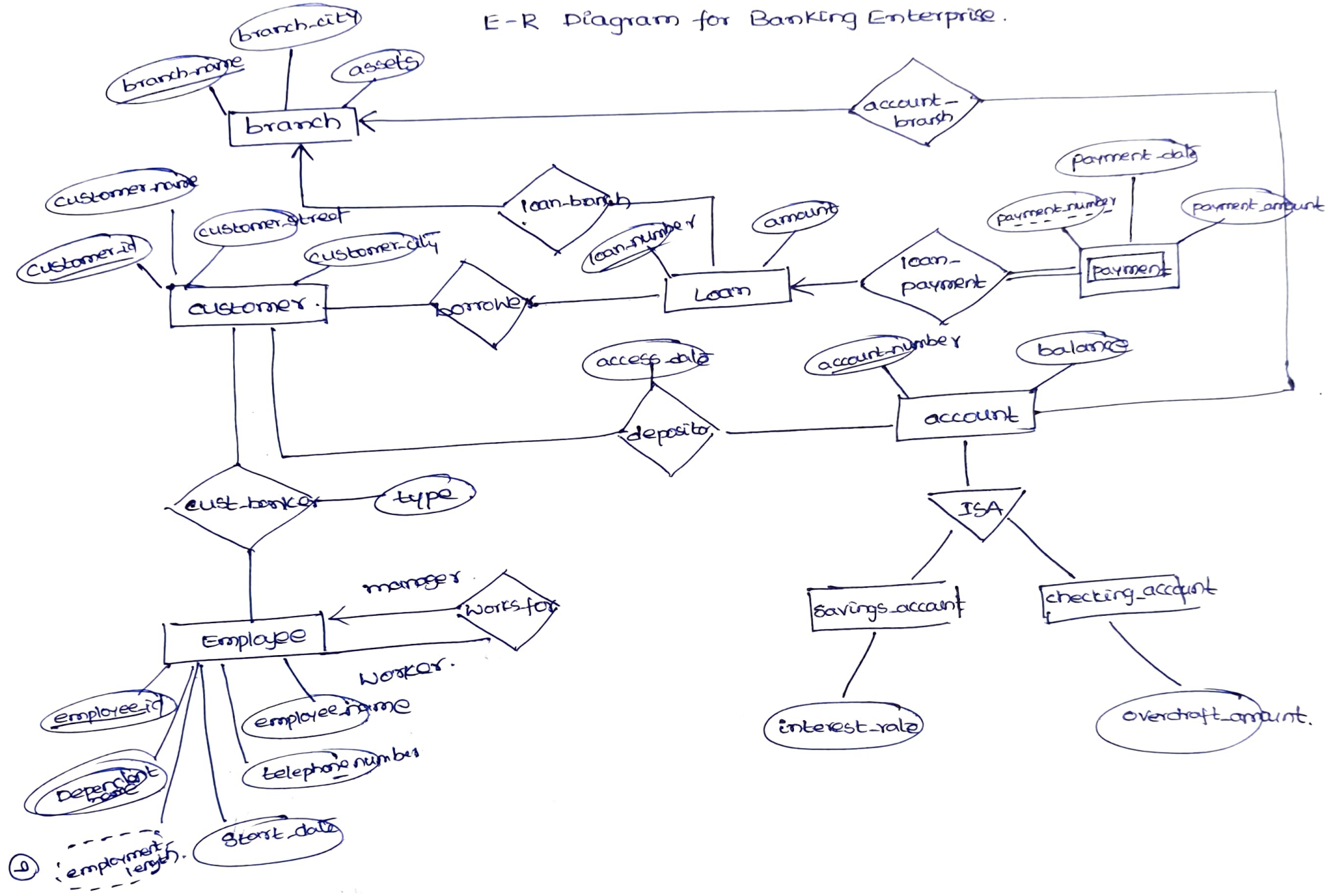

customer_name
customer_street
customer_id
customer_city
loan_number
amount
customer
Borrower
Loan


customer_name
customer_street
customer_id
customer_city
account_number
balance
Customer
Depositor
Account


branch_name
branch_city
assets
branch
employee_id
employee_name
phone_number
Employee
start_date
employee_length.
payment_date
payment_number
payment_amount.
Payment


branch
account_branch.
account
branch
loan_branch
Loan.
Loan
loan_payment.
customer
cust_banker
Employee.

E-R Diagram for Banking Enterprise.



Conversion of E-R Diagram to Relational Model

branch = (branch_name, branch_city, assets)

customer = (customer_id, customer_name, customer_street, customer_city)

loan = (loan_number, amount)

account = (account_number, balance)

employee = (employee_id, employee_name, telephone_number, start_date)

account_branch = (account_number, branch_name)

loan_branch = (loan_number, branch_name)

borrower = (customer_id, loan_number)

depositor = (customer_id, account_number)

cust_banker = (customer_id, employee_id, type)

works_for = (worker_employee_id, manager_employee_id)

payment = (loan_number, payment_number, payment_date, payment_amount)

savings_account = (account_number, interest_rate)

checking_account = (account_number, overdraft_amount)

Extended E-R features

* Specialization
* Generalization
* Attribute Inheritance.
* Aggregation

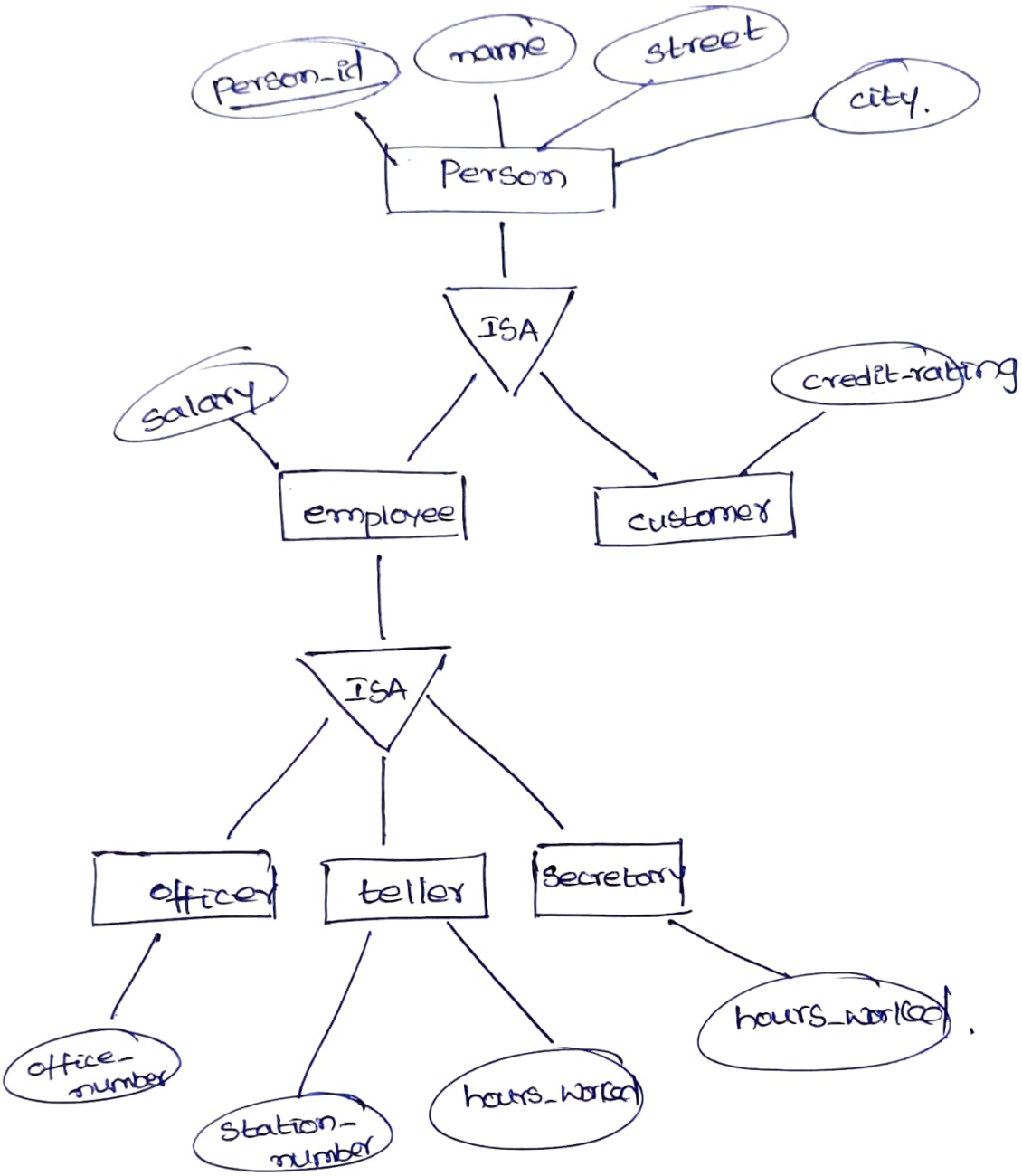
Specialization :- It is the result of taking a subset of a higher-level entity set to form a lower-level entity set

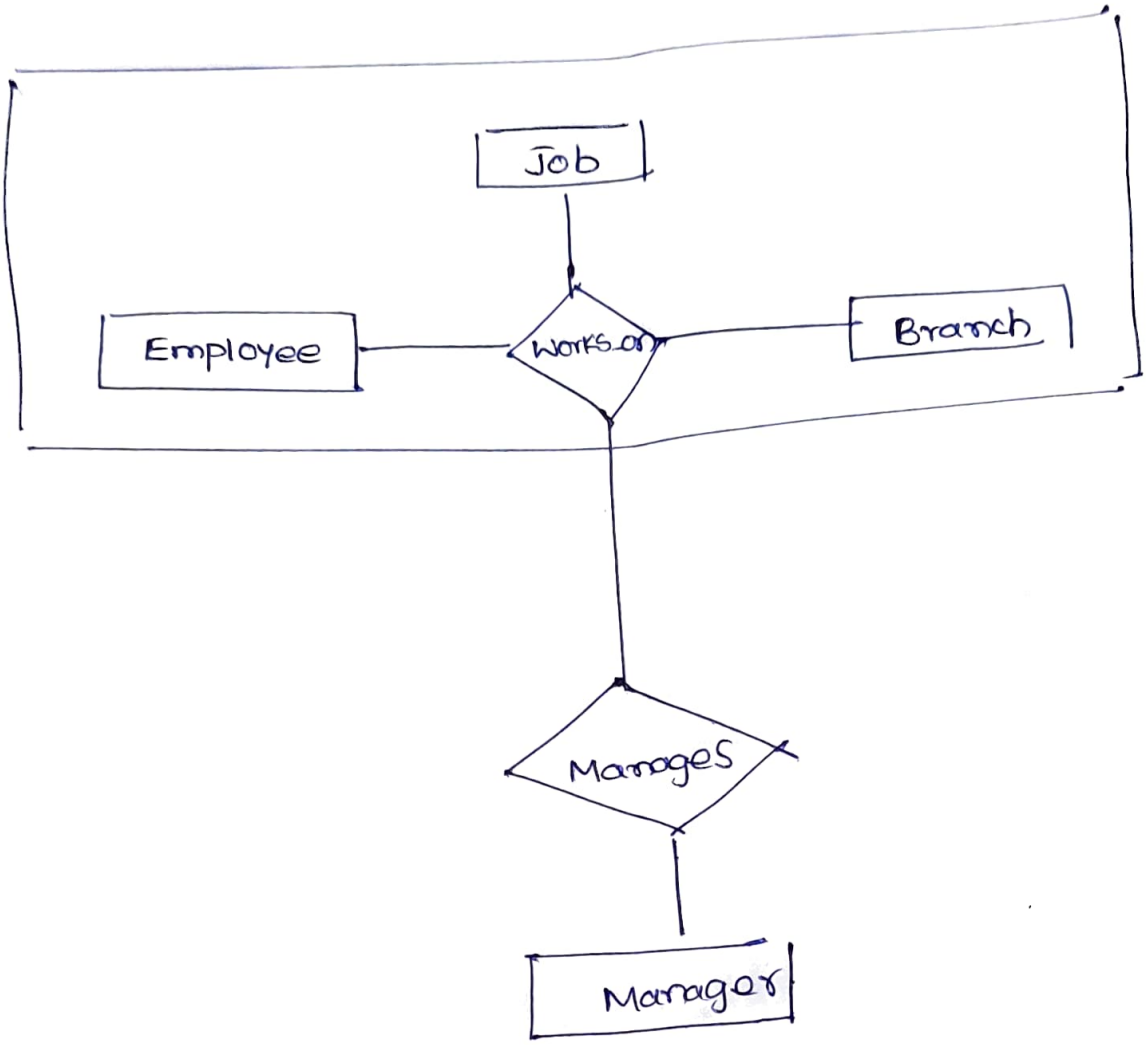
Generalization :- It is the result of taking the union of two or more disjoint lower-level entity sets to produce a higher-level entity set.

Attribute Inheritance :- The attributes of the higher-level entity sets are said to be inherited by the lower-level entity sets.

Aggregation :- It is an abstraction in which relationship sets along with their associated entity sets are treated as higher-level entity sets and can participate in relationships.

Specialization and Generalization.





Aggregation

Normalization

Normalization is the process of eliminating redundancy and maintaining consistency in a relation or a set of relations.

First Normal Form (1NF):-

A relation schema R is in First Normal Form (1NF) if the domains of all attributes of R are atomic.

A domain is atomic if elements of the domain are considered to be indivisible units.

Second Normal Form (2NF):-

A relation schema R is in second Normal form (2NF) if each attribute A in R meets one of the following criteria:

1. It appears in a candidate key
2. It is not partially dependent on a candidate key.

Third Normal Form (3NF):-

A relation schema R is in Third Normal form (3NF) with respect to a set F of functional dependencies if, for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, atleast one of the following holds:

* $\alpha \rightarrow \beta$ is a trivial functional dependency.
* α is a superkey for R
* Each attribute A in $\beta - \alpha$ is contained in a candidate key for R.

Boyce Codd Normal Form (BCNF):-

A relation schema R is in BCNF with respect to a set F of functional dependencies if, for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, atleast one of the following holds:

* $\alpha \rightarrow \beta$ is a trivial functional dependency.
* α is a superkey for schema R.

Fourth Normal Form (4NF):-

A relation schema R is in Fourth Normal Form (4 NF) with respect to a set D of functional and multivalued dependencies if, for all multivalued dependencies in D^+ of the form $\alpha \twoheadrightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, atleast one of the following holds.

* $\alpha \twoheadrightarrow \beta$ is a trivial multivalued dependency.
* α is a superkey for schema R

Functional Dependency :-

Consider a relation schema R and let $\alpha \subseteq R$ and $\beta \subseteq R$. The functional dependency $\alpha \rightarrow \beta$ holds on schema R if, in any legal relation $r(R)$, for all pairs of tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, it is also the case that $t_1[\beta] = t_2[\beta]$.

Supplier

SNo	SName	Scity
S001	Suresh	Hyderabad
S002	Vishnu	Bengaluru
S003	Vijay	Kochi

SNo → Supplier NO

SName → Supplier Name

Scity → Supplier city.

SNo ⟶ SName

The above notation can be read as

Supplier No functionally determines Supplier Name.

or

Supplier Name is functionally dependent on Supplier No.

Uses

1. To test relations to see whether they are legal under a given set of functional dependencies.
2. To specify constraints on the set of legal relations.

Trivial Functional Dependency :-

A functional dependency of the form $\alpha \rightarrow \beta$ is trivial if $\beta \subseteq \alpha$.

E.g.

$$\{ SNO, SName \} \longrightarrow SNO$$

$$\alpha \longrightarrow \beta$$

$$\beta \subseteq \alpha$$

Functional dependencies are also known as equality - generating dependencies

Multivalued Dependency

Let R be a relation schema and let $\alpha \subseteq R$ and $\beta \subseteq R$. The multivalued dependency $\alpha \twoheadrightarrow \beta$ holds on R if, in any legal relation r(R), for all pairs of tuples t_1 and t_2 in r such that $t_1[\alpha] = t_2[\alpha]$, there exist tuples t_3 and t_4 in r such that

$$t_1[\alpha] = t_2[\alpha] = t_3[\alpha] = t_4[\alpha]$$

$$t_1[\beta] = t_3[\beta]$$

$$t_2[\beta] = t_4[\beta]$$

$$t_3[R - \alpha - \beta] = t_4[R - \alpha - \beta]$$

$$t_4[R - \alpha - \beta] = t_1[R - \alpha - \beta]$$

	α	β	$R-\alpha-\beta$
t_1	a_1	b_1	c_1
t_2	a_1	b_2	c_1
t_3	a_1	b_1	c_2
t_4	a_1	b_2	c_2

Subject	Teacher	Textbook
OOPJ	Suresh	T1
OOPJ	vishnu	T1
OOPJ	Suresh	T2
OOPJ	vishnu	T2

E.g.

Subject $\longrightarrow\!\!>$ Teacher.

Trivial Multivalued Dependency

A multivalued dependency $\alpha \longrightarrow\!\!> \beta$ is trivial if $\beta \subseteq \alpha$ or $\alpha \cup \beta = R$.

Join Dependency

A join dependency is a constraint which specifies that the relation concerned is a join of specific number of projections.

Decomposition of a Relation

The process of breaking up or dividing a single relation into two or more sub relations is called as decomposition.

Types of Decomposition

Decomposition can be classified into two types.

1. Lossless Join Decomposition
2. Lossy Join Decomposition

Lossless Join Decomposition

Consider that a relation R is divided into $R_1, R_2 \dots R_n$ sub relations.

The decomposition of the relation is said to be lossless when the join of the sub relations yield the same relation that was decomposed.

$$R = R_1 \bowtie R_2 \bowtie \dots \bowtie R_n$$

Lossy Join Decomposition

Consider a relation R which is decomposed into sub relations $R_1, R_2 \dots R_n$.

The decomposition of the relation is said to be lossy when the join of the sub relations does not yield the same relation that was decomposed.

$$R_1 \bowtie R_2 \bowtie \dots \bowtie R_n \supset R$$

E.g. Consider the relation R(A,B,C)

A	B	C
1	2	1
2	5	3
3	3	3

The relation R is decomposed into two sub relations $R_1(A,C)$ and $R_2(B,C)$

A	C
1	1
2	3
3	3

R_1

B	C
2	1
5	3
3	3

R_2

If we perform the natural join ($\bowtie$) of the sub relations R_1 and R_2, we get

E.g. Consider the relation R(A,B,C)

A	B	C
1	2	1
2	5	3
3	3	3

The relation R is decomposed into two sub relations $R_1(A,B)$ and $R_2(B,C)$

A	B
1	2
2	5
3	3

$R_1(A,B)$

B	C
2	1
5	3
3	3

$R_2(B,C)$

If we perform the natural join($\bowtie$) of the sub relations R_1 and R_2, we get

A	B	C
1	2	1
2	5	3
3	3	3

This relation is same as the original relation R

A	B	C
1	2	1
2	5	3
2	3	3
3	5	3
3	3	3

$R_1 \bowtie R_2 \supset R$

Properties of Decomposition

* Lossless Decomposition
* Dependency preservation.

Lossless decomposition ensures

* No information is lost from the original relation during decomposition.
* When the subrelations are joined back, the same relation is obtained that was decomposed.

Dependency preservation ensures

* None of the functional dependencies that holds on the original relation are lost.
* The sub relations still hold or satisfy the functional dependencies of the original relation.

5NF :- A relation R is said to be in 5NF if and only if it is in 4NF and no join dependency exists.

BCNF (Boyce-Codd Normal Form)

A relation schema R is in BCNF with respect to a set F of functional dependencies if, for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, atleast one of the following holds:

1. $\alpha \rightarrow \beta$ is a trivial functional dependency
2. α is a superkey for schema R.

BCNF Decomposition Algorithm

```
result := { R } ;
done := false ;
compute F+ ;
While ( not done) do
   if ( there is a schema Ri in result that is not
                              in BCNF ) then

   begin
      let α → β be a non trivial functional dependency
      that holds on Ri such that α → Ri is not in F+
      and α ∩ β = ϕ ;
      result := (result - Ri) U ( Ri - β) U (α, β);
   end
   else
      done := true;
```

3 NF (Third Normal form)

A relation schema R is in 3NF with respect to a set F of functional dependencies if, for all functional dependencies in F^+ of the form $\alpha \rightarrow \beta$, where $\alpha \subseteq R$ and $\beta \subseteq R$, atleast one of the following holds

1. $\alpha \rightarrow \beta$ is a trivial functional dependency.
2. α is a superkey for R
3. Each attribute A in $\beta - \alpha$ is contained in a candidate key for R

3NF Decomposition Algorithm

```
let Fc be a canonical cover for F;
i := 0;
for each functional dependency α → β in Fc do
    if none of the schemas Rj, j = 1,2 --- i contains αβ
    then
        begin
            i := i+1;
            Ri := αβ;
        end
    if none of the schemas Rj, j = 1,2 --- i contains a
    candidate key for R then
    begin
            i := i+1;
            Ri := any candidate key for R;
    end
return (R1, R2 - - Ri)
```

Differences between 3NF and BCNF

Parameters	3NF	BCNF
Strength	3NF is less strong than that of BCNF	BCNF is more strong than 3NF
Functional dependencies	The functional dependencies in 3NF already exist in 2NF and 1NF	The functional dependencies in BCNF already exist in 3NF, 2NF and 1NF
Redundancy	3NF has higher redundancy	BCNF has lower redundancy.
preservation	In 3NF, preservation occurs for all functional dependencies.	In BCNF, there is no preservation for all functional dependencies.
Decomposition	Lossless decomposition is easy to achieve	Lossless decomposition is hard to achieve

Closure of a set of functional dependencies (F^+)

Let F be a set of functional dependencies. The closure of F, denoted by F^+, is the set of all functional dependencies logically implied by F.

closure of a set of attributes (α^+)

Let α be a set of attributes and F be a set of functional dependencies. the closure of α under F is the set of all attributes functionally determined by α under F. and it is denoted by α^+.

Canonical cover of a set of Functional dependencies (F_c)

A canonical cover F_c of F is a set of functional dependencies such that F logically implies all dependencies in F_c and F_c logically implies all dependencies in F.

F_c must have the following properties.

* NO functional dependency in F_c contains an extraneous attribute.
* Each left side of a functional dependency in F_c is unique.

Restriction of F to R_i

Let F be a set of functional dependencies on schema R and let $R_1, R_2 \dots R_n$ be a decomposition of R. The restriction of F to R_i is the set F_i of all functional dependencies in F^+ that include only attributes of R_i

Armstrong's Axioms

1. **Reflexivity Rule**:

   If α is a set of attributes and $\beta \subseteq \alpha$ then $\alpha \rightarrow \beta$ holds

2. **Augmentation Rule**:

   If $\alpha \rightarrow \beta$ holds and γ is a set of attributes, then $\gamma\alpha \rightarrow \gamma\beta$ holds.

3. **Transitivity Rule**:-

   If $\alpha \rightarrow \beta$ holds and $\beta \rightarrow \gamma$ holds then $\alpha \rightarrow \gamma$ holds.

These rules are sound and complete.

A procedure to compute F^+

① $F^+ = F$

② repeat

ⓐ for each functional dependency f in F^+

apply reflexivity and augmentation rules on f and add the resulting functional dependencies to F^+.

ⓑ for each pair of functional dependencies f_1 and f_2 in F^+

if f_1 and f_2 can be combined using transitivity, add the resulting functional dependency to F^+

until F^+ does not change any further.

An algorithm to compute α^+

① result := α

② while (changes to result) do

ⓐ for each functional dependency $\beta \rightarrow \gamma$ in F do

```
begin
    if β ⊆ result then
        result := result ∪ γ;
end
```

Computing Canonical Cover

1. $F_c = F$
2. repeat
   a. Use the union rule to replace any dependencies in F_c of the form $\alpha_1 \rightarrow \beta_1$ and $\alpha_1 \rightarrow \beta_2$ with $\alpha_1 \rightarrow \beta_1 \beta_2$

   b. Find a functional dependency $\alpha \rightarrow \beta$ in F_c with an extraneous attribute either in α or in β

   If an extraneous attribute is found, delete it from $\alpha \rightarrow \beta$

   until F_c does not change

Testing for Dependency Preservation

1. compute F^+
2. for each schema R_i in D do
   begin
   F_i := the restriction of F^+ to R_i
   end
3. $F' = \phi$;
4. for each restriction F_i do
   begin
   $F' = F' \cup F_i$
   end
   compute F'^+;
   if ($F'^+ = F^+$) then
   return(true);
   else
   return(false);