# R. C. Patel Institute of Technology, Shirpur

*Department of Computer Engineering*


## Sub: Database Management System(RCP23CCPC303)

**by**

**Prof. J. S. Sonawane**

# Unit I

# Introduction to DBMS

# What is a Database?

- **Data**: Facts, figures, statistics etc. having no particular meaning (e.g. 1, ABC, 19 etc).
- **Record**: Collection of related data items, e.g. in the above example the three data items had no meaning. But if we organize them in the following way, then they collectively represent meaningful information.

| Roll | Name | Age |
|---|---|---|
| 1 | ABC | 19 |

- **Table** or **Relation**: Collection of related records.

| Roll | Name | Age |
|---|---|---|
| 1 | ABC | 19 |
| 2 | DEF | 22 |
| 3 | XYZ | 28 |

The columns of this relation are called **Fields**, **Attributes** or **Domains**. The rows are called **Tuples** or **Records**.

- **Database**: Collection of related relations. Consider the following collection of tables:

**T1**

| Roll | Name | Age |
|---|---|---|
| 1 | ABC | 19 |
| 2 | DEF | 22 |
| 3 | XYZ | 28 |

**T2**

| Roll | Address |
|---|---|
| 1 | KOL |
| 2 | DEL |
| 3 | MUM |

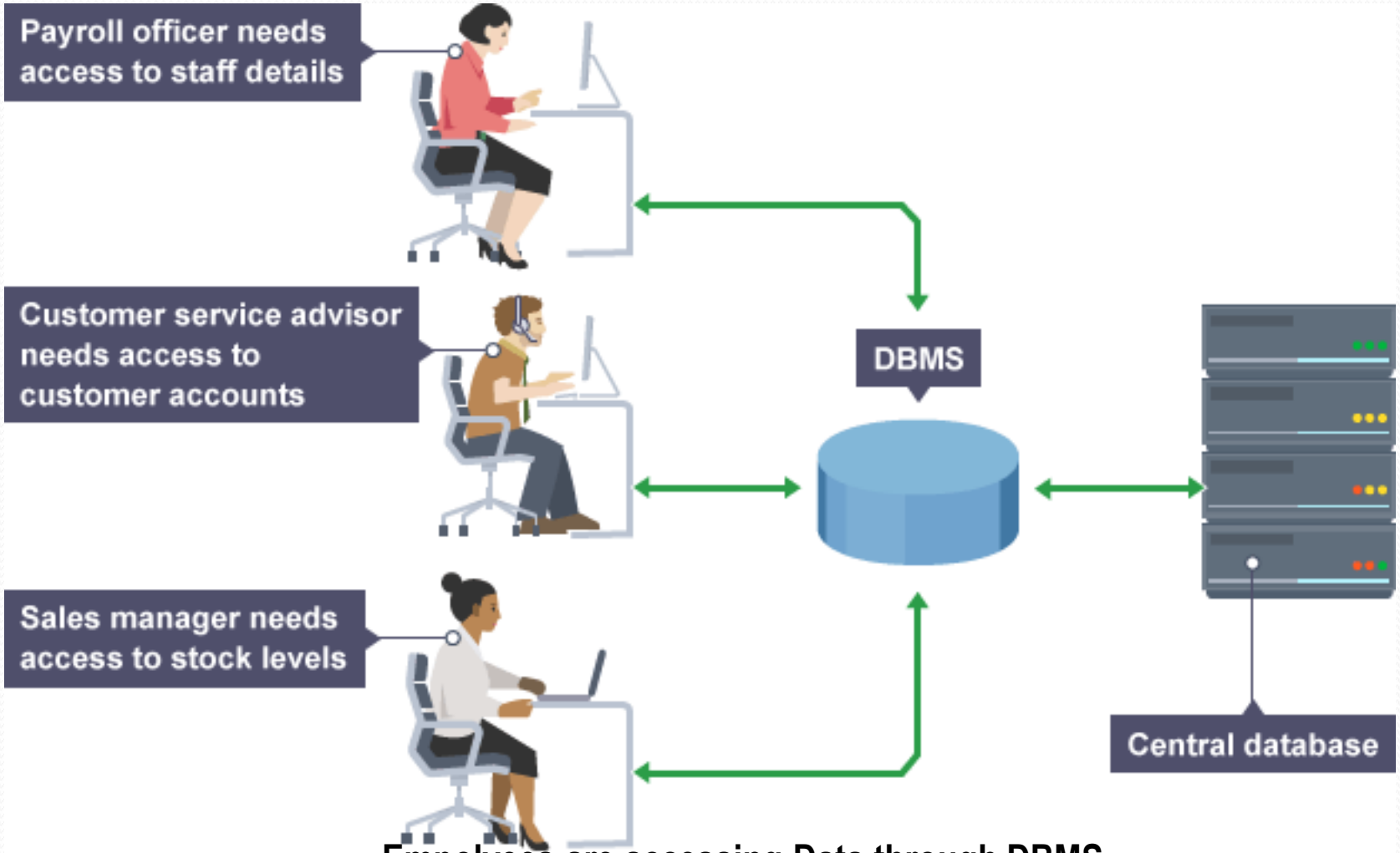
**T3**

| Roll | Year |
|---|---|
| 1 | I |
| 2 | II |
| 3 | I |

**T4**

| Year | Hostel |
|---|---|
| I | H1 |
| II | H2 |

- A database in a DBMS could be viewed by lots of different people with different responsibilities.




Empolyees are accessing Data through DBMS

# Introduction Database Concepts

- A database-management system (DBMS) is a collection of interrelated data and a set of programs to access those data.
- The collection of data, usually referred to as the database, contains information relevant to an enterprise.
- The primary goal of a DBMS is to provide a way to store and retrieve database information that is both convenient and efficient.

# Features of Database System

- Organizes data efficiently
- Reduces redundancy
- Ensures data consistency
- Provides data security and controlled access
- Supports concurrent access

# The Entity-Relationship (ER) Model:

- The **entity-relationship** (**E-R**) data model was developed to facilitate database design by allowing specification of an *enterprise schema* that represents the overall logical structure of a database.
- The E-R model is very useful in mapping the meanings and interactions of real-world enterprises onto a conceptual schema. Because of this usefulness, many database-design tools draw on concepts from the E-R model.
- The E-R data model employs three basic concepts: entity sets, relationship sets, and attributes. The E-R model also has an associated diagrammatic representation.

# Entity-Relationship (ER) Model

- The Entity-Relationship (E-R) data model is used in database design.

It was developed to facilitate the process of designing databases.

- The model allows for the specification of an enterprise schema.
- The enterprise schema represents the overall logical structure of a database.
- The E-R model is very useful in mapping the meanings and interactions of real world enterprises onto a conceptual schema.
- The E-R data model is based on three basic concepts:
  - **Entity sets**
  - **Relationship sets**
  - **Attributes**
- The E-R model includes a diagrammatic representation known as the **E-R diagram.**
- An **E-R diagram** can graphically express the overall logical structure of a database.

# Entity

- An entity is a “thing” or “object” in the real world that is distinguishable from all other objects.
- **Examples:** student, course, employee An entity has a set of properties.
- The values for some set of properties must uniquely identify an entity.
- An entity may be concrete (Physical), such as a person or a student, or it may be abstract(Conceptual), such as a course.
- Entities are represented as **rectangles** in ER diagrams.

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

  1. All students in a college form a **student entity set**.
  2. All employees in an organization form a **employee entity set**
- Entity sets do not need to be disjoint. For example, it is possible to define the entity set person consisting of all people in a university. A person entity may be an faculty entity, a student entity, both, or neither.

- **Attributes**
- Attributes are the properties or characteristics of an entity.
  - **Example:** For a student entity, attributes might include student ID, name, and age.
  - Attributes are descriptive properties possessed by each member of an entity set.
  - The designation of an attribute for an entity set expresses that the database stores similar information concerning each entity in the entity set.
- Each entity may have its own value for each attribute
- Each entity has a value for each of its attributes for example particular student entity may have value 101 for student ID, ”Rahul” for Name and 20 for age attribute.
- Attributes are represented as **Oval** in ER diagrams.

# Entity Example

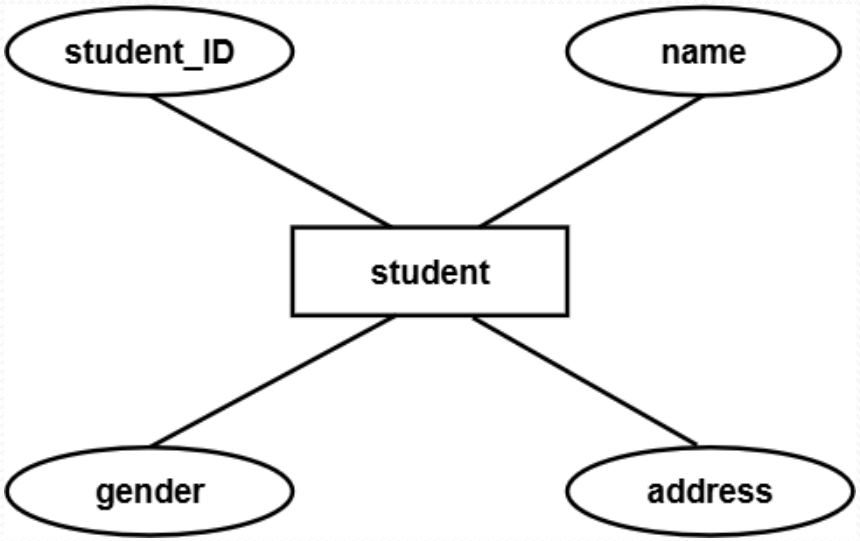



Figure: 1: E-R diagram showing entity Student

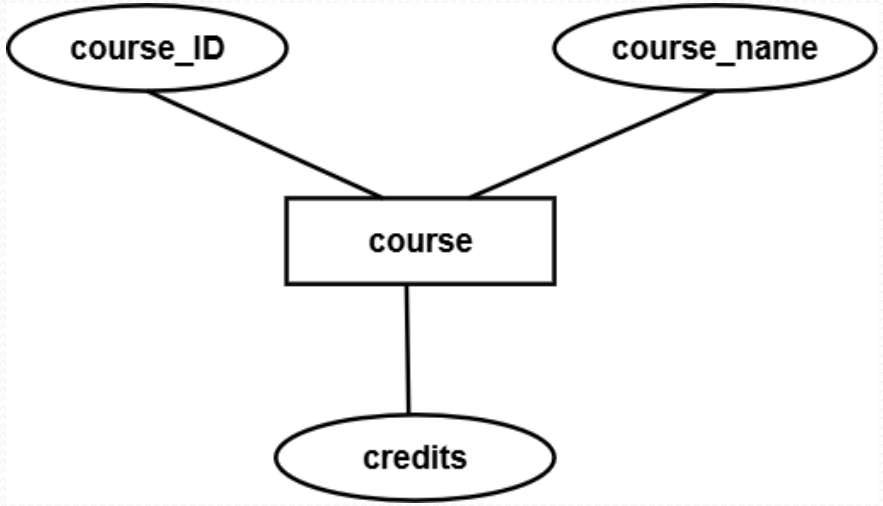



Figure: 2: E-R diagram showing entity Course

- In above ER diagram(Figure 1) **student** is entity and **student_ID**, **name**, **gender** and **address** are attributes
- In above ER diagram(Figure 2) **course** is entity and **course_ID**, **course_name** and **credits** are attributes

# Types of Attributes

Following are the types of attributes:

- **Simple (Atomic) Attribute:** Cannot be divided further (e.g., student ID)
- **Composite Attribute:** Can be divided into sub-parts (e.g., name = first name + last name)
- **Derived Attribute:** Can be derived from other attributes (e.g., age from DOB)
- **Multivalued Attribute:** Can have multiple values (e.g., phone numbers

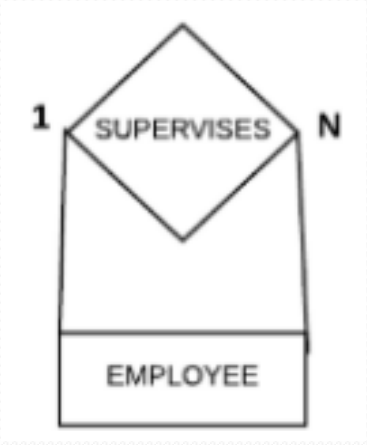
# Degree of a Relationship

- **The number of entity types which took part in the entity relationship** is called the degree of relationships.
- three different types of degree of relationships,

1. Unary relationship
2. Binary relationship
3. Ternary relationship
4. N-ary relationship

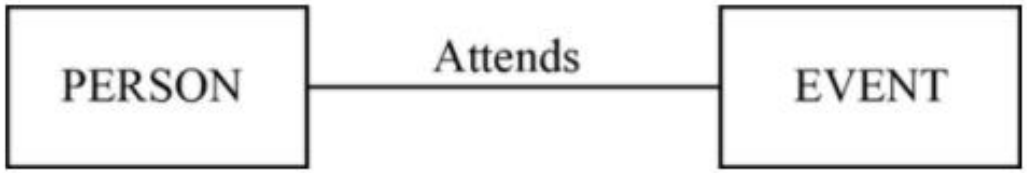
# Unary relationship

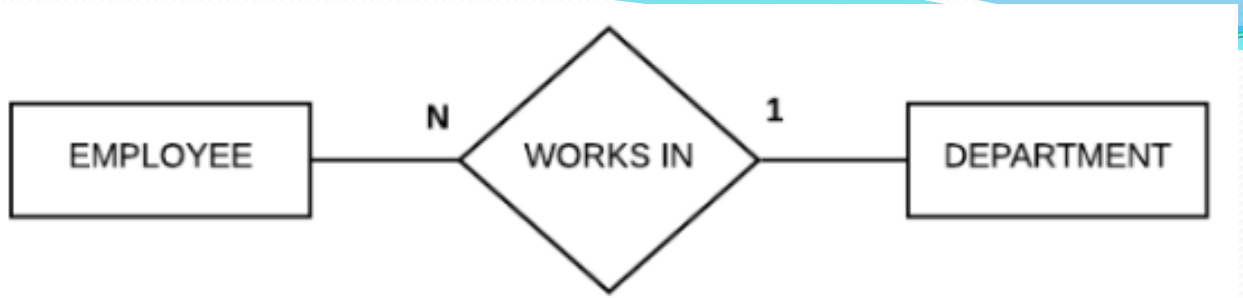
- It is the relationship between the instances of a single entity type. It is also called a recursive relationship.
- **Two entities are of the same type**
- In a particular class, we have many students, there are monitors too. So, here class monitors are also students. Thus, we can say that only students are participating here. So the degree of such type of relationship is 1.

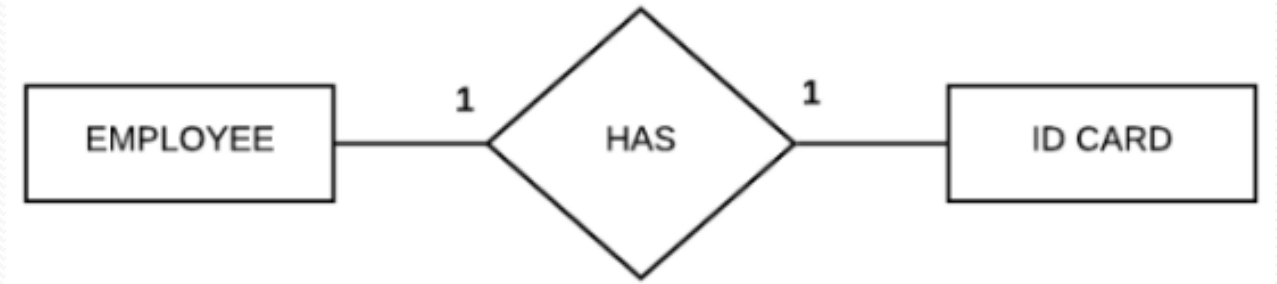



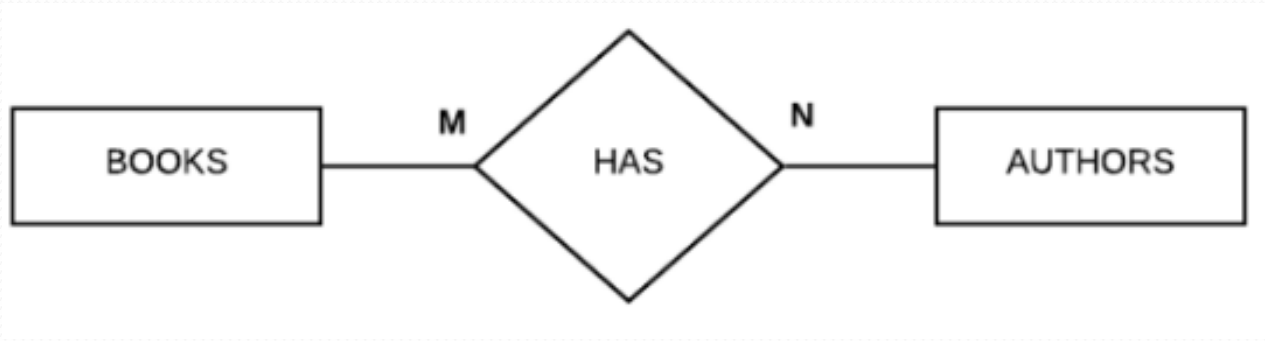
# Binary relationship

- It is the relationship between the instances of two different entity types. Two entities will participate in the relationship.
- **Relationship between two different entities**
- Person and events are two different entity types which are related by using the relationship called “attends”.



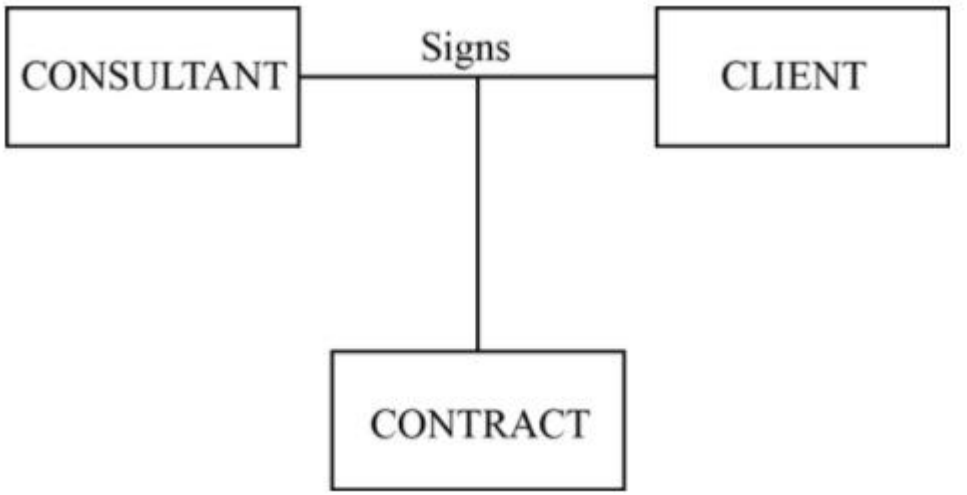

EMPLOYEE
N
WORKS IN
1
DEPARTMENT

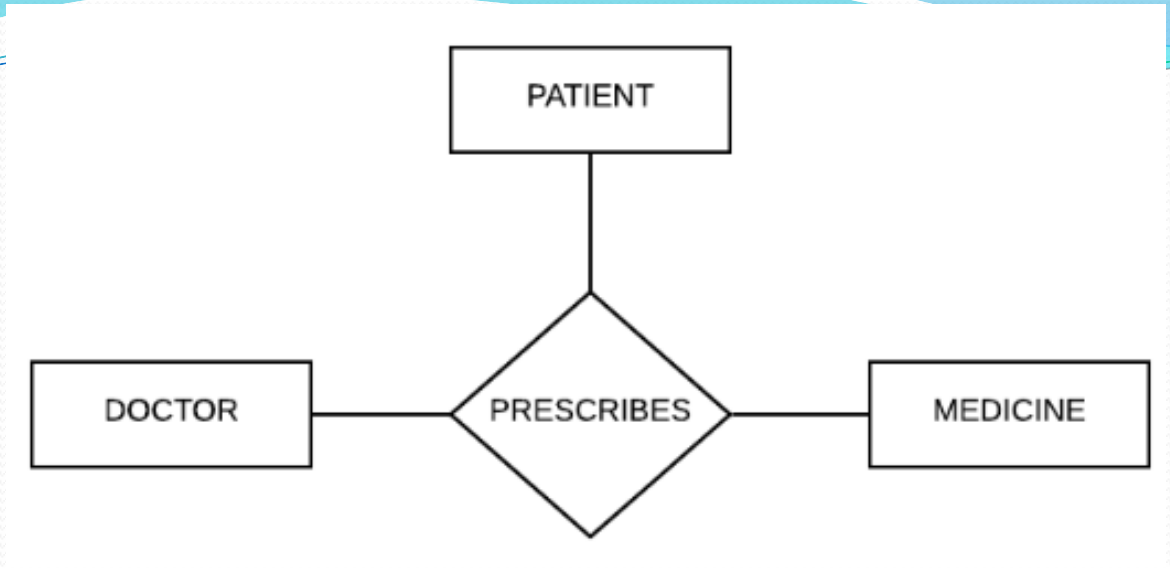

EMPLOYEE
1
HAS
1
ID CARD

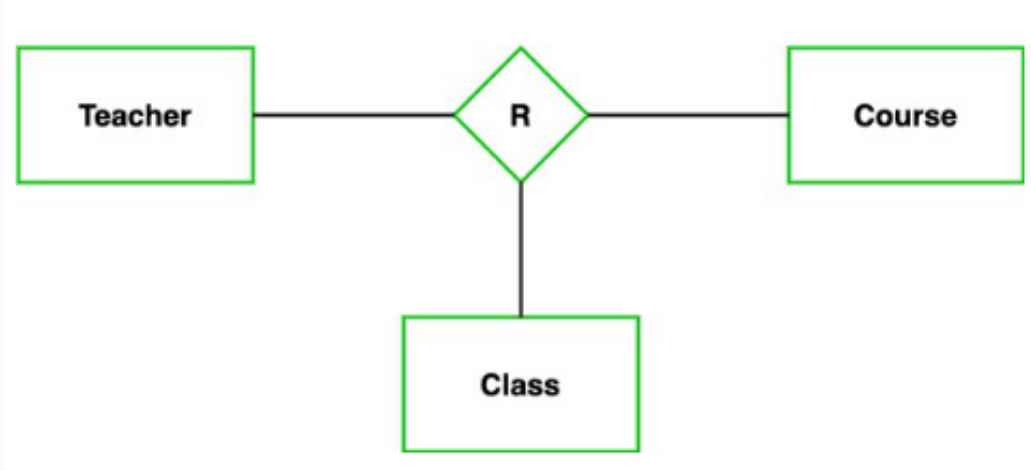

BOOKS
M
HAS
N
AUTHORS

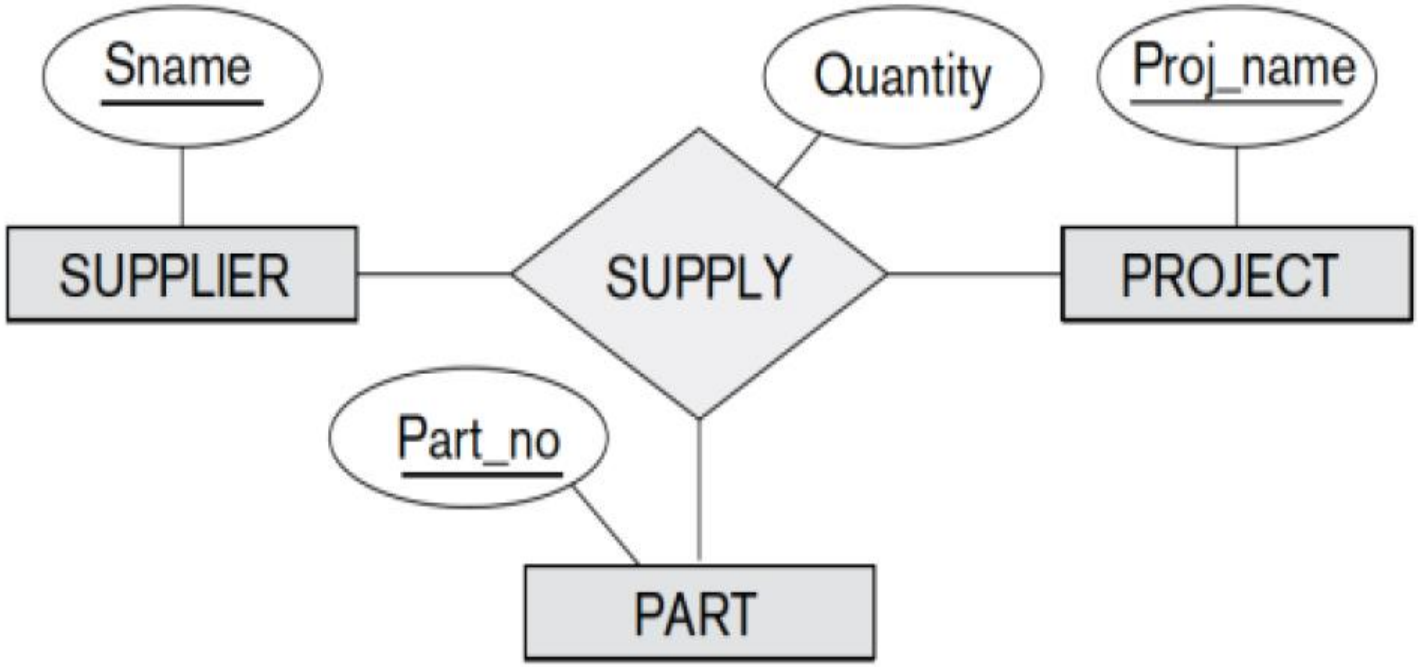
# Ternary relationship

- A simultaneous relationship between the instances of three entity types with unique attributes is called a ternary relationship.
- **relationship between three different entities**
- Consultant, client and contract are three different entities with different attributes.
- These three entities are related with a single relationship called “signs”.

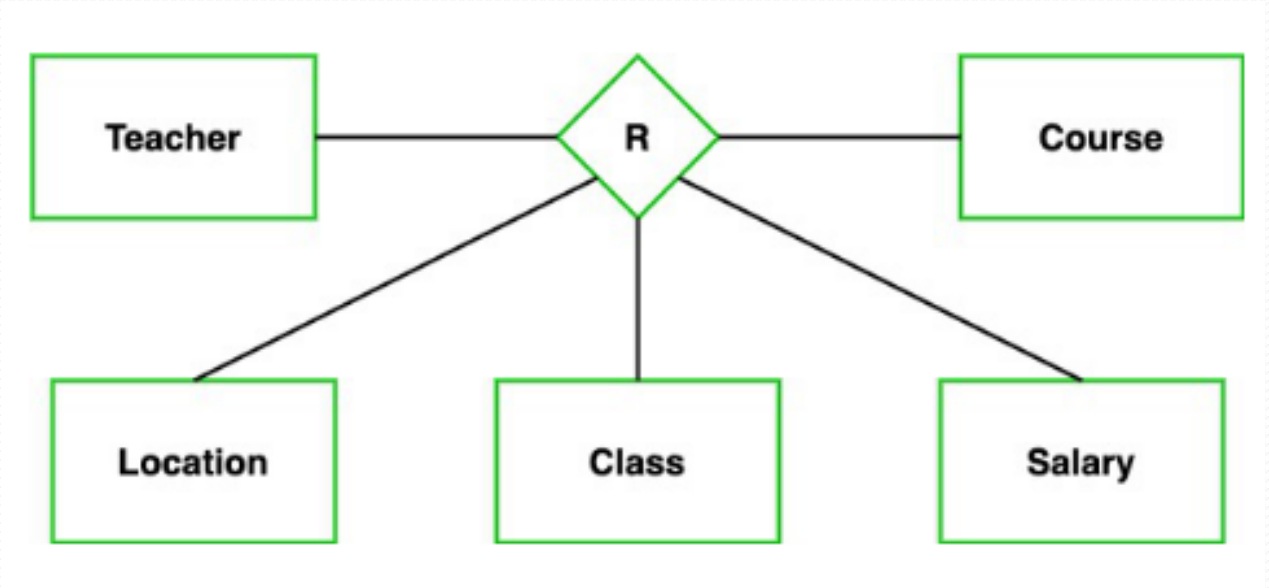



PATIENT
DOCTOR
PRESCRIBES
MEDICINE


Teacher
R
Course
Class


Sname
Quantity
Proj_name
SUPPLIER
SUPPLY
PROJECT
Part_no
PART

# N-ary Relationship

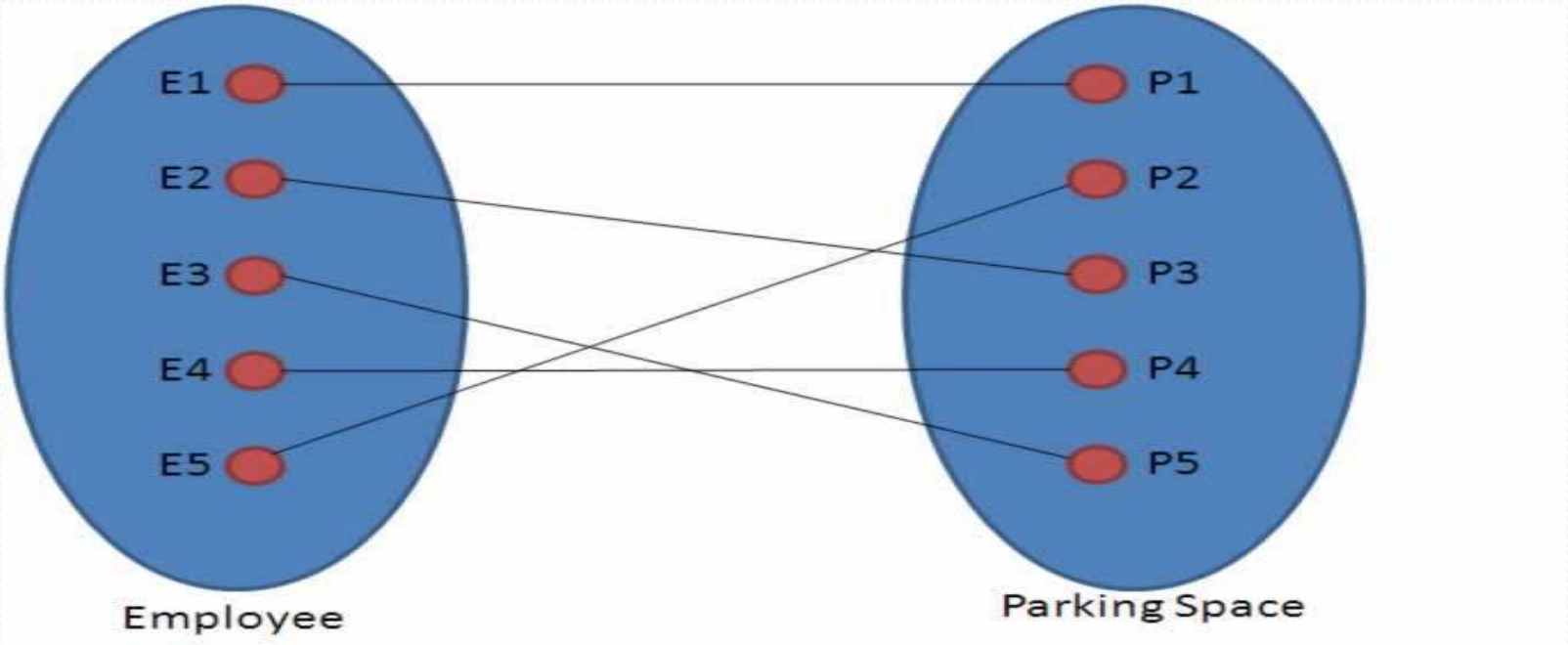


# Mapping Cardinalities

- Express the number of entities to which another entity can be associated via a relationship set.
- Most useful in describing binary relationship sets.
- **For a binary relationship** set the mapping cardinality must be one of the following types:
  - One to one
  - One to many
  - Many to one
  - Many to many

# One-to-one

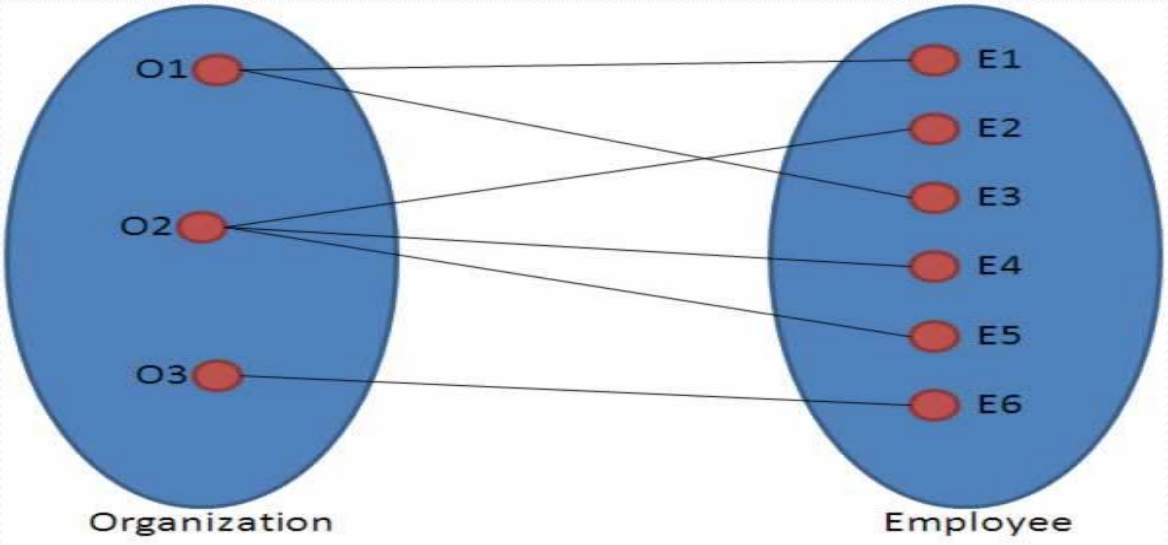
- One entity from entity set A can be associated with **at most one** entity of entity set B and vice versa.



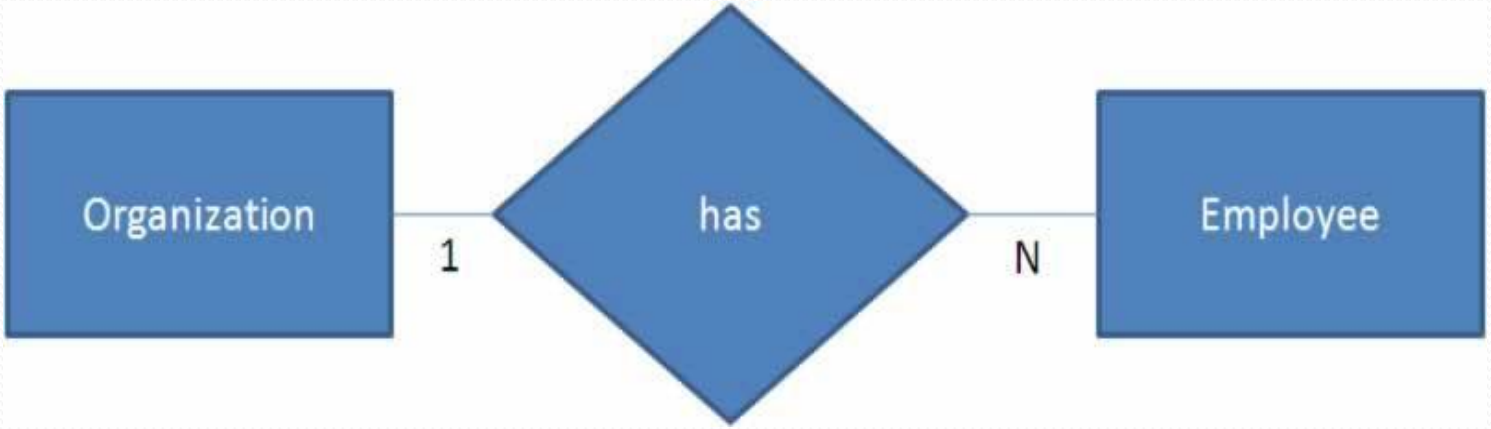
- One employee is assigned with only one parking space and one parking space is assigned to only one employee. Hence it is a 1:1 relationship and cardinality is One-To-One (1:1).

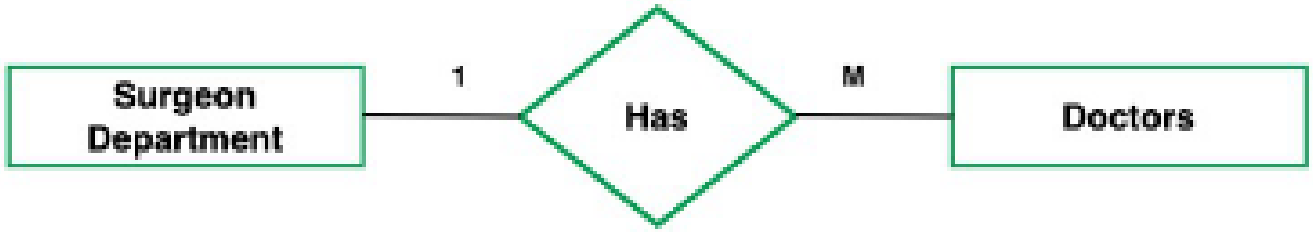
# One-to-many

- One entity from entity set A can be associated with more than one entities of entity set B however an entity from entity set B, can be associated with at most one entity.



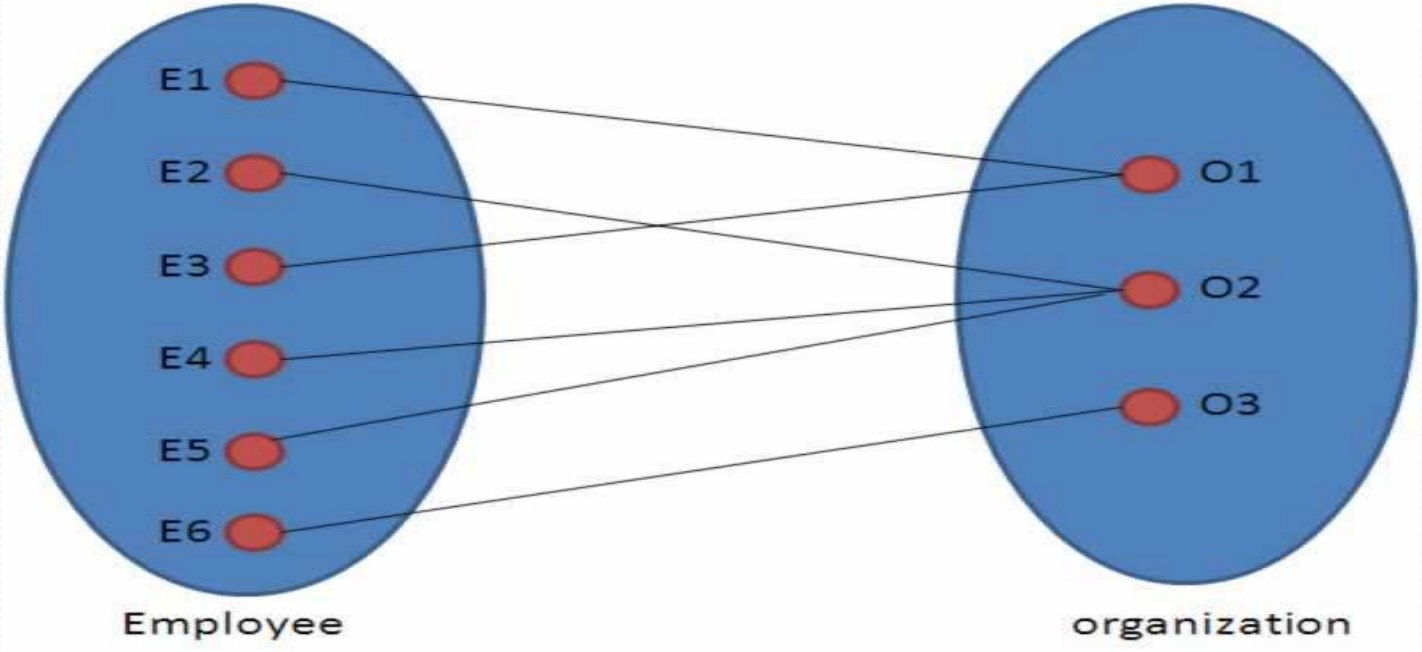
- One organization can have many employees , but one employee works in only one organization. Hence it is a 1:N relationship and cardinality is One-To-Many (1:N).



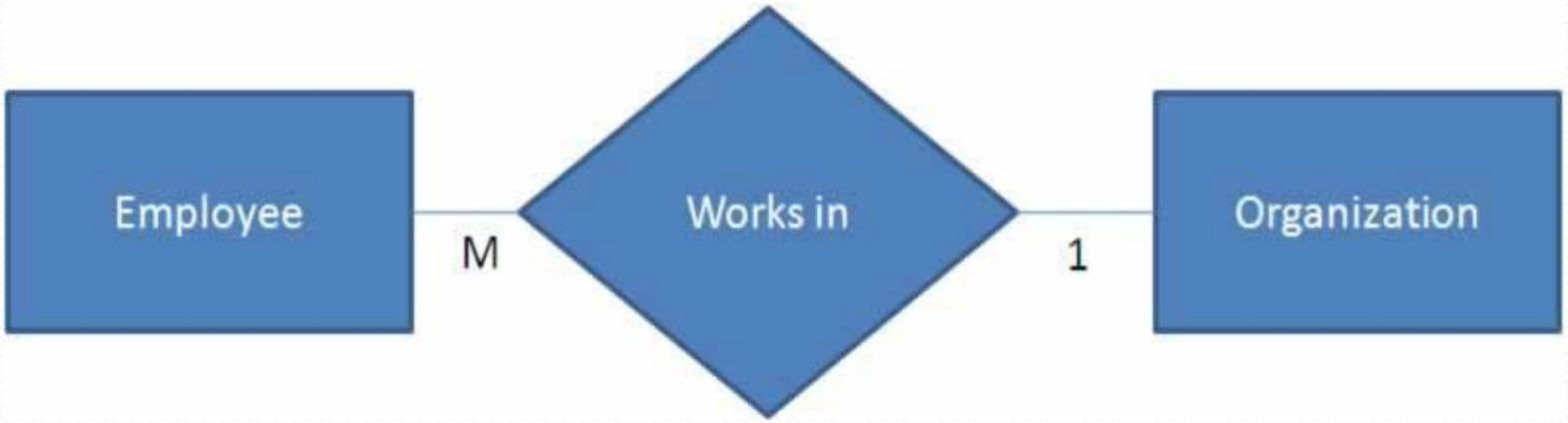



# Many-to-one

- More than one entities from entity set A can be associated with at most one entity of entity set B, however an entity from entity set B can be associated with more than one entity from entity set A.

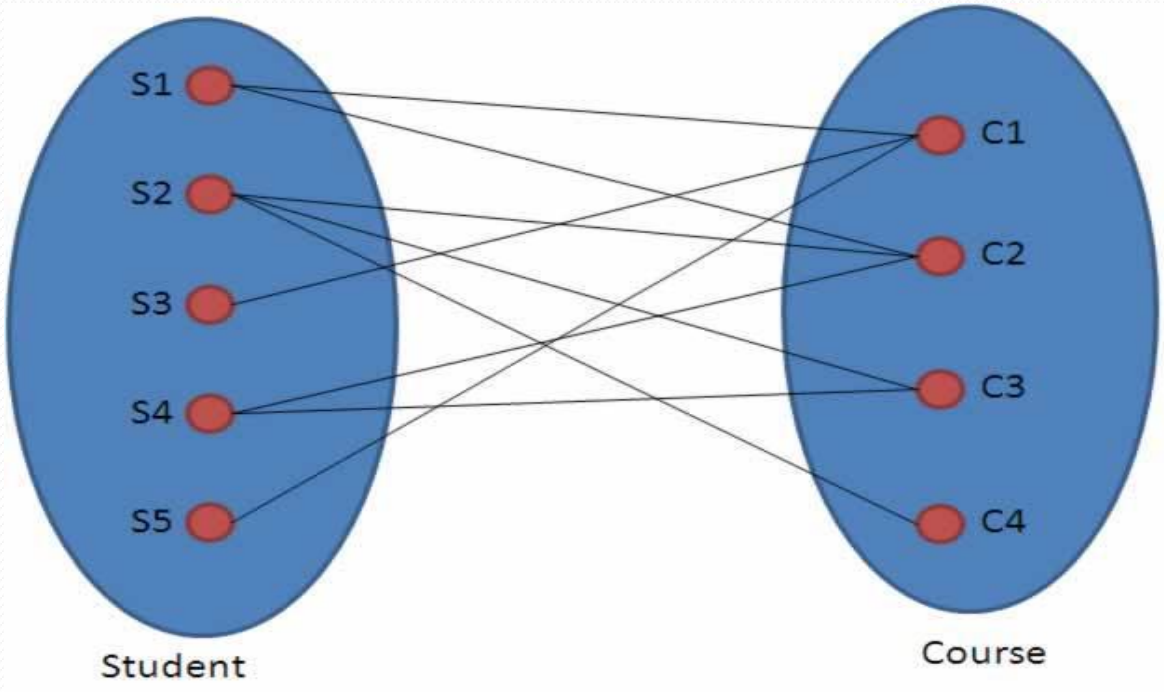


- One employee works in only one organization But one organization can have many employees. Hence it is a M:1 relationship and cardinality is Many-to-One (M :1).

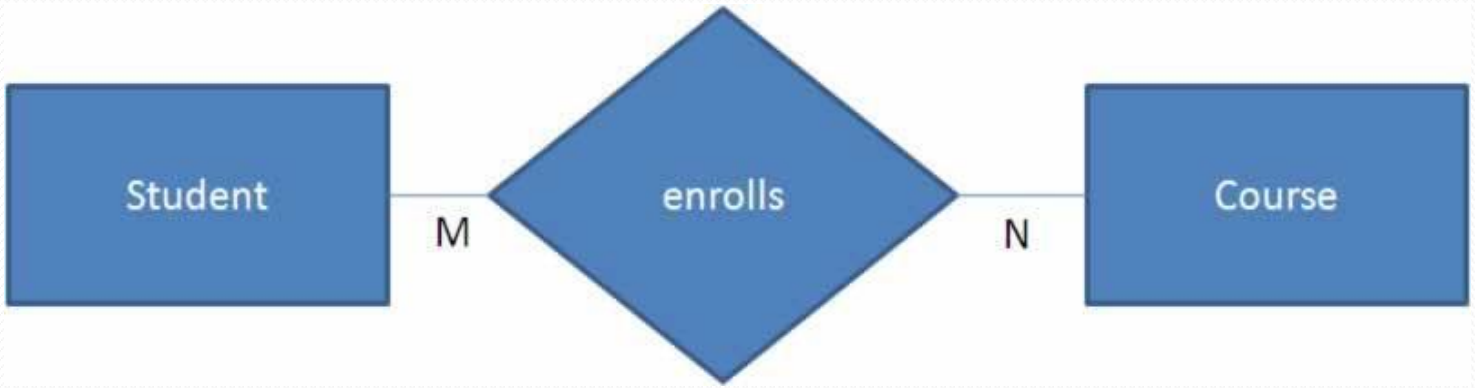


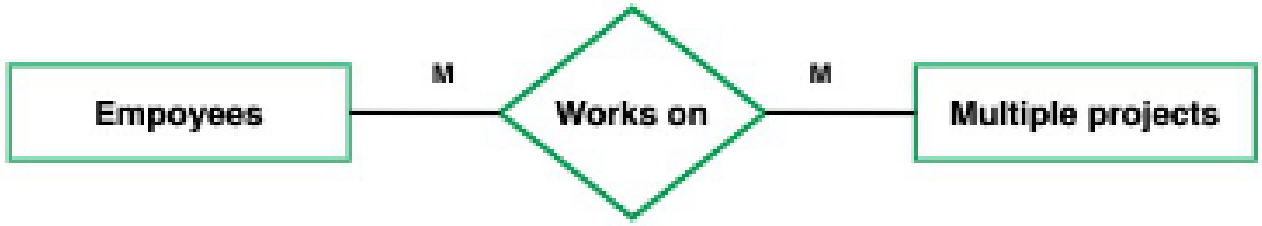

# Many-to-Many (M:N)

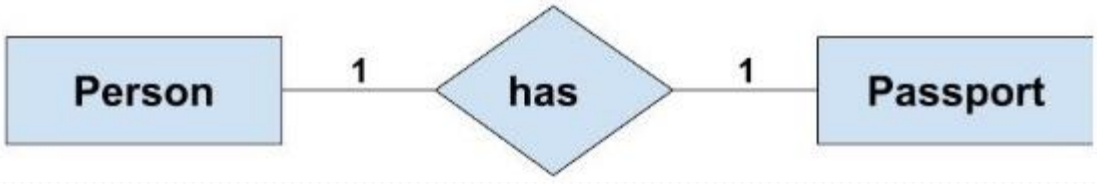
- One entity from A can be associated with more than one entity from B and vice versa.



- One student can enroll for many courses and one course can be enrolled by many students. Hence it is a M:N relationship and cardinality is Many-to-Many (M:N).




Person
1
has
1
Passport


Customer
1
has
n
Account


Customer
m
buys
n
Product

# Participation Constraint / Minimum cardinality constraint

- In a Relationship, Participation constraint specifies the presence of an entity when it is related to another entity in a relationship type.
- This constraint **specifies the number of instances of an entity that are participating in the relationship type.**

•There are two types of Participation constraint:

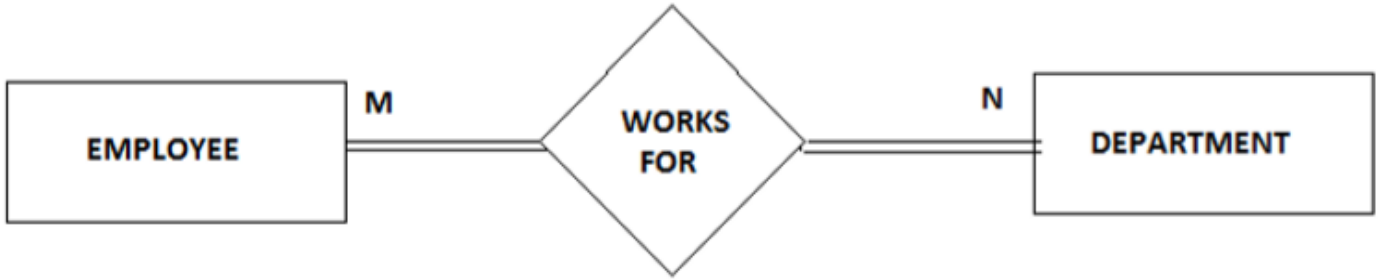
- •Total participation
- •Partial participation

# Participation Constraint

**1. Total Participation –** Each entity in the entity set **must participate** in the relationship. If each student must enroll in a course, the participation of student will be total. Total participation is shown by double line in ER diagram.

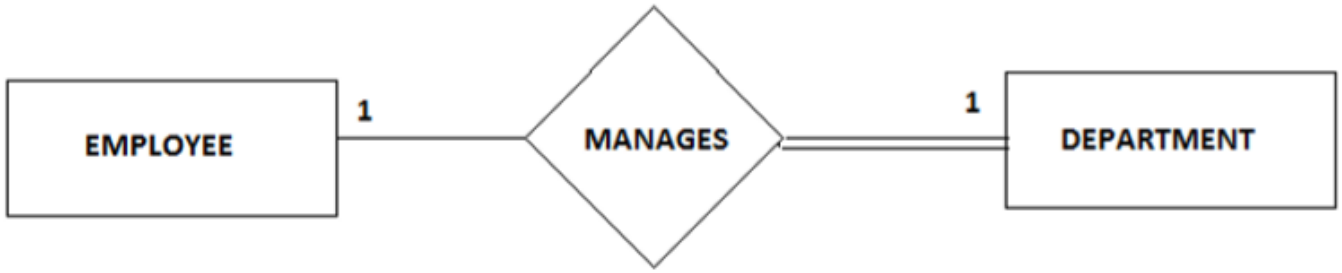
**2. Partial Participation –** The entity in the entity set **may or may NOT participat**e in the relationship. If some courses are not enrolled by any of the student, the participation of course will be partial.

# Total Participation

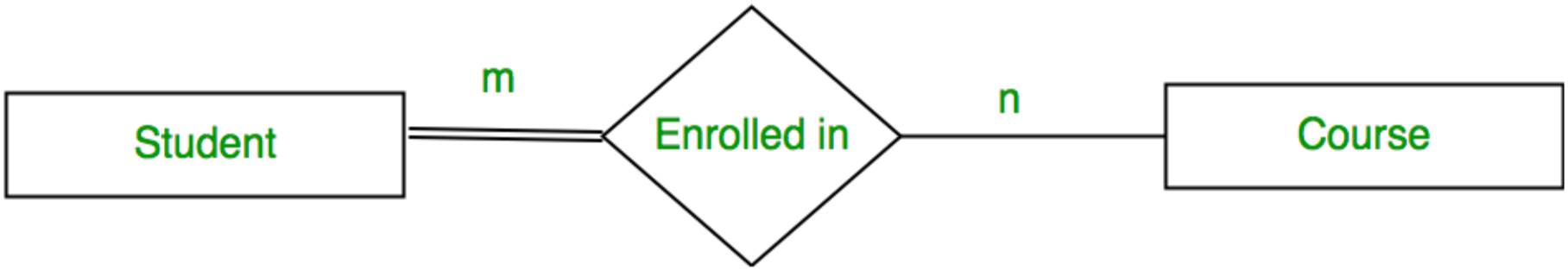



- Each entity in the entity set is involved in at least one relationship in a relationship set i.e. the number of relationship in every entity is involved is greater than 0.
- Consider two entities Employee and Department related via Works_For relationship. Now, every Employee works in at least one department therefore an Employee entity exist if it has at least one Works_For relationship with Department entity. Thus the participation of Employee in Works_For is total relationship. Total Participation is **represented by double line** in ER diagram.

# Partial Participation

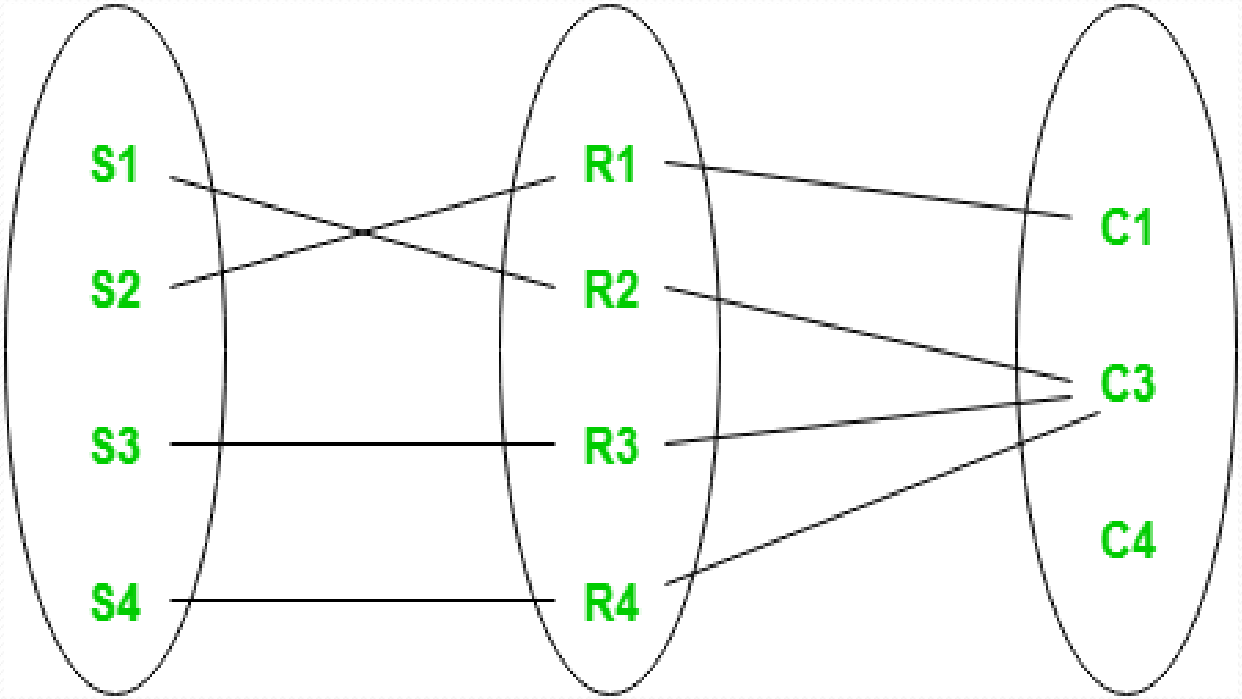



- Each entity in entity set may or may not occur in at least one relationship in a relationship set.
- For example: Consider two entities Employee and Department and they are related to each other via Manages relationship. An Employee must manage a Department, he or she could be the head of the department. But not every Employee in the company manages the department. So, participation of employee in the Manages relationship type is partial i.e. only a particular set of Employees will manage the Department but not all.




- The diagram depicts the ‘Enrolled in’ relationship set with Student Entity set having total participation and Course Entity set having partial participation.

- Using set, it can be represented as,




Every student in Student Entity set is participating in relationship but there exists a course C4 which is not taking part in the relationship.

# Extended Entity-Relationship (EER) Model:

- EER is a high-level data model that incorporates the extensions to the original ER model. Enhanced ERD are high level models that represent the requirements and complexities of complex database.

- In addition to ER model concepts EER includes −
- Specialization
- Generalization.
- Aggregation.

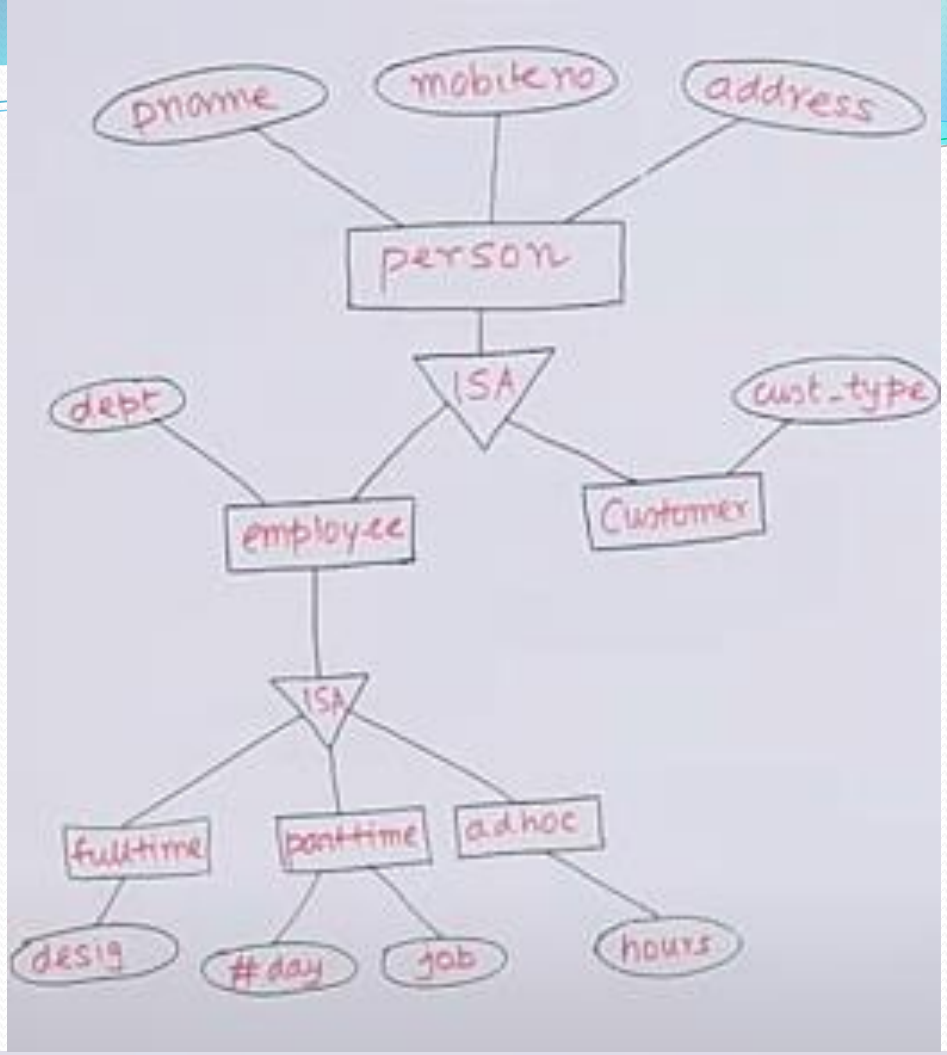
These concepts are used to create EER diagram

# Features of EER Model

1. EER creates a design more accurate to database schemas.
2. It reflects the data properties and constraints more precisely.
3. It includes all modeling concepts of the ER model.
4. Diagrammatic technique helps for displaying the EER schema.
5. It includes the concept of specialization and generalization.
6. It is used to represent a collection of objects that is union of objects of different of different entity types.

**Superclass:** An entity type that represents a general concept at a high level.

**Subclass:** An entity type that represents a specific concept at lower levels.




Customer (pname, mobileno, address, cust_type)

fulltime (pname, mobileno, address, dept, desig)

parttime (pname, mobileno, address, dept, #day, job)

adhoc (pname, mobileno, address, dept, hours)

# Specialization

- Specialization is a process that defines a group entities which is divided into sub groups based on their characteristic.
- It is a top down approach, in which one higher entity can be broken down into two lower level entity.
- It maximizes the difference between the members of an entity by identifying the unique characteristic or attributes of each member.
- It defines one or more sub class for the super class and also forms the superclass/subclass relationship.

The specialization process allows us to do the following:

- Define a set of subclasses of an entity type
- Establish additional specific attributes with each subclass
- Establish additional specific relationship types between each subclass and other entity types or other subclasses

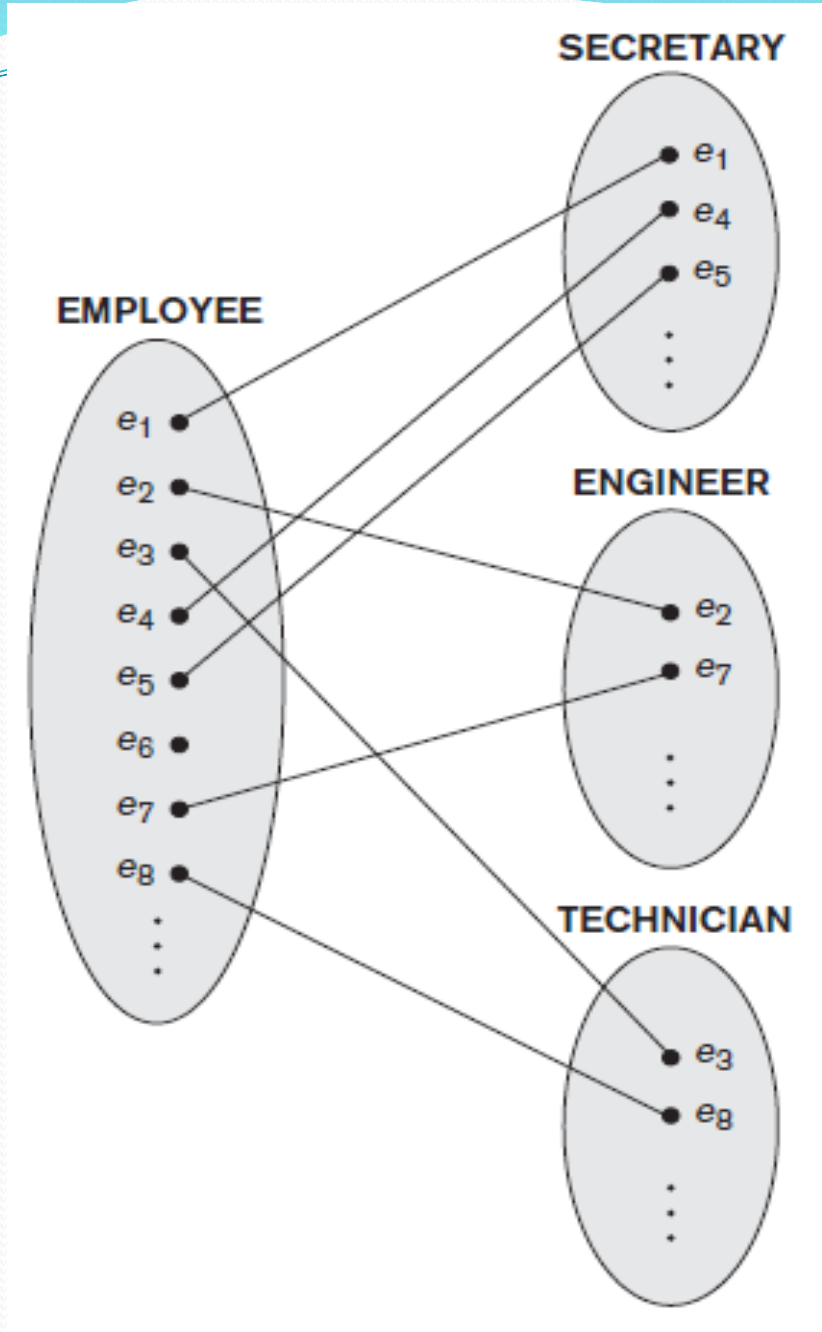
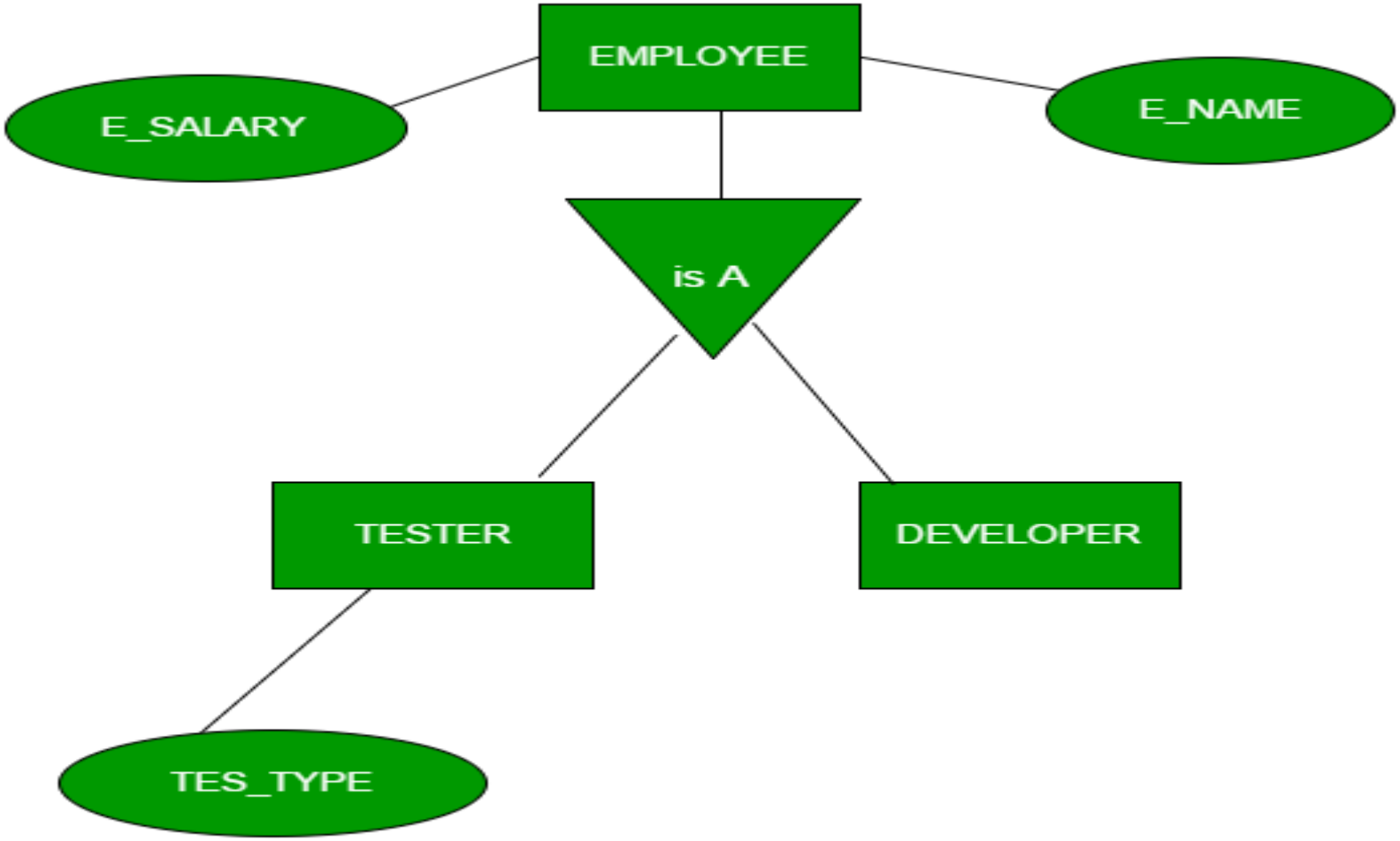



Fig: Instances of Specialization

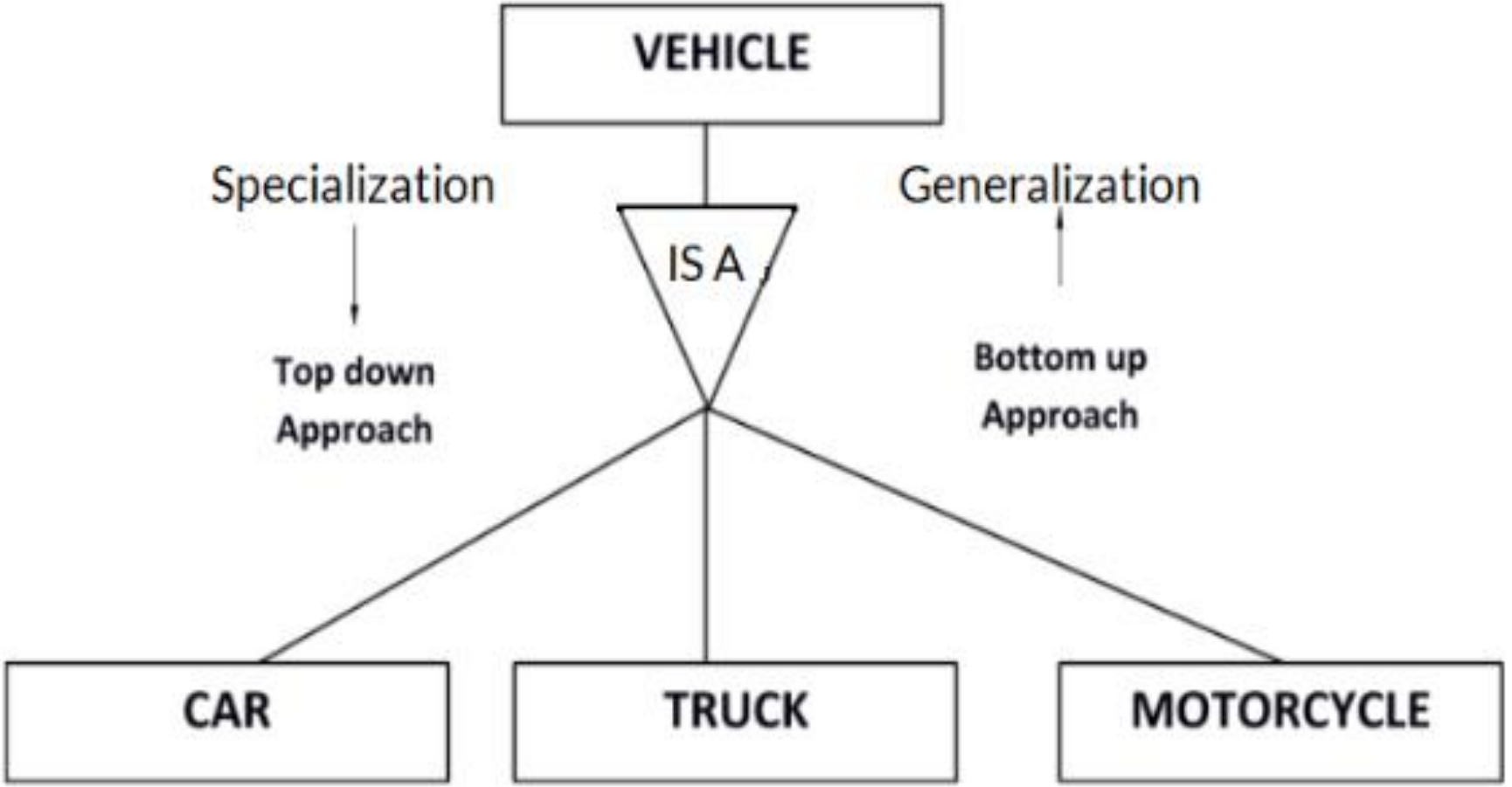



Specialization

In the above example, Employee can be specialized as Developer or Tester, based on what role they play in an Organization.

# Generalization

- Generalization is the process of generalizing the entities which contain the properties of all the generalized entities.
- It is a bottom-up approach, in which two lower level entities combine to form a higher level entity.
- Generalization is the reverse process of Specialization.
- It defines a general entity type from a set of specialized entity type.
- It minimizes the difference between the entities by identifying the common features.


VEHICLE
Specialization
Generalization
IS A
Top down
Approach
Bottom up
Approach
CAR
TRUCK
MOTORCYCLE

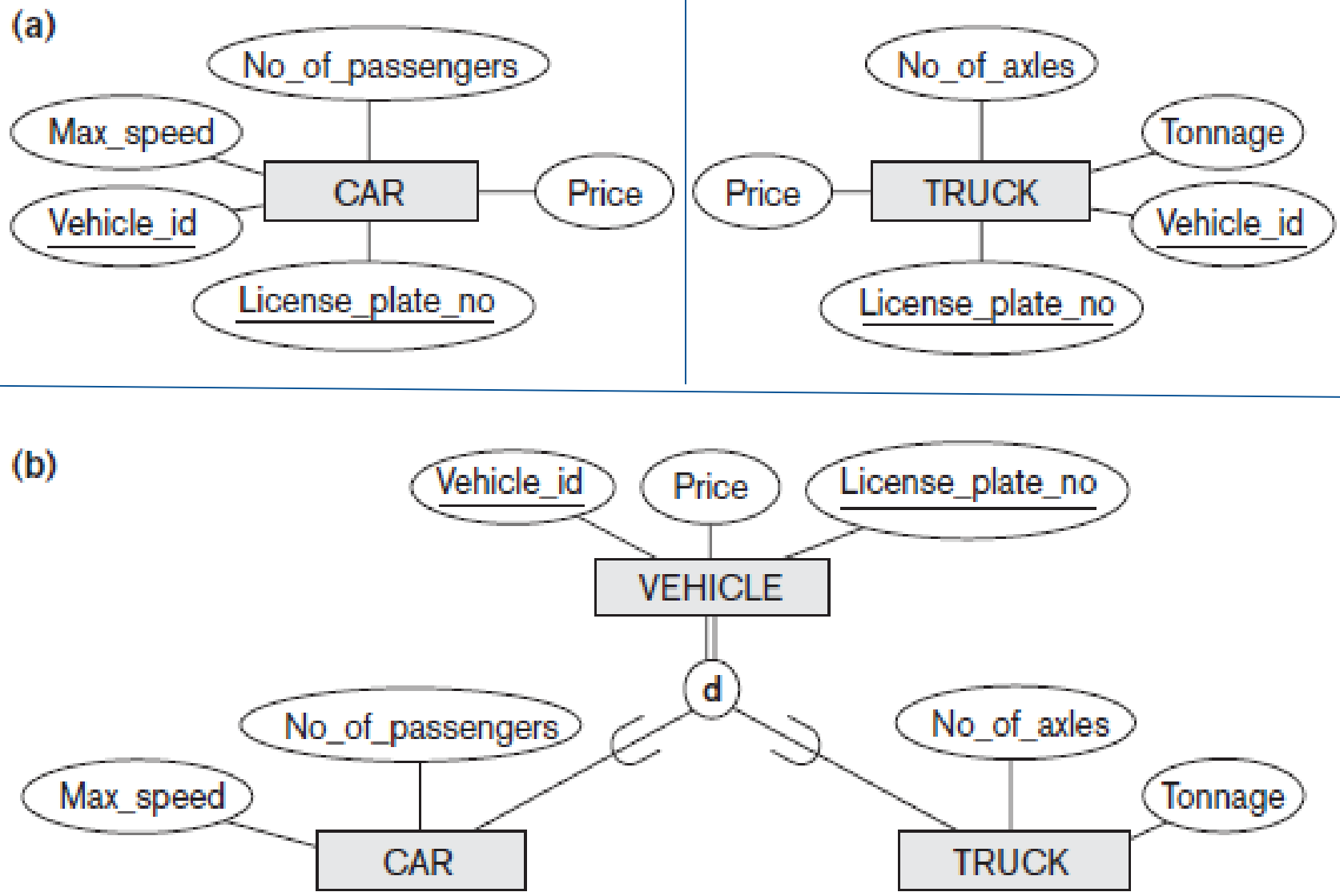



**Figure 8.3**
Generalization. (a) Two entity types, CAR and TRUCK. (b) Generalizing CAR and TRUCK into the superclass VEHICLE.

# Association

- Association represents the relationship between two classes.
- It can be Unidirectional (one way) or Bidirectional (two way).
- Unidirectional- Customer places an order.
- Bidirectional- A is married to B, B is married to A

# Aggregation

- There is a one limitation with E-R model that it cannot express relationships among relationships.
- So aggregation is an abstraction through which relationship is treated as *higher level entities*.

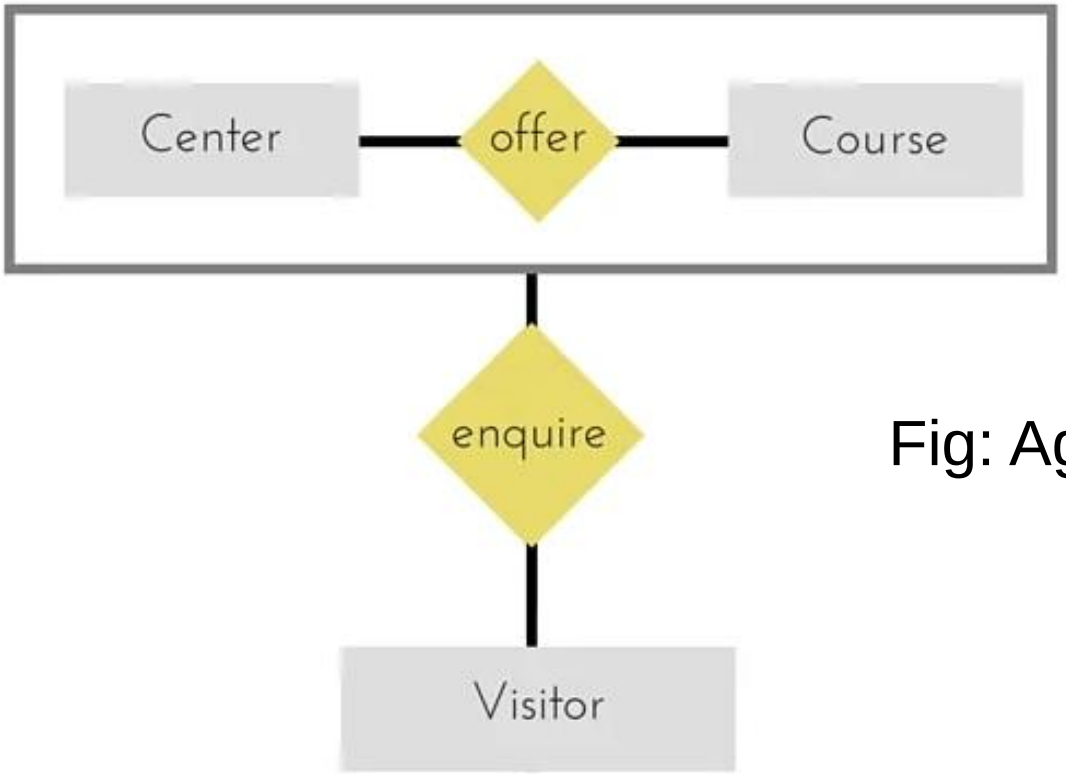
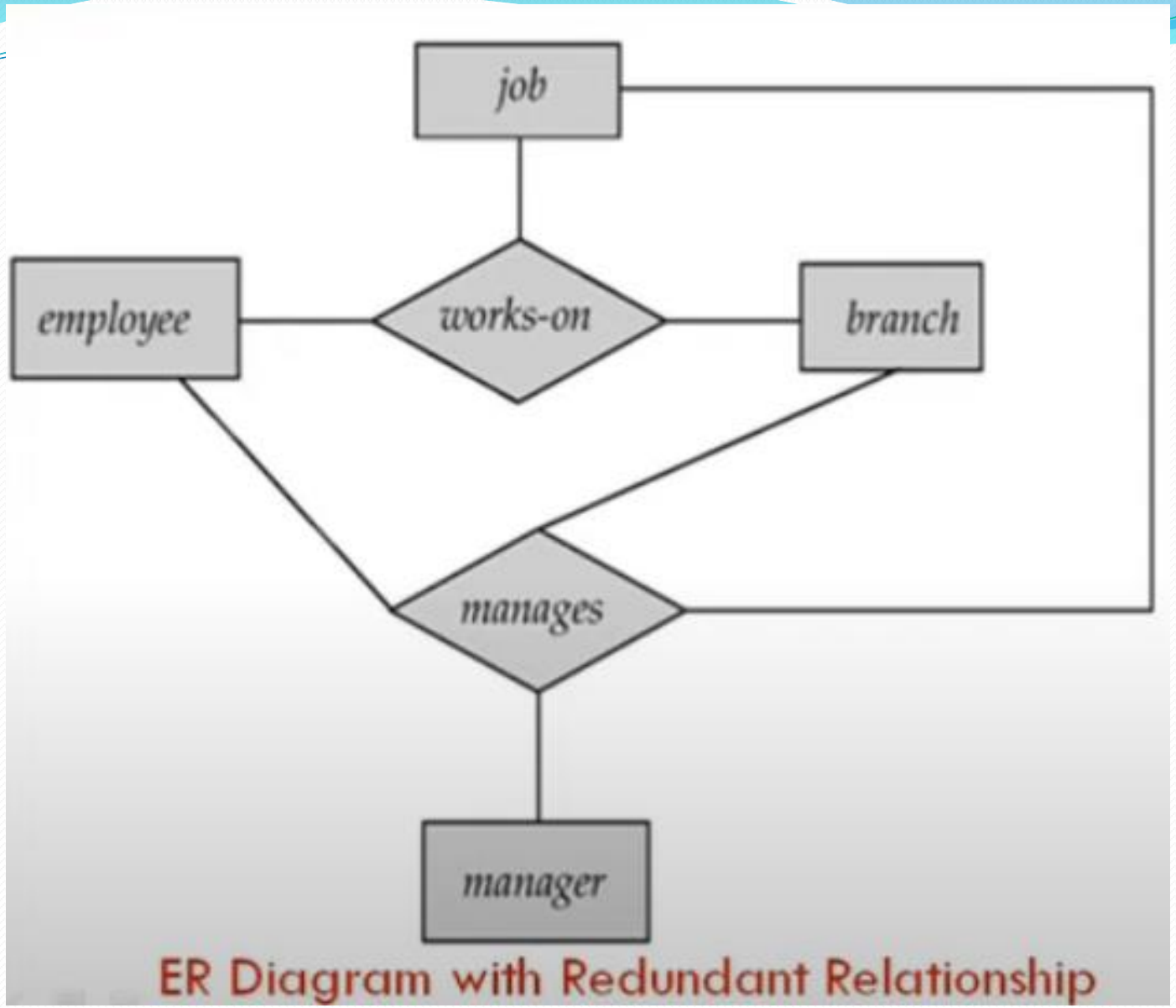
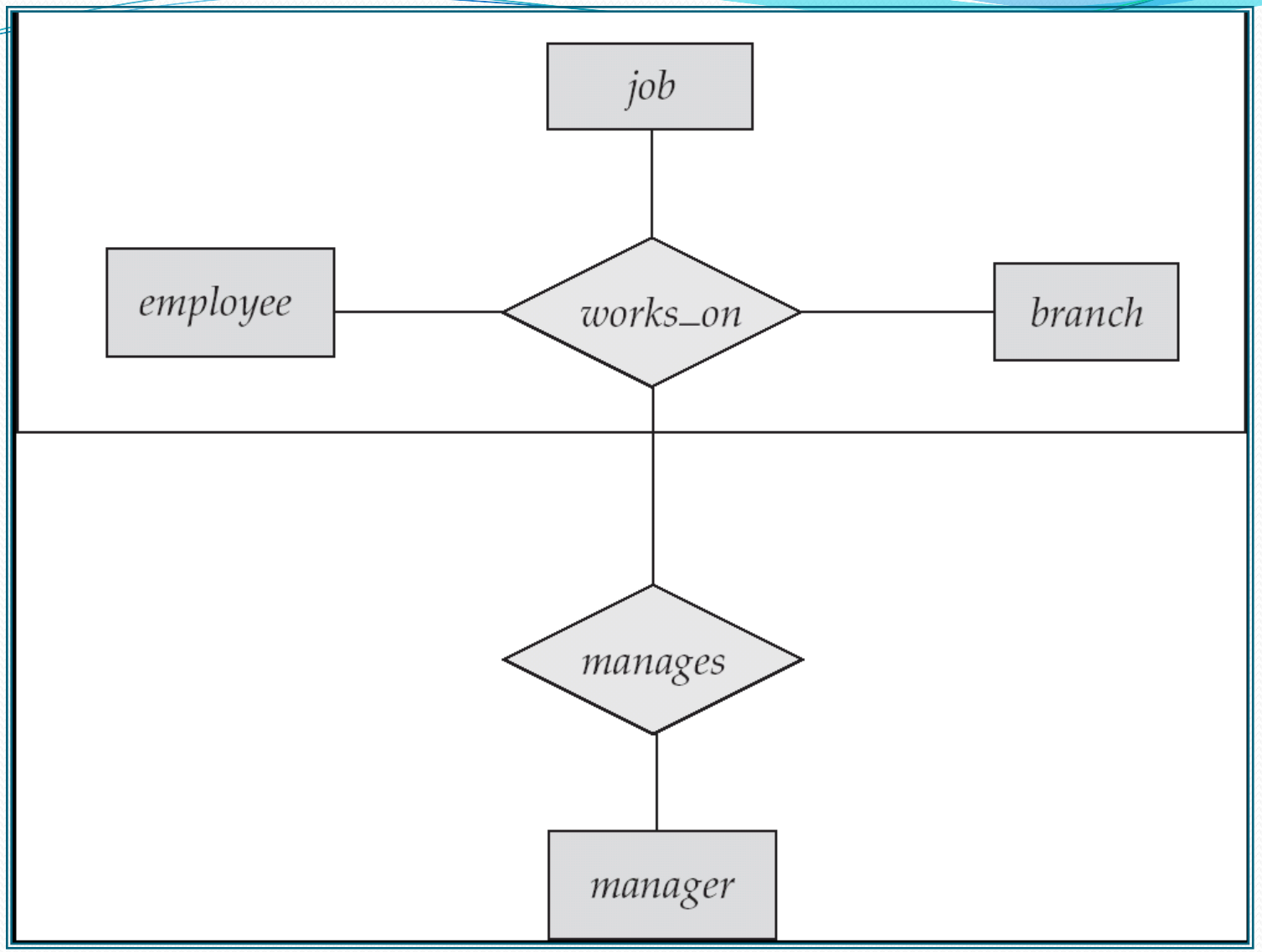



Fig: Aggregation

In the diagram above, the relationship between **Center** and **Course** together, is acting as an Entity, which is in relationship with another entity **Visitor**. Now in real world, if a Visitor or a Student visits a Coaching Center, he/she will never enquire about the center only or just about the course, rather he/she will ask enquire about both.




ER Diagram with Redundant Relationship


job
employee
works_on
branch
manages
manager

# Relational Model in DBMS

- Concept of Mathematical Relation and theoretically uses the concept of discrete mathematics.
- Relational Model was proposed by E. F. Codd to model data in the form of relations or tables.
- After designing the conceptual model of Database using ER diagram, we need to convert the conceptual model in the relational model which can be implemented using any RDBMS languages like Oracle SQL, MySQL etc.

**What is Relational Model?**

- Relational Model represents how data is stored in Relational Databases.
- A relational database stores data in the form of relations (tables). Consider a relation STUDENT with attributes ROLL_NO, NAME, ADDRESS, PHONE and AGE shown in Table 1.

| ROLL_NO | NAME | ADDRESS | PHONE | AGE |
|---|---|---|---|---|
| 1 | RAM | DELHI | 9455123451 | 18 |
| 2 | RAMESH | GURGAON | 9652431543 | 18 |
| 3 | SUJIT | ROHTAK | 9156253131 | 20 |
| 4 | SURESH | DELHI | | 18 |

# IMPORTANT TERMINOLOGIES

- A table is called a relation.
- Attribute (column headers): Attributes are the properties that define a relation.
  e.g.; ROLL_NO, NAME
- It contains the name of a column in a particular table. Each attribute Ai must have a domain, dom(Ai)
- Relation Schema: A relation schema represents name of the relation with its attributes.
  - Describes a relation.
  - A relational schema contains the name of the relation and name of all columns or attributes.
  - Made up of a relation name R and list of Attributes (A1, A2, A3, ……..)
  - e.g.; STUDENT (ROLL_NO, NAME, ADDRESS, PHONE, AGE) is relation schema for STUDENT.
  - STUDENT (ROLL_NO: int, NAME: varchar, ADDRESS: varchar, PHONE: int, AGE: int)
  - If a schema has more than 1 relation, it is called Relational Schema.

- Tuple: Each row in the relation is known as tuple. The above relation contains 4 tuples
- Domain: It contains a set of atomic values that an attribute can take.
- **Relation Instance (Relation State):** Set of tuples at a given time
- The set of tuples of a relation at a particular instance of time is called as relation instance. Table 1 shows the relation instance of STUDENT at a particular time.
- It can change whenever there is insertion, deletion or updation in the database.
- Relation instances do not have duplicate tuples.
- **Relational Database Schema:** set of relation schemas and a set of integrity constraints
- **Degree (arity):** The number of attributes in the relation is known as degree of the relation.
- The **STUDENT** relation defined above has degree 5.
- **Cardinality:** The number of tuples in a relation is known as cardinality.
- The **STUDENT** relation defined above has cardinality 4.

- **Column**: Column represents the set of values for a particular attribute.
- **NULL Values**: The value which is not known or unavailable is called NULL value.
  - It is represented by blank space.
  - e.g.; PHONE of STUDENT having ROLL_NO 4 is NULL.
- **Relational key**: In the relational key, each row has one or more attributes. It can identify the row in the relation uniquely.

# Types of Keys in Relational Model

- Keys are one of the basic requirements of a relational database model. It is widely used to identify the tuples(rows) uniquely in the table. We also use keys to set up relations amongst various columns and tables of a relational database.
- **Different Types of Database Keys**
  - Candidate Key
  - Primary Key
  - Super Key
  - Alternate Key
  - Foreign Key
  - Composite Key

# Candidate Key

- The minimal set of attributes that can uniquely identify a tuple is known as a candidate key. For Example, STUD_NO in STUDENT relation.
- It is a minimal super key.
- It is a super key with no repeated data is called a candidate key.
- The minimal set of attributes that can uniquely identify a record.
- It must contain unique values.
- It can contain NULL values.
- Every table must have at least a single candidate key.
- A table can have multiple candidate keys but only one primary key.
- The value of the Candidate Key is unique and may be null for a tuple.
- There can be more than one candidate key in a relationship.

- Example:
- STUD_NO is the candidate key for relation STUDENT.
- **Table STUDENT**

| STUD_NO | SNAME | ADDRESS | PHONE |
|---|---|---|---|
| 1 | Shyam | Delhi | 123456789 |
| 2 | Rakesh | Kolkata | 223365796 |
| 3 | Suraj | Delhi | 175468965 |

•The candidate key can be simple (having only one attribute) or composite as well.
•Example:
• {STUD_NO, COURSE_NO} is a composite
• candidate key for relation STUDENT_COURSE.
•**Table STUDENT_COURSE**

| STUD_NO | TEACHER_NO | COURSE_NO |
|---|---|---|
| 1 | 001 | C001 |
| 2 | 056 | C005 |

# Primary Key

- There can be more than one candidate key in relation out of which one can be chosen as the primary key. For Example, STUD_NO, as well as STUD_PHONE, are candidate keys for relation STUDENT but STUD_NO can be chosen as the primary key (only one out of many candidate keys).

- It is a unique key.
- It can identify only one tuple (a record) at a time.
- It has no duplicate values, it has unique values.
- It cannot be NULL.
- Primary keys are not necessarily to be a single column; more than one column can also be a primary key for a table.

- Example:
- STUDENT table -> Student(STUD_NO, SNAME,
- ADDRESS, PHONE) , STUD_NO is a primary key

- **Table STUDENT**

| STUD_NO | SNAME | ADDRESS | PHONE |
|---|---|---|---|
| 1 | Shyam | Delhi | 123456789 |
| 2 | Rakesh | Kolkata | 223365796 |
| 3 | Suraj | Delhi | 175468965 |

# Super Key

- The set of attributes that can uniquely identify a tuple is known as Super Key. For Example, STUD_NO, (STUD_NO, STUD_NAME), etc. A super key is a group of single or multiple keys that identifies rows in a table. It supports NULL values.

- Adding zero or more attributes to the candidate key generates the super key.
- A candidate key is a super key but vice versa is not true.
- Super Key values may also be NULL.

- Example:

**Table STUDENT**

| STUD_NO | SNAME | ADDRESS | PHONE |
|---|---|---|---|
| 1 | Shyam | Delhi | 123456789 |
| 2 | Rakesh | Kolkata | 223365796 |
| 3 | Suraj | Delhi | 175468965 |

- Consider the table shown above.
- STUD_NO+PHONE is a super key.

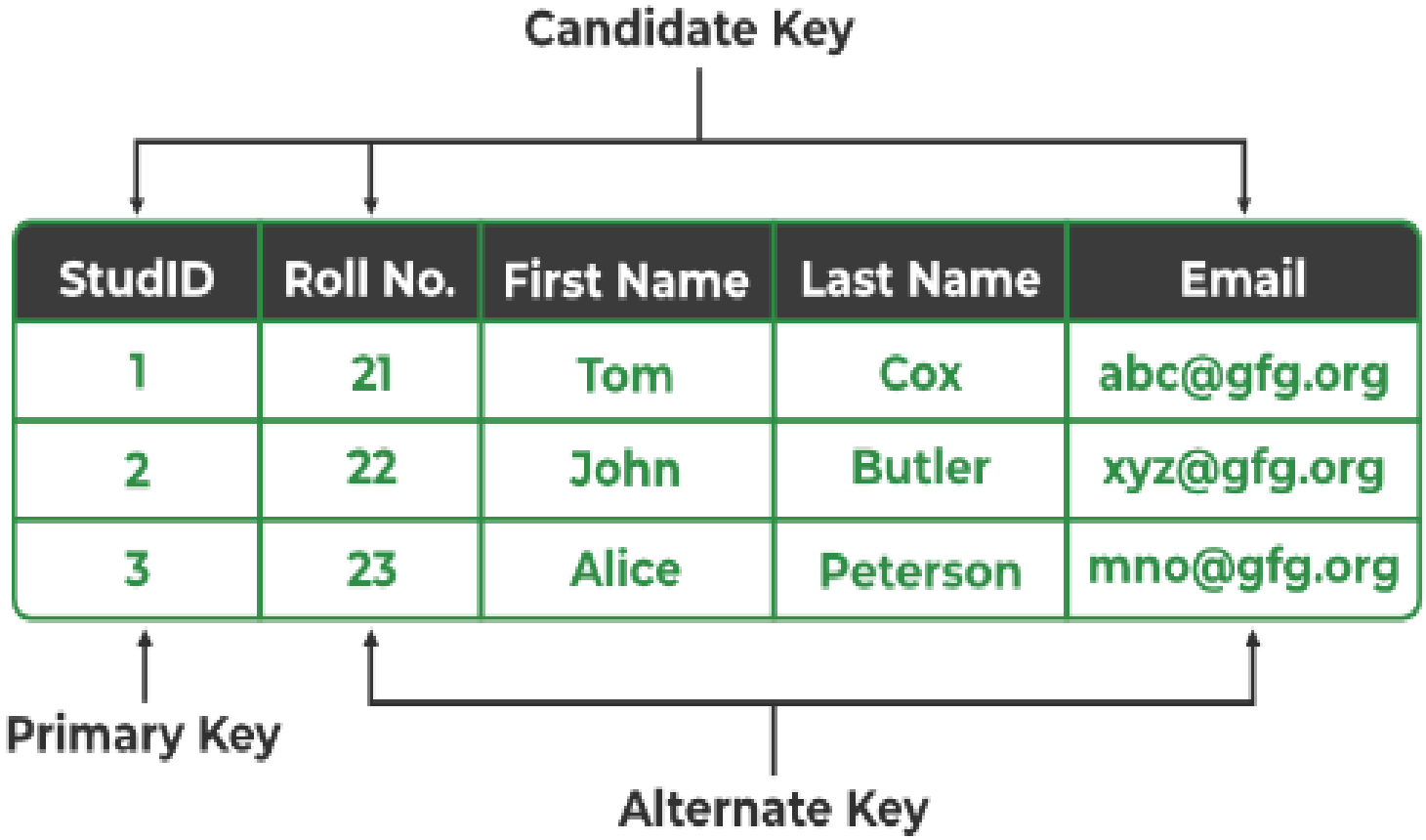
# Alternate Key

- The candidate key other than the primary key is called an alternate key.
- All the keys which are not primary keys are called alternate keys.
- It is a secondary key.
- It contains two or more fields to identify two or more records.
- These values are repeated.

- Example:

| STUD_NO | SNAME | ADDRESS | PHONE |
|---|---|---|---|
| 1 | Shyam | Delhi | 123456789 |
| 2 | Rakesh | Kolkata | 223365796 |
| 3 | Suraj | Delhi | 175468965 |

- Consider the table shown above.
- STUD_NO, as well as PHONE both, are candidate keys for relation STUDENT but PHONE will be an alternate key (only one out of many candidate keys).




*Primary Key, Candidate Key, and Alternate Key*

# Foreign Key

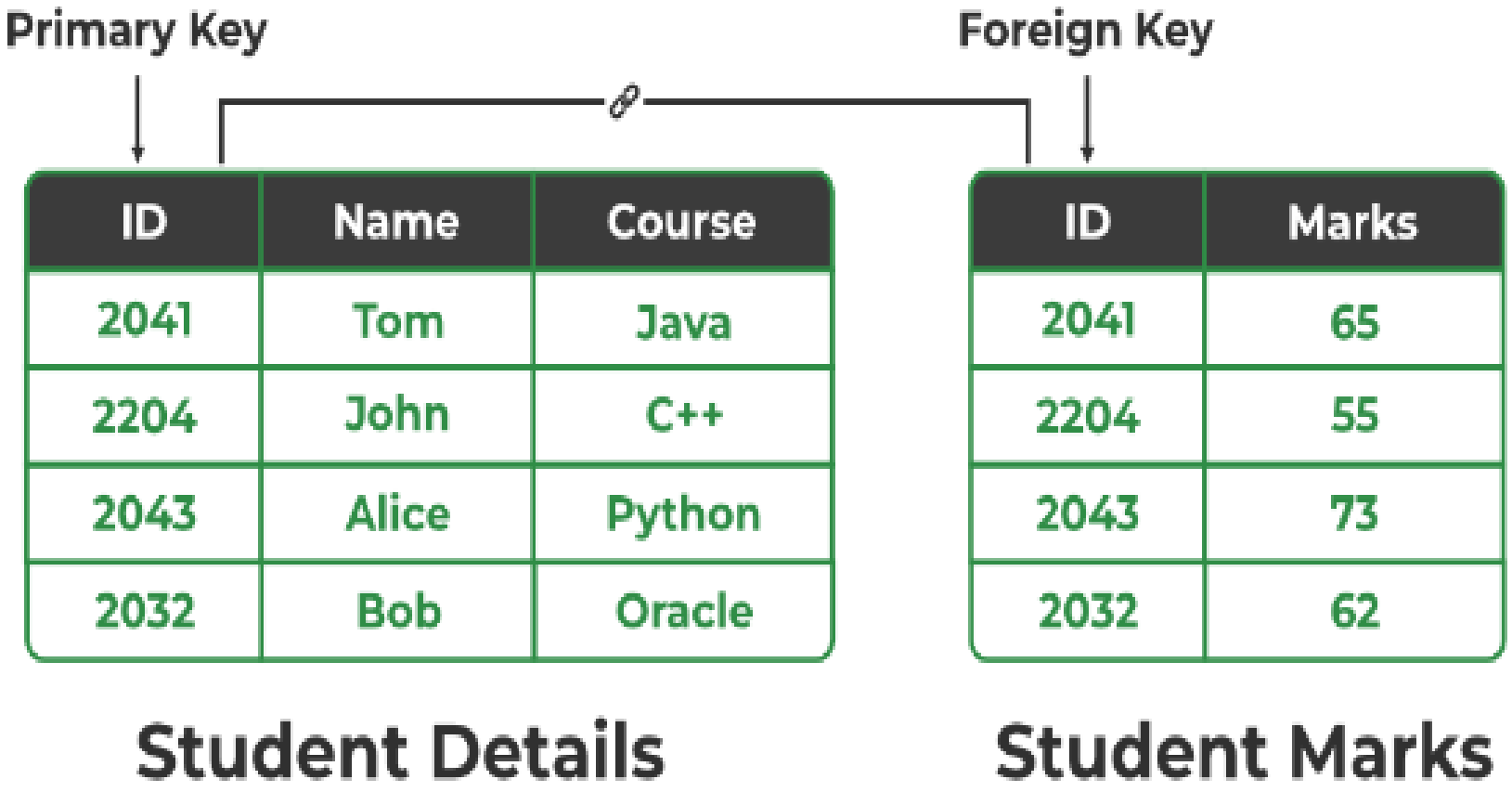
- If an attribute can only take the values which are present as values of some other attribute, it will be a foreign key to the attribute to which it refers. The relation which is being referenced is called referenced relation and the corresponding attribute is called referenced attribute. The referenced attribute of the referenced relation should be the primary key to it.
- It is a key it acts as a primary key in one table and it acts as secondary key in another table.
- It combines two or more relations (tables) at a time.
- They act as a cross-reference between the tables.
- For example, DNO is a primary key in the DEPT table and a non-key in EMP

- Example:

**Table STUDENT**

| STUD_NO | SNAME | ADDRESS | PHONE |
|---|---|---|---|
| 1 | Shyam | Delhi | 123456789 |
| 2 | Rakesh | Kolkata | 223365796 |
| 3 | Suraj | Delhi | 175468965 |

- Refer Table STUDENT shown above.
- STUD_NO in STUDENT_COURSE is a
- foreign key to STUD_NO in STUDENT relation.




*Relation between Primary Key and Foreign Key*

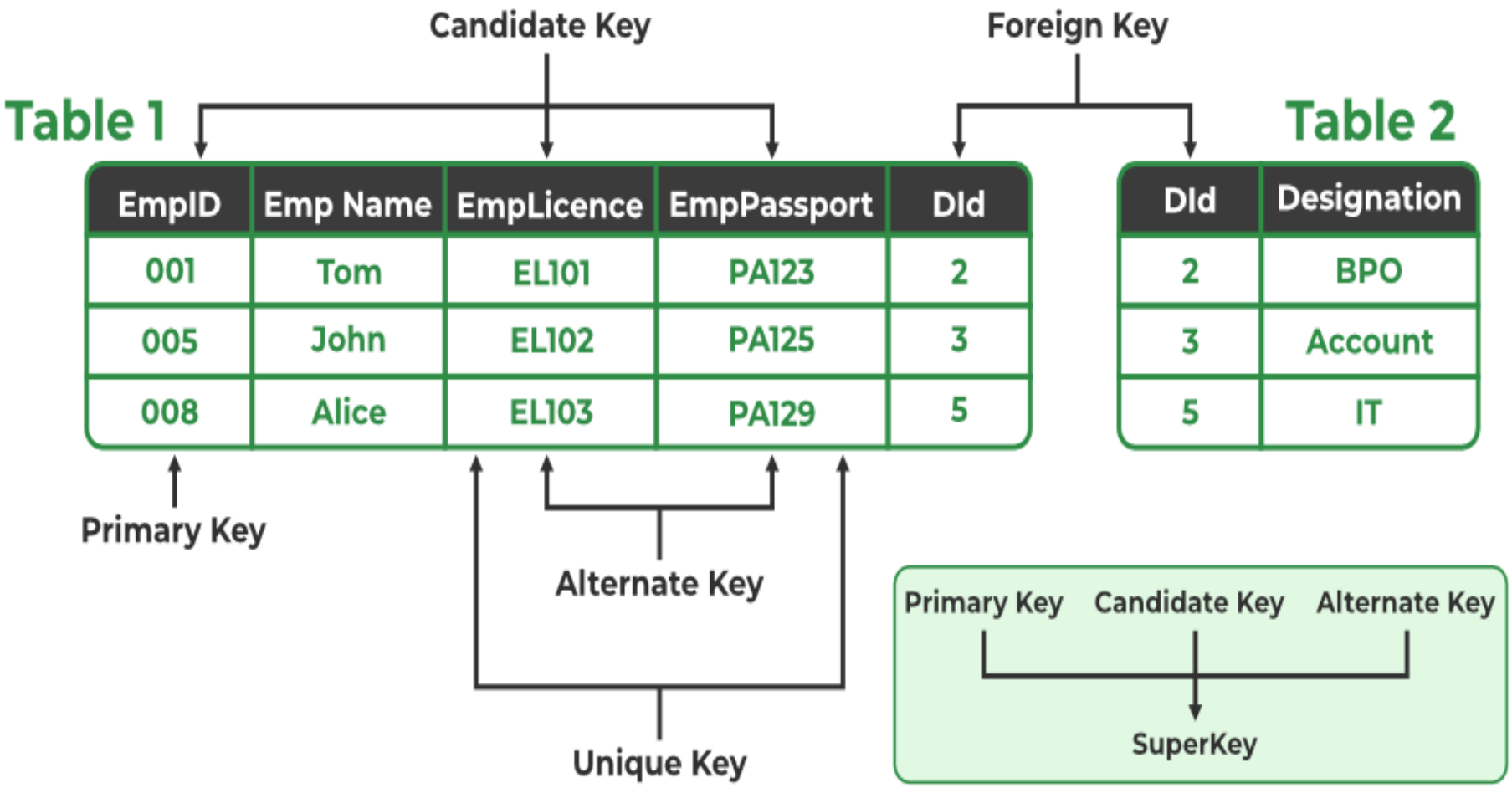
# Composite Key

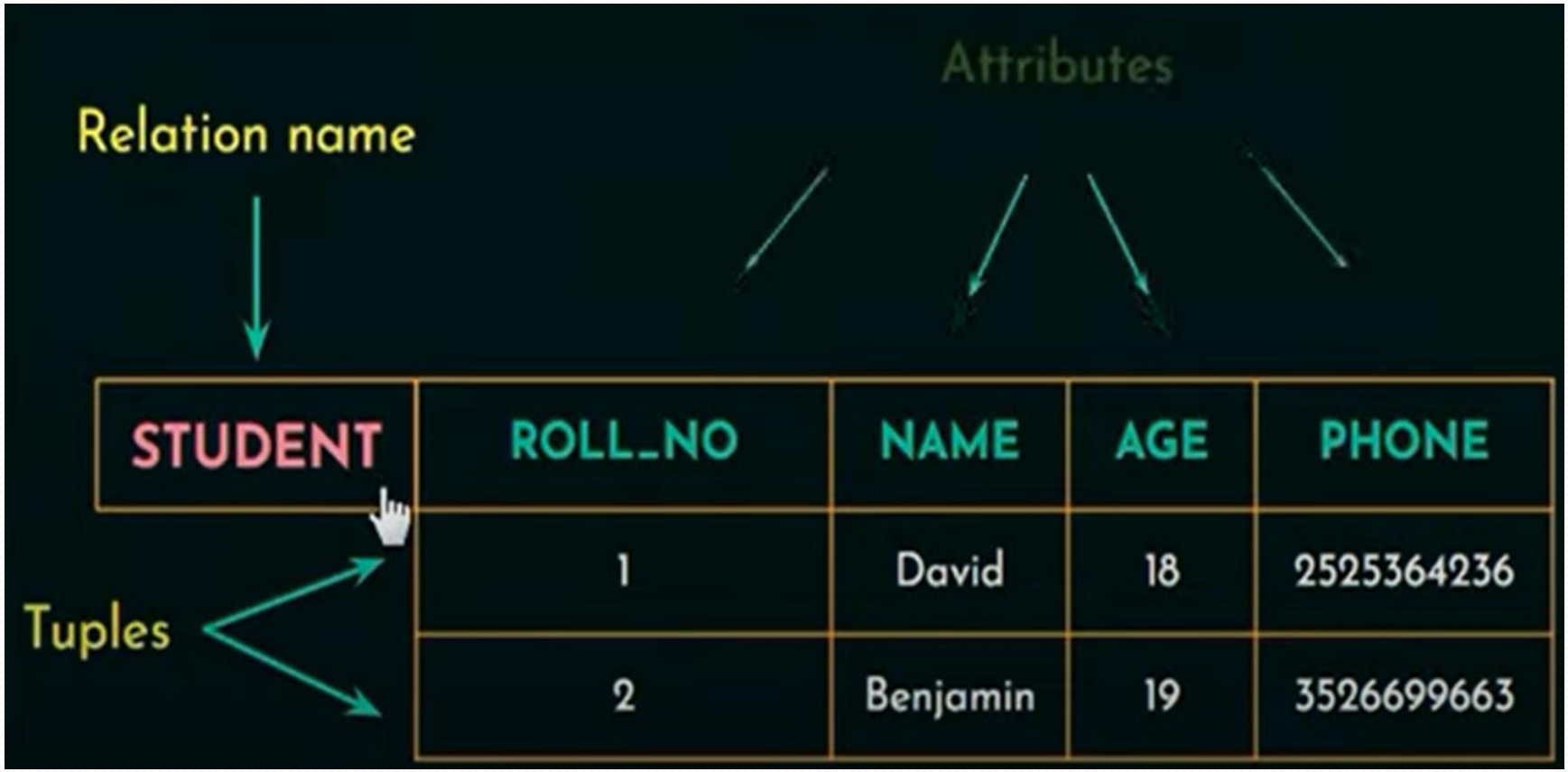
- Sometimes, a table might not have a single column/attribute that uniquely identifies all the records of a table. To uniquely identify rows of a table, a combination of two or more columns/attributes can be used. It still can give duplicate values in rare cases. So, we need to find the optimal set of attributes that can uniquely identify rows in a table.

- It acts as a primary key if there is no primary key in a table
- Two or more attributes are used together to make a composite key.
- Different combinations of attributes may give different accuracy in terms of identifying the rows uniquely.

- Example:

- FULLNAME + DOB can be combined
- together to access the details of a student.


Candidate Key
Foreign Key
Table 1
Table 2
EmpID
Emp Name
EmpLicence
EmpPassport
DId
001
Tom
EL101
PA123
2
005
John
EL102
PA125
3
008
Alice
EL103
PA129
5
DId
Designation
2
BPO
3
Account
5
IT
Primary Key
Alternate Key
Unique Key
Primary Key
Candidate Key
Alternate Key
SuperKey


Attributes
Relation name
STUDENT
ROLL_NO
NAME
AGE
PHONE
1
David
18
2525364236
2
Benjamin
19
3526699663
Tuples

- Relational database systems are expected to be equipped with a query language that can assist its users to query the database instances.
- There are two kinds of query languages −

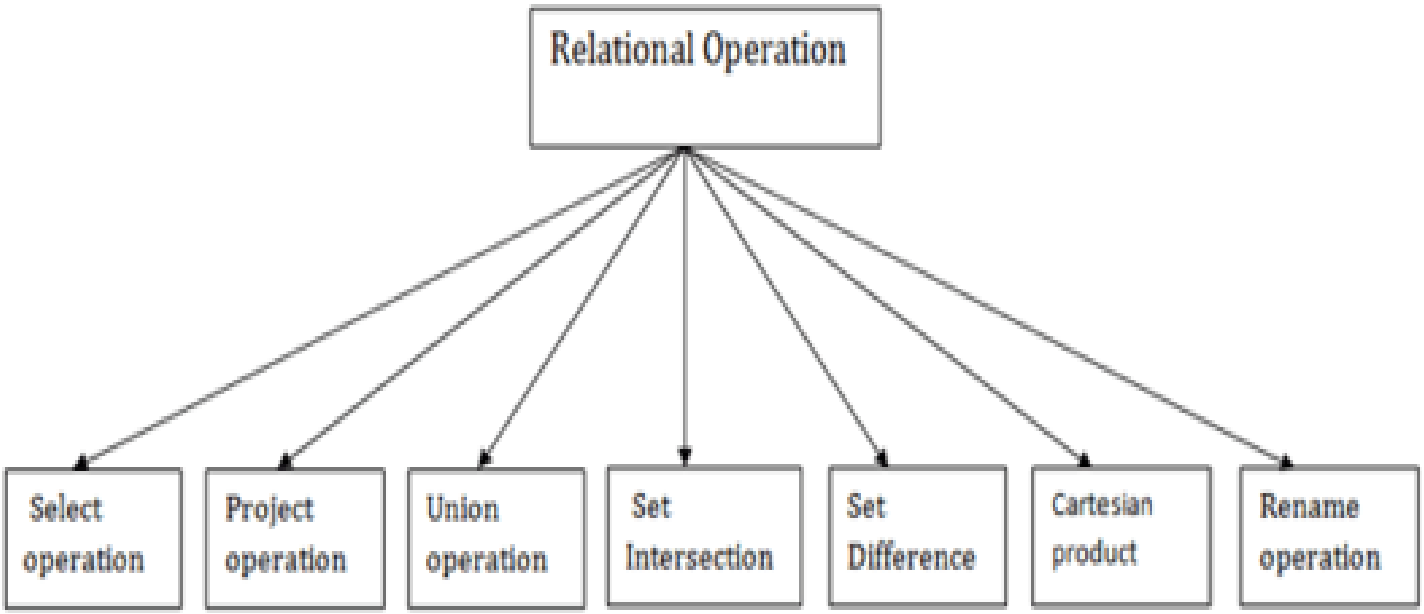
1. Relational Algebra - 6 fundamental operations

2. Relational Calculus - Tuple Relational Calculus (TRC)

   Domain Relational Calculus (DRC)

# Relational Algebra

- Relational algebra is a procedural query language.
- RA takes instances of relations as input and yields instances of relations as output.
- It gives a step by step process to obtain the result of the query.
- It uses operators to perform queries.
- An operator can be either unary or binary.
- Relational algebra is performed recursively on a relation and intermediate results are also considered as relations.


Relational Operation
Select operation
Project operation
Union operation
Set Intersection
Set Difference
Cartesian product
Rename operation

**Unary Relational Operations**

- SELECT (symbol: σ)
- PROJECT (symbol: π)
- RENAME (symbol: ρ)

**Relational Algebra Operations From Set Theory**

- UNION (υ)
- INTERSECTION ( ),
- DIFFERENCE (-)
- CARTESIAN PRODUCT ( x )

**Binary Relational Operations**

- JOIN
- DIVISION

## 1. Select Operation:

• The select operation selects tuples that satisfy a given predicate from a relation.

• It is denoted by sigma (σ).

Notation: σ p(r)

**Where:**

**σ** is used for selection prediction

**r** is used for relation

**p** is used as a propositional logic formula which may use connectors like: AND OR and NOT.

These relational can use as relational operators like =, ≠, ≥, <, >, ≤.

**For example: LOAN Relation**

| BRANCH_NAME | LOAN_NO | AMOUNT |
|---|---|---|
| Downtown | L-17 | 1000 |
| Redwood | L-23 | 2000 |
| Perryride | L-15 | 1500 |
| Downtown | L-14 | 1500 |
| Mianus | L-13 | 500 |
| Roundhill | L-11 | 900 |
| Perryride | L-16 | 1300 |

**Input:**

σ BRANCH_NAME="perryride" (LOAN)

**Output:**

| BRANCH_NAME | LOAN_NO | AMOUNT |
|---|---|---|
| Perryride | L-15 | 1500 |
| Perryride | L-16 | 1300 |

**For example** −

$$\sigma_{subject = "database"}(Books)$$

**Output** − Selects tuples from books where subject is 'database'.

$$\sigma_{subject = "database" \text{ and } price = "450"}(Books)$$

**Output** − Selects tuples from books where subject is 'database' and 'price' is 450.

$$\sigma_{subject = "database" \text{ and } price = "450" \text{ or } year > "2010"}(Books)$$

**Output** − Selects tuples from books where subject is 'database' and 'price' is 450 or those books published after 2010.

## 2. Project Operation:

• This operation shows the list of those attributes that we wish to appear in the result.

• Rest of the attributes are eliminated from the table.

• Duplicate rows are automatically eliminated, as relation is a set.

• It is denoted by ∏.

Notation: ∏ **A1, A2, An (r)**

Where

**A1**, **A2**, **A3** is used as an attribute name of relation **r**.

**For example** −

$$\prod_{\text{subject, author}} \textbf{(Books)}$$

Selects and projects columns named as subject and author from the relation Books

**Example: CUSTOMER RELATION**

| NAME | STREET | CITY |
|---|---|---|
| Jones | Main | Harrison |
| Smith | North | Rye |
| Hays | Main | Harrison |
| Curry | North | Rye |
| Johnson | Alma | Brooklyn |
| Brooks | Senator | Brooklyn |

**Input:**

∏ NAME, CITY (CUSTOMER)

**Output:**

| NAME | CITY |
|---|---|
| Jones | Harrison |
| Smith | Rye |
| Hays | Harrison |
| Curry | Rye |
| Johnson | Brooklyn |
| Brooks | Brooklyn |

## 3. Union Operation:

- Suppose there are two tuples R and S.
- The union operation contains all the tuples that are either in R or S or both in R & S.
- It eliminates the duplicate tuples. It is denoted by ∪.
- It performs binary union between two given relations and is defined as
- Notation: $R \cup S = \{ t \mid t \in R \text{ or } t \in S \}$

A union operation must hold the following conditions:

- R and S must have the attribute of the same number.
- Attribute domains must be compatible
- Duplicate tuples are eliminated automatically.

$\prod_{author}$ (Books) ∪ $\prod_{author}$ (Articles)

**Output** – Projects the names of the authors who have either written a book or an article or both.

## DEPOSITOR RELATION

| CUSTOMER_NAME | ACCOUNT_NO |
|---|---|
| Johnson | A-101 |
| Smith | A-121 |
| Mayes | A-321 |
| Turner | A-176 |
| Johnson | A-273 |
| Jones | A-472 |
| Lindsay | A-284 |

## BORROW RELATION

| CUSTOMER_NAME | LOAN_NO |
|---|---|
| Jones | L-17 |
| Smith | L-23 |
| Hayes | L-15 |
| Jackson | L-14 |
| Curry | L-93 |
| Smith | L-11 |
| Williams | L-17 |

**Input:**

∏ CUSTOMER_NAME (BORROW) ∪ ∏ CUSTOMER_NAME (DEPOSITOR)

**Output:**

| CUSTOMER_NAME |
|---|
| Johnson |
| Smith |
| Hayes |
| Turner |
| Jones |
| Lindsay |
| Jackson |
| Curry |
| Williams |
| Mayes |

## 4. Set Intersection:

• Suppose there are two tuples R and S. The set intersection operation contains all tuples that are in both R & S.

• It is denoted by intersection ∩.

Notation: R ∩ S

**Example:** Using the above DEPOSITOR table and BORROW table

**Input:**

∏ CUSTOMER_NAME (BORROW) ∩ ∏ CUSTOMER_NAME (DEPOSITOR)

**Output:**

| CUSTOMER_NAME |
| --- |
| Smith |
| Jones |

# 5. Set Difference:

- Suppose there are two tuples R and S.
- The set Difference operation contains all tuples that are in R but not in S.
- It is denoted by intersection minus (-).
- Notation: R - S

**Example:** Using the above DEPOSITOR table and BORROW table

**Input:**

| CUSTOMER_NAME |
|---|
| Jackson |
| Hayes |
| Willians |
| Curry |

**Example:**

$\prod$ CUSTOMER_NAME (BORROW) -

$\prod$ CUSTOMER_NAME (DEPOSITOR)

$\prod_{author}$ (Books) – $\prod_{author}$ (Articles)

**Output** – Provides the name of authors who have written books but not articles.

## 6. Cartesian product

• The Cartesian product is used to combine each row in one table with each row in the other table.

• It is also known as a **cross product**.

• It is denoted by X.

Notation: R X S

Example:

$r \times s = \{ q\ t \mid q \in r \text{ and } t \in s \}$

$\sigma_{author = \text{'jss'}}(Books \times Articles)$

**Output** – Yields a relation, which shows all the books and articles written by jss.

**EMPLOYEE**

| EMP_ID | EMP_NAME | EMP_DEPT |
|---|---|---|
| 1 | Smith | A |
| 2 | Harry | C |
| 3 | John | B |

**DEPARTMENT**

| DEPT_NO | DEPT_NAME |
|---|---|
| A | Marketing |
| B | Sales |
| C | Legal |

**Input:**

EMPLOYEE X DEPARTMENT

**Output:**

| EMP_ID | EMP_NAME | EMP_DEPT | DEPT_NO | DEPT_NAME |
|---|---|---|---|---|
| 1 | Smith | A | A | Marketing |
| 1 | Smith | A | B | Sales |
| 1 | Smith | A | C | Legal |
| 2 | Harry | C | A | Marketing |
| 2 | Harry | C | B | Sales |
| 2 | Harry | C | C | Legal |
| 3 | John | B | A | Marketing |
| 3 | John | B | B | Sales |
| 3 | John | B | C | Legal |

## 7. Rename Operation:

- The rename operation is used to rename the output relation.
- It is denoted by **rho** (ρ).

**Example:** We can use the rename operator to rename STUDENT relation to STUDENT1.

ρ (STUDENT1, STUDENT)

**Join Operations**

- Join operation is essentially a Cartesian product followed by a selection criterion.
- Join operation denoted by ⋈
- JOIN operation also allows joining variously related tuples from different relations.

**Types of JOIN:**

Various forms of join operation are:

| Inner Joins: | Outer joins: |
| --- | --- |
| • Theta join<br>• EQUI join<br>• Natural join | • Left Outer Join<br>• Right Outer Join<br>• Full Outer Join |

**Inner Join:**

In an inner join, **only those tuples that satisfy the matching criteria are included**, while the rest are excluded.

**Theta Join:**

The general case of JOIN operation is called a Theta join. It is denoted by symbol **θ**

Example $A \bowtie_{\theta} B$

Theta join can use any conditions in the selection criteria.

For example: $A \bowtie_{A.column\ 2 > B.column\ 2} (B)$

Table A

| column 1 | column 2 |
|---|---|
| 1 | 1 |
| 1 | 2 |

Table B

| column 1 | column 2 |
|---|---|
| 1 | 1 |
| 1 | 3 |

$A \bowtie A.column\ 2 > B.column\ 2\ (B)$

| column 1 | column 2 |
|---|---|
| 1 | 2 |

**EQUI join:**

- When a theta join uses only equivalence condition, it becomes a equi join.
- EQUI join is the most difficult operations to implement efficiently using SQL in an RDBMS and one reason why RDBMS have essential performance problems.

For example:

$$A \bowtie_{A.column\ 2 = B.column\ 2} (B)$$

| Table A | |
|---|---|
| column 1 | column 2 |
| 1 | 1 |
| 1 | 2 |

| Table B | |
|---|---|
| column 1 | column 2 |
| 1 | 1 |
| 1 | 3 |

| A ⋈ A.column 2 = B.column 2 (B) | |
|---|---|
| column 1 | column 2 |
| 1 | 1 |

## NATURAL JOIN (⋈)

Natural join can only be performed if there is a common attribute (column) between the relations. The name and type of the attribute must be same.

| C | |
|---|---|
| Num | Square |
| 2 | 4 |
| 3 | 9 |

| D | |
|---|---|
| Num | Cube |
| 2 | 8 |
| 3 | 27 |

**C ⋈ D**

| C ⋈ D | | |
|---|---|---|
| Num | Square | Cube |
| 2 | 4 | 8 |
| 3 | 9 | 27 |

## OUTER JOIN

In an outer join, along with tuples that satisfy the matching criteria, we also include some or all tuples that do not match the criteria.

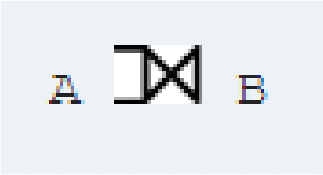
**Left Outer Join (A ⟕ B)**

In the left outer join, operation allows keeping all tuple in the left relation. However, if there is no matching tuple is found in right relation, then the attributes of right relation in the join result are filled with null values.

| A | |
|---|---|
| Num | Square |
| 2 | 4 |
| 3 | 9 |
| 4 | 16 |

| B | |
|---|---|
| Num | Cube |
| 2 | 8 |
| 3 | 18 |
| 5 | 75 |

## Left Outer Join




| Num | Square | Cube |
|---|---|---|
| 2 | 4 | 8 |
| 3 | 9 | 18 |
| 4 | 16 | – |

**Right Outer Join:** **( A ⟖ B )**

In the right outer join, operation allows keeping all tuple in the right relation. However, if there is no matching tuple is found in the left relation, then the attributes of the left relation in the join result are filled with null values.


| | A |
|---|---|
| Num | Square |
| 2 | 4 |
| 3 | 9 |
| 4 | 16 |

| | B |
|---|---|
| Num | Cube |
| 2 | 8 |
| 3 | 18 |
| 5 | 75 |

**( A ⟖ B )**

| Num | Cube | Square |
|---|---|---|
| 2 | 8 | 4 |
| 3 | 18 | 9 |
| 5 | 75 | – |

**Full Outer Join:** **( A ⟗ B)**

In a full outer join, all tuples from both relations are included in the result, irrespective of the matching condition.

| | A |
|---|---|
| **Num** | **Square** |
| 2 | 4 |
| 3 | 9 |
| 4 | 16 |

| | B |
|---|---|
| **Num** | **Cube** |
| 2 | 8 |
| 3 | 18 |
| 5 | 75 |

**A ⟗ B**

| | | A ⋈ B |
|---|---|---|
| **Num** | **Cube** | **Square** |
| 2 | 4 | 8 |
| 3 | 9 | 18 |
| 4 | 16 | – |
| 5 | – | 75 |

| Operation(Symbols) | Purpose |
|---|---|
| Select(σ) | The SELECT operation is used for selecting a subset of the tuples according to a given selection condition |
| Projection(π) | The projection eliminates all attributes of the input relation but those not mentioned in the projection list. |
| Union Operation(∪) | UNION is symbolized by symbol U. It includes all tuples that are in tables A or in B. |
| Set Difference(-) | – Symbol denotes it. The result of A – B, is a relation which includes all tuples that are in A but not in B. |
| Intersection(∩) | Intersection defines a relation consisting of a set of all tuple that are in both A and B. |
| Cartesian Product(X) | Cartesian operation is helpful to merge columns from two relations. |
| Inner Join | Inner join, includes only those tuples that satisfy the matching criteria. |
| Theta Join(θ) | The general case of JOIN operation is called a Theta join. It is denoted by symbol θ. |
| EQUI Join | When a theta join uses only equivalence condition, it becomes a equi join. |
| Natural Join(⋈) | Natural join can only be performed if there is a common attribute (column) between the relations. |
| Outer Join | In an outer join, along with tuples that satisfy the matching criteria. |
| Left Outer Join( ) | In the left outer join, operation allows keeping all tuple in the left relation. |
| Right Outer join() | In the right outer join, operation allows keeping all tuple in the right relation. |
| Full Outer Join() | In a full outer join, all tuples from both relations are included in the result irrespective of the matching condition. |

# Example Queries on Relational Algebra

**(1).** Suppose there is a banking database which comprises following tables :

Customer(Customer_name, Cust_street, Cust_city)

Branch(Branch_name, Branch_city, Assets)

Account (Branch_name, Account_number, Balance)

Loan(Branch_name, Loan_number, Amount)

Depositor(Customer_name, Account_number)

Borrower(Customer_name, Loan_number)

**Query :** Find the names of all the customers who have taken a loan from the bank and also have an account at the bank.

Solution:

Depositor(Customer_name, Account_number)

Borrower(Customer_name, Loan_number)

**Query :** Find the names of all the customers who have taken a loan from the bank and also have an account at the bank.

Solution:

**Step 1 :** Identify the relations that would be required to frame the resultant query.

First half of the query(i.e. names of customers who have taken loan) indicates "borrowers" information.

So Relation 1 ——> Borrower.

Second half of the query needs Customer Name and Account number which can be obtained from Depositor relation.

Hence, Relation 2——> Depositor.

**Step 2 :** Identify the columns which you require from the relations obtained in Step 1.

**Column 1 :** Customer_name from Borrower

**Column 2 :** Customer_name from Depositor

**Step 3 :** Identify the operator to be used. We need to find out the **names of customers** who are present in **both Borrower** table and **Depositor** table.

Hence, operator to be used—-> Intersection.

Final Query will be

$$\Pi_{customer\_name}(borrower) \cap \Pi_{customer\_name}(depositor)$$

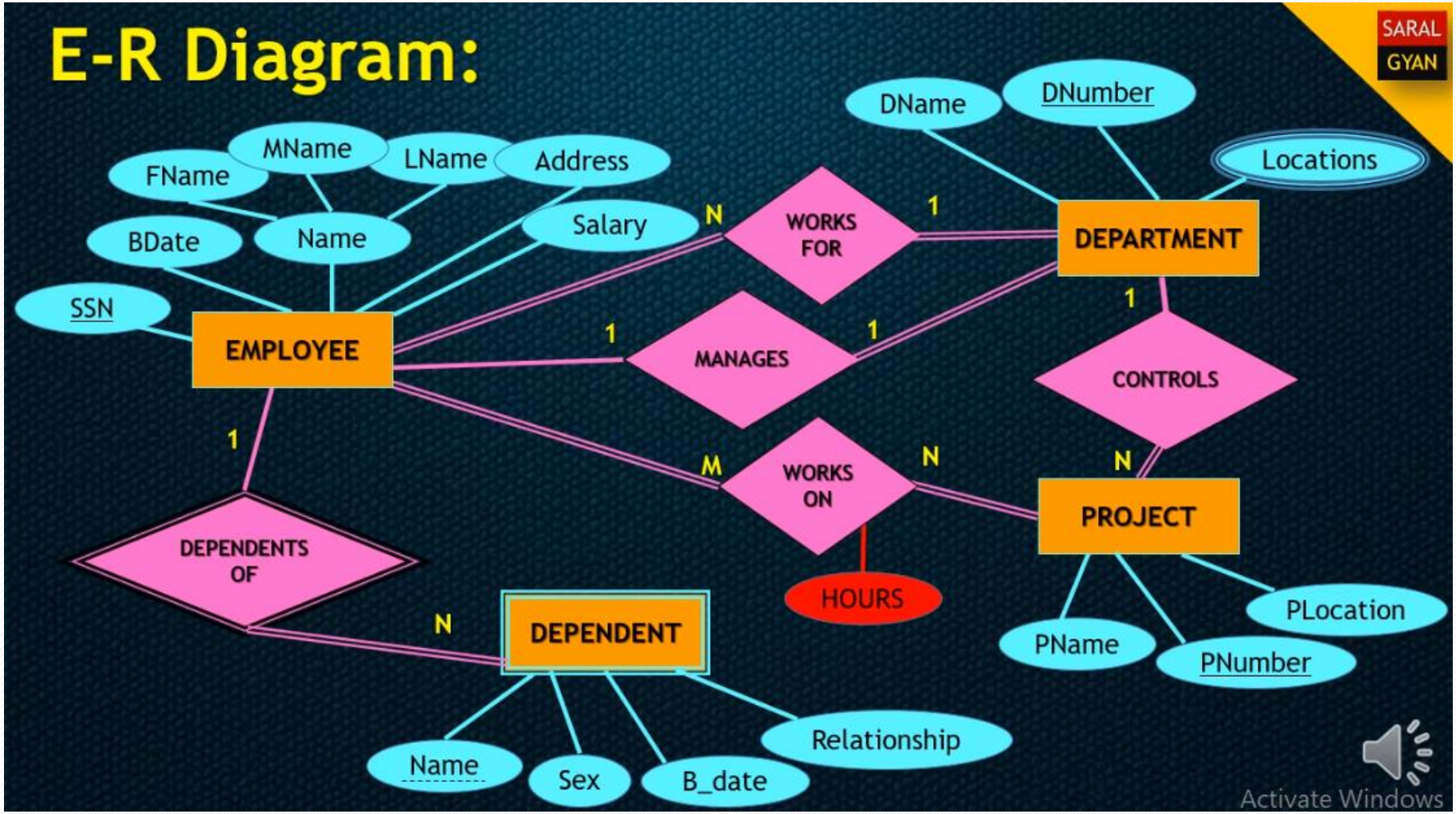
- Consider the following relational database of a college.
- Student(RollNumber, StudentName, Address)
- Teachers(TeacherID, TecherName, TeachingSubject)
- College(RollNumber, TeacherID)
- Write SQL or Relational algebra expressions for the following requests.

1. Find the name of Students who live in Lalitpur.
2. Find the name of the teacher who teaches Database Management System subject.
3. Delete records of students whose address is "Pokhara".

- Let us now see the queries.

1. $\Pi_{studentname}(\sigma Address= \text{“Lalitpur”}(Student))$
   Here we will be using select operator with projection operator to select all tuples that satisfy the condition where the address is Lalitpur.
2. $\Pi_{TeacherNamet}(\sigma TeachingSubject=\text{“Database Management System”}(Teachers))$
   Here we will be using select operator with projection operator to select all tuples that satisfy the condition where the subject is DBMS.
3. Student<Student-(σAddress= “Pokhara”(Students))
   Here we will be using the assignment operator first and then subtracting with the select operator to remove the data.

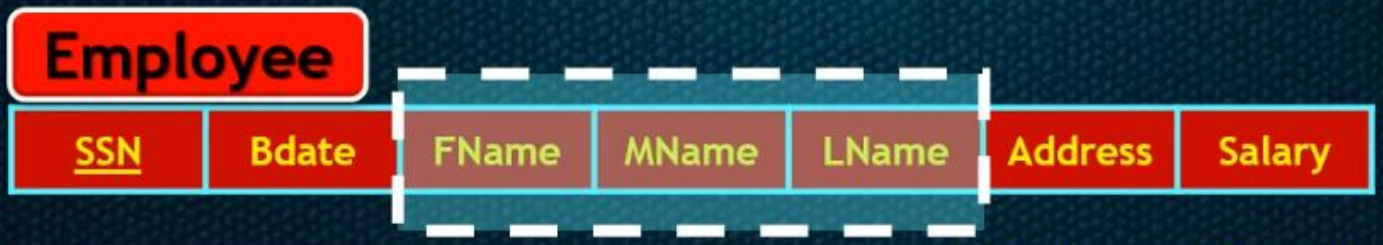
# Mapping the ER and EER Model to the Relational Model


E-R Diagram:
SARAL GYAN
DNumber
DName
Locations
MName
LName
Address
FName
Name
BDate
Salary
SSN
EMPLOYEE
WORKS FOR
DEPARTMENT
MANAGES
CONTROLS
WORKS ON
PROJECT
DEPENDENTS OF
DEPENDENT
HOURS
PLocation
PName
PNumber
Relationship
Name
Sex
B_date
1
N
M
Activate Windows

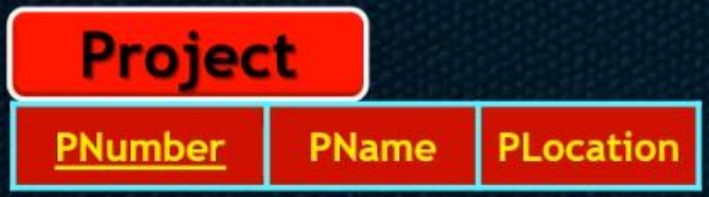
# Steps for Mapping:

SARAL GYAN

## Step 1: Convert all Strong Entity Sets in to Relations

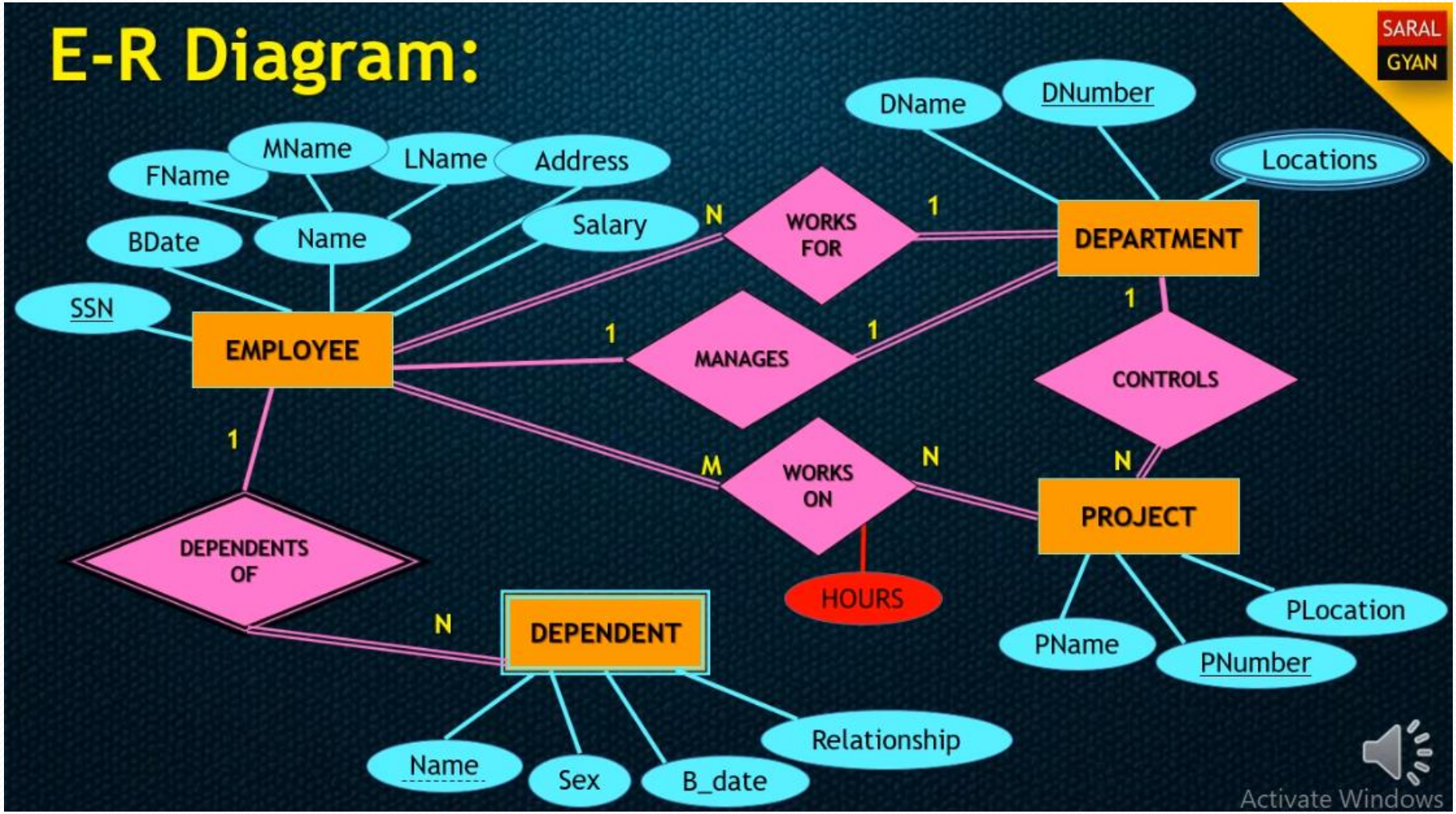




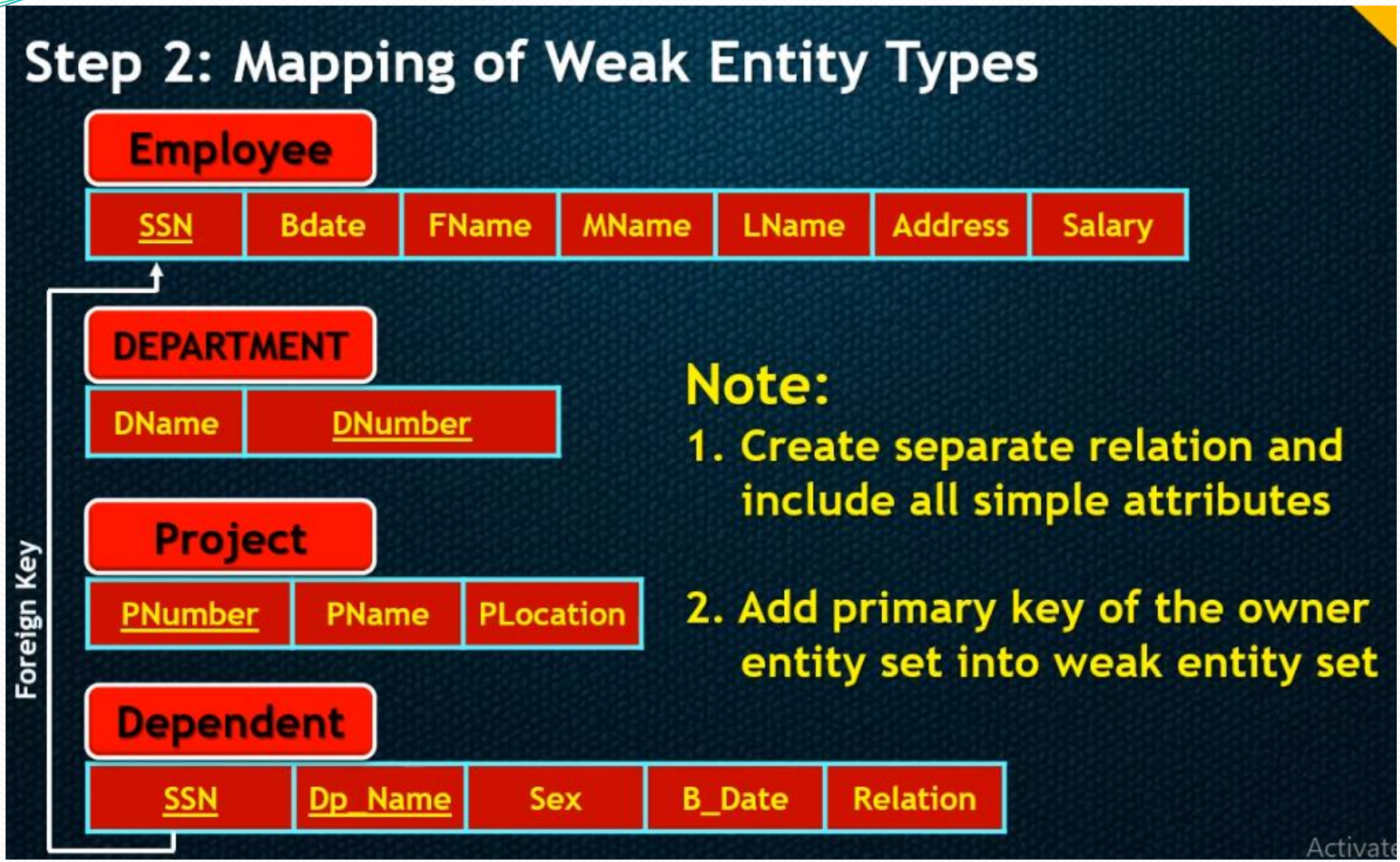



1. Only constituent simple attributes will be mapped in case of composite attribute e.g. name in the entity set employee.

2. Exclude Multivalued attribute from mapping into tables/ relations e.g. Locations attribute is not included into table Department


E-R Diagram:
SARAL GYAN
DNumber
DName
Locations
MName
LName
Address
FName
Name
BDate
Salary
SSN
EMPLOYEE
WORKS FOR
DEPARTMENT
MANAGES
CONTROLS
WORKS ON
PROJECT
DEPENDENTS OF
DEPENDENT
HOURS
PLocation
PName
PNumber
Relationship
Name
Sex
B_date
1
N
M
Activate Windows

# Step 2: Mapping of Weak Entity Types



# Step 3: Mapping 1:1 Relationship Types

## Method1: Foreign Key Approach (most popular)

Let R & S be two entity sets

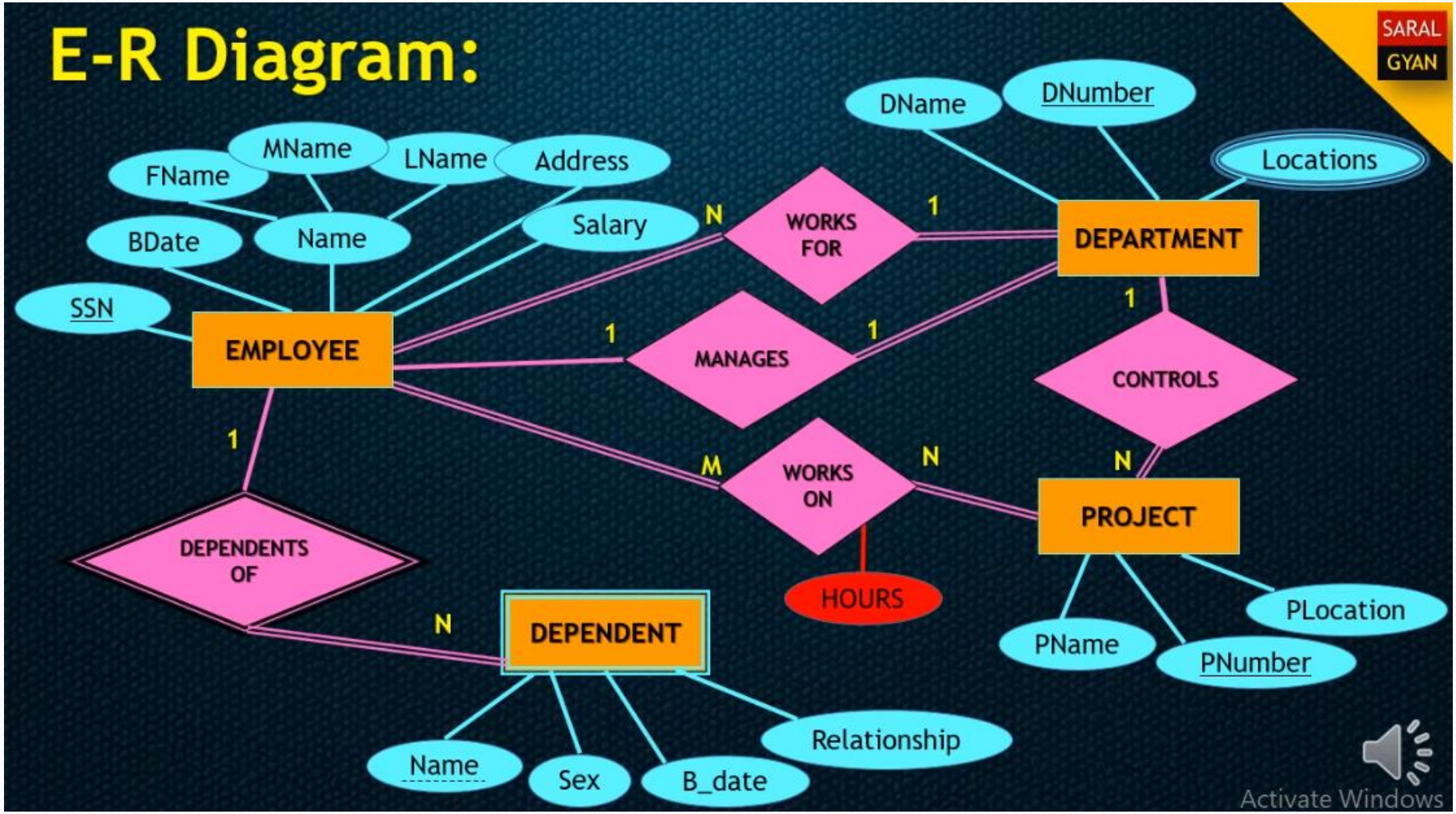
1. Identify the Entity set with total participation (say S)
2. Add primary key of R into S as Foreign Key

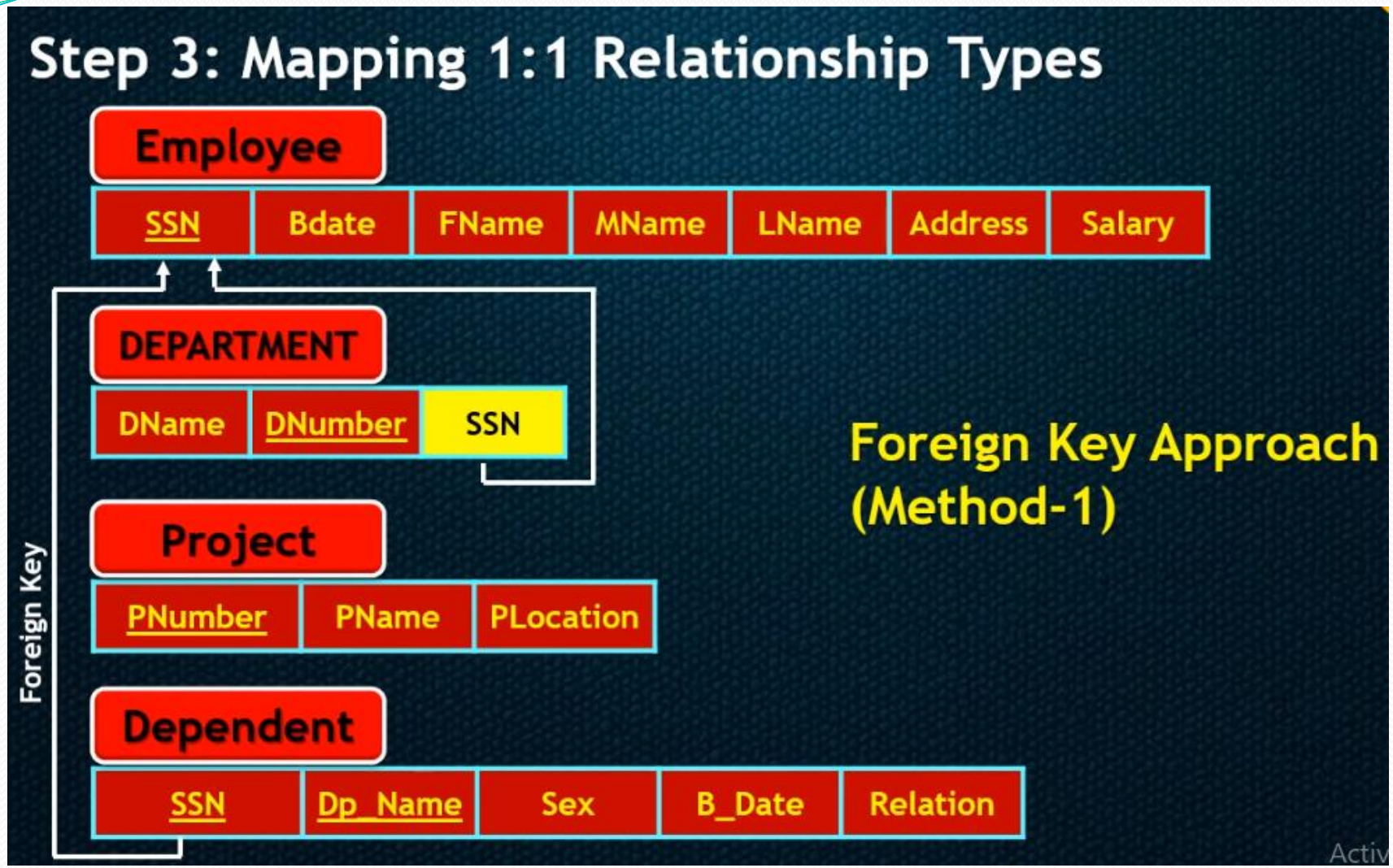
## Method 2: Merged Relation Approach

If both entity sets are having total participation then they can be merged into a single Relation

## Method 3: Cross Reference Approach

Create a third Relation comprising primary key of both entity sets


E-R Diagram:
SARAL GYAN
DNumber
DName
Locations
MName
LName
Address
FName
Name
BDate
Salary
SSN
EMPLOYEE
WORKS FOR
DEPARTMENT
MANAGES
CONTROLS
WORKS ON
PROJECT
DEPENDENTS OF
DEPENDENT
HOURS
PLocation
PName
PNumber
Relationship
Name
Sex
B_date
1
N
M
Activate Windows

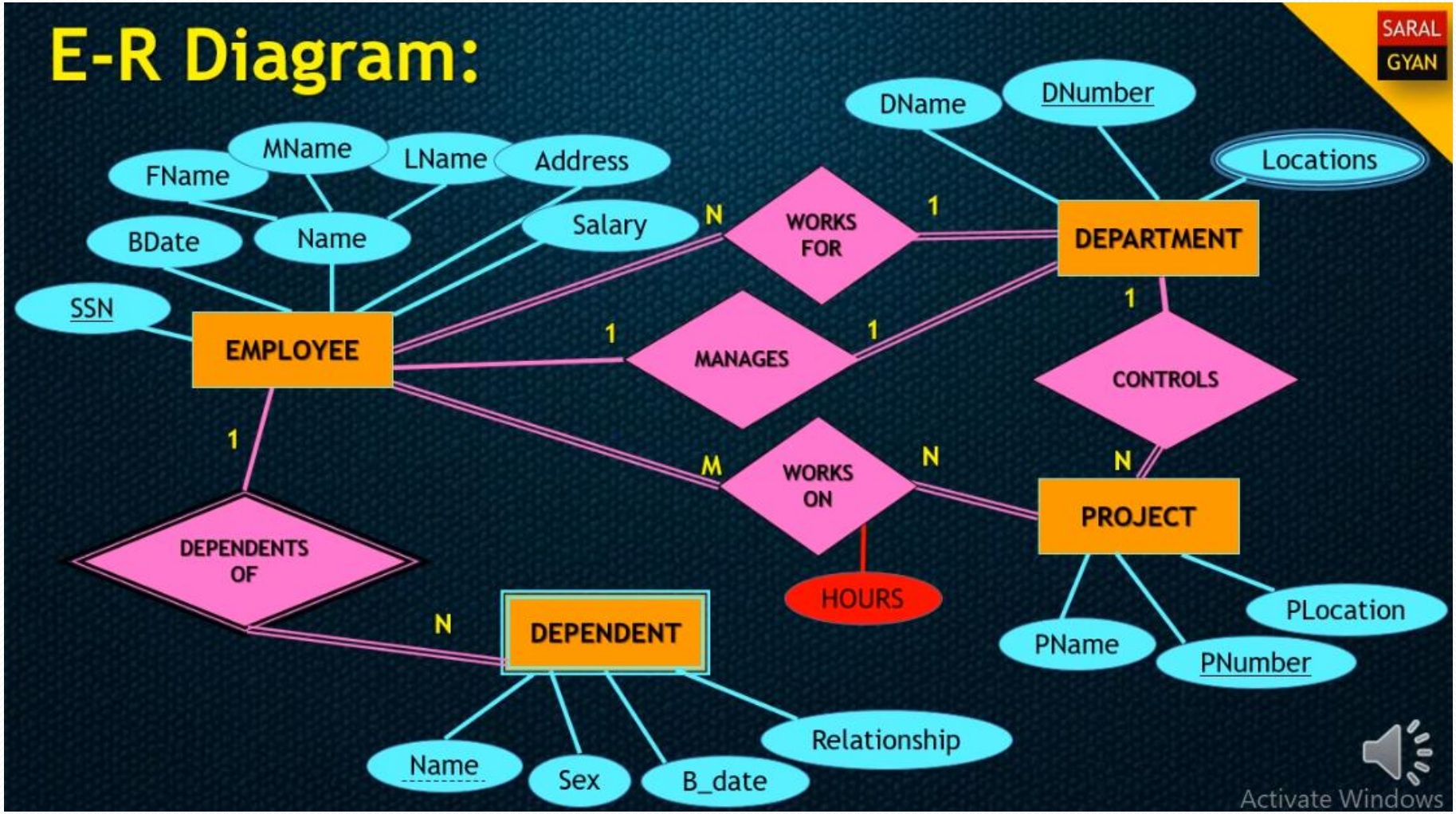

Step 3: Mapping 1:1 Relationship Types
Employee
SSN
Bdate
FName
MName
LName
Address
Salary
DEPARTMENT
DName
DNumber
SSN
Project
PNumber
PName
PLocation
Dependent
SSN
Dp_Name
Sex
B_Date
Relation
Foreign Key
Foreign Key Approach
(Method-1)

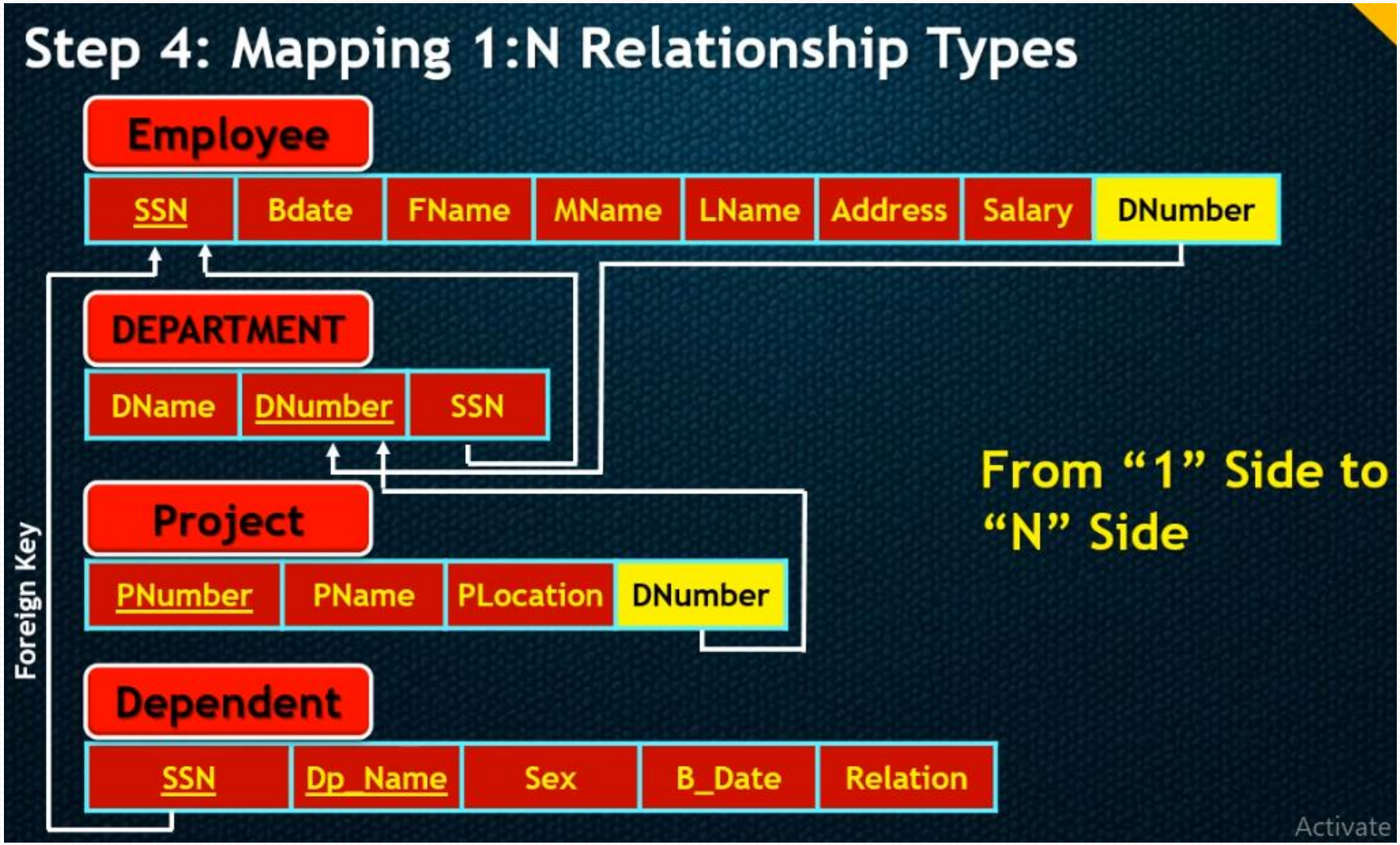
# Step 4: Mapping 1:N Relationship Types

Let R & S be the Entity Sets (with 1:N) Where S is having Total Participation in Relationship

Add Primary Key of R in S as Foreign key

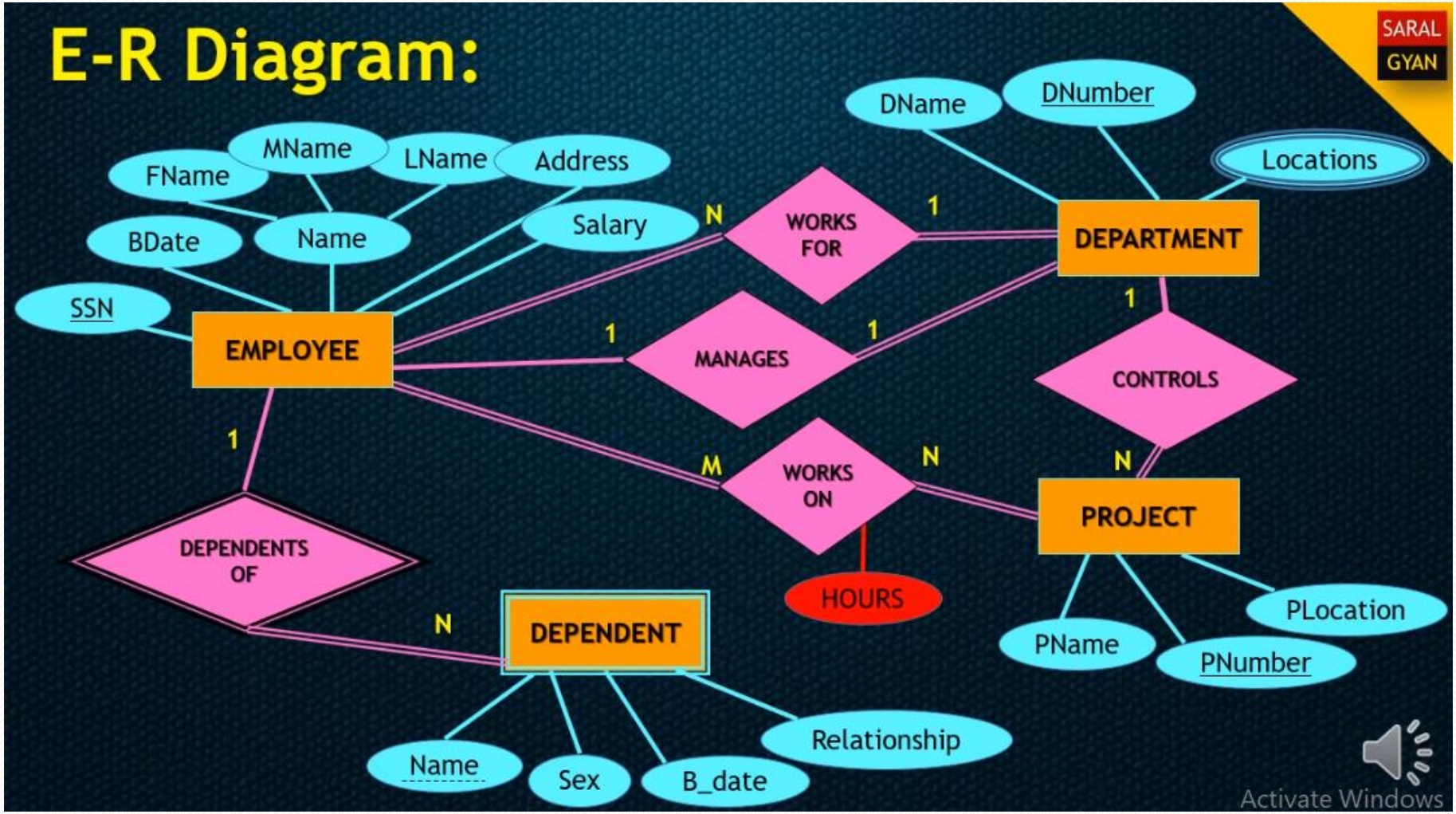
i.e. Primary key from “1” side to “N” side as Foreign Key

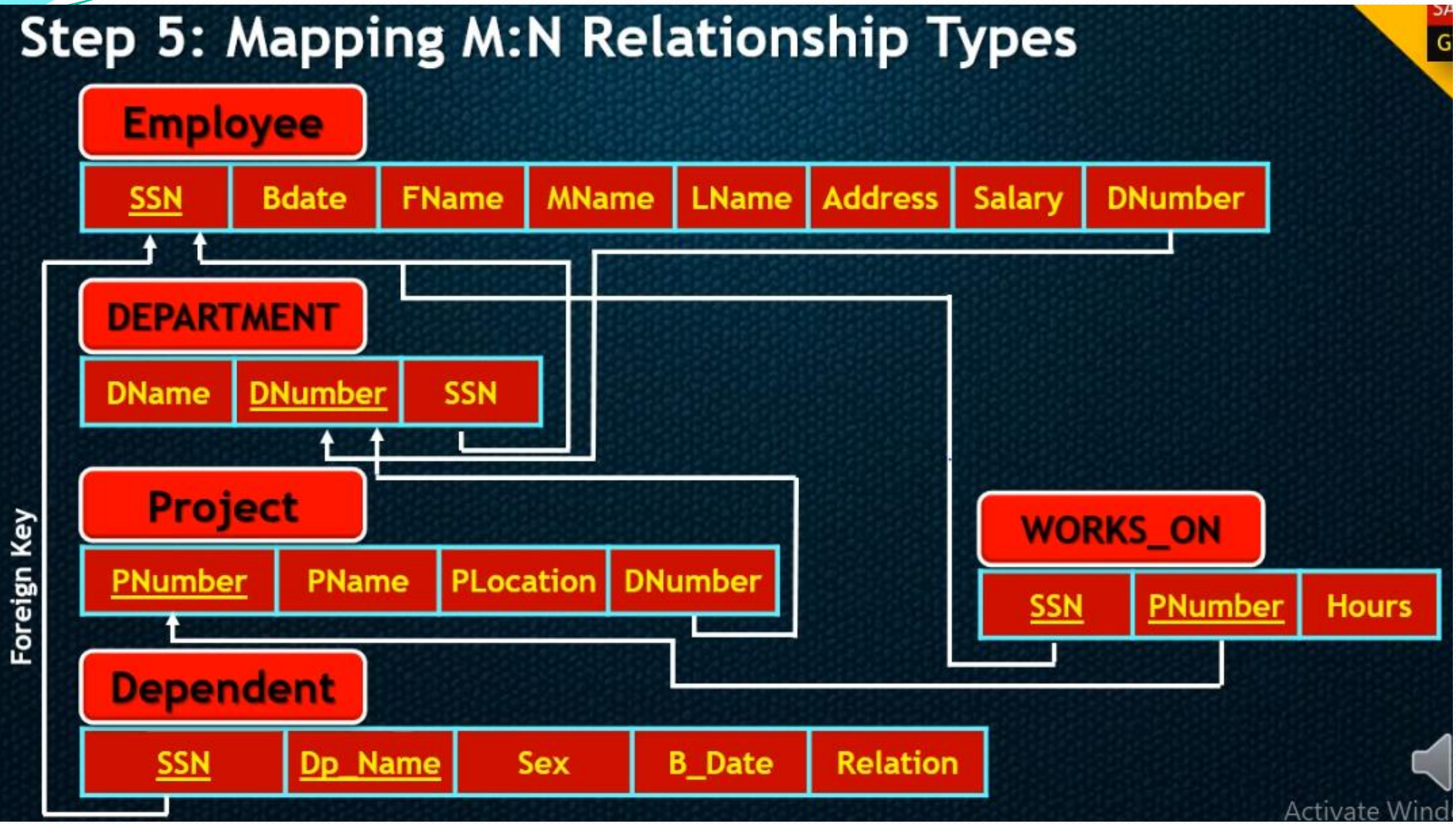

E-R Diagram:
SARAL GYAN
DNumber
DName
Locations
MName
LName
Address
FName
Name
BDate
Salary
SSN
EMPLOYEE
WORKS FOR
DEPARTMENT
MANAGES
CONTROLS
WORKS ON
PROJECT
DEPENDENTS OF
DEPENDENT
HOURS
PLocation
PName
PNumber
Relationship
Name
Sex
B_date
1
N
M
Activate Windows


Step 4: Mapping 1:N Relationship Types
Employee
SSN
Bdate
FName
MName
LName
Address
Salary
DNumber
DEPARTMENT
DName
DNumber
SSN
Project
PNumber
PName
PLocation
DNumber
Dependent
SSN
Dp_Name
Sex
B_Date
Relation
Foreign Key
From "1" Side to "N" Side
Activate

# Step 5: Mapping M:N Relationship Types

Create a third relation containing the primary keys of both the Entity sets and descriptive attribute (if any)


E-R Diagram:
SARAL GYAN
DNumber
DName
Locations
MName
LName
Address
FName
Name
BDate
Salary
SSN
EMPLOYEE
WORKS FOR
DEPARTMENT
MANAGES
CONTROLS
WORKS ON
PROJECT
DEPENDENTS OF
DEPENDENT
HOURS
PLocation
PName
PNumber
Relationship
Name
Sex
B_date
1
N
M
Activate Windows

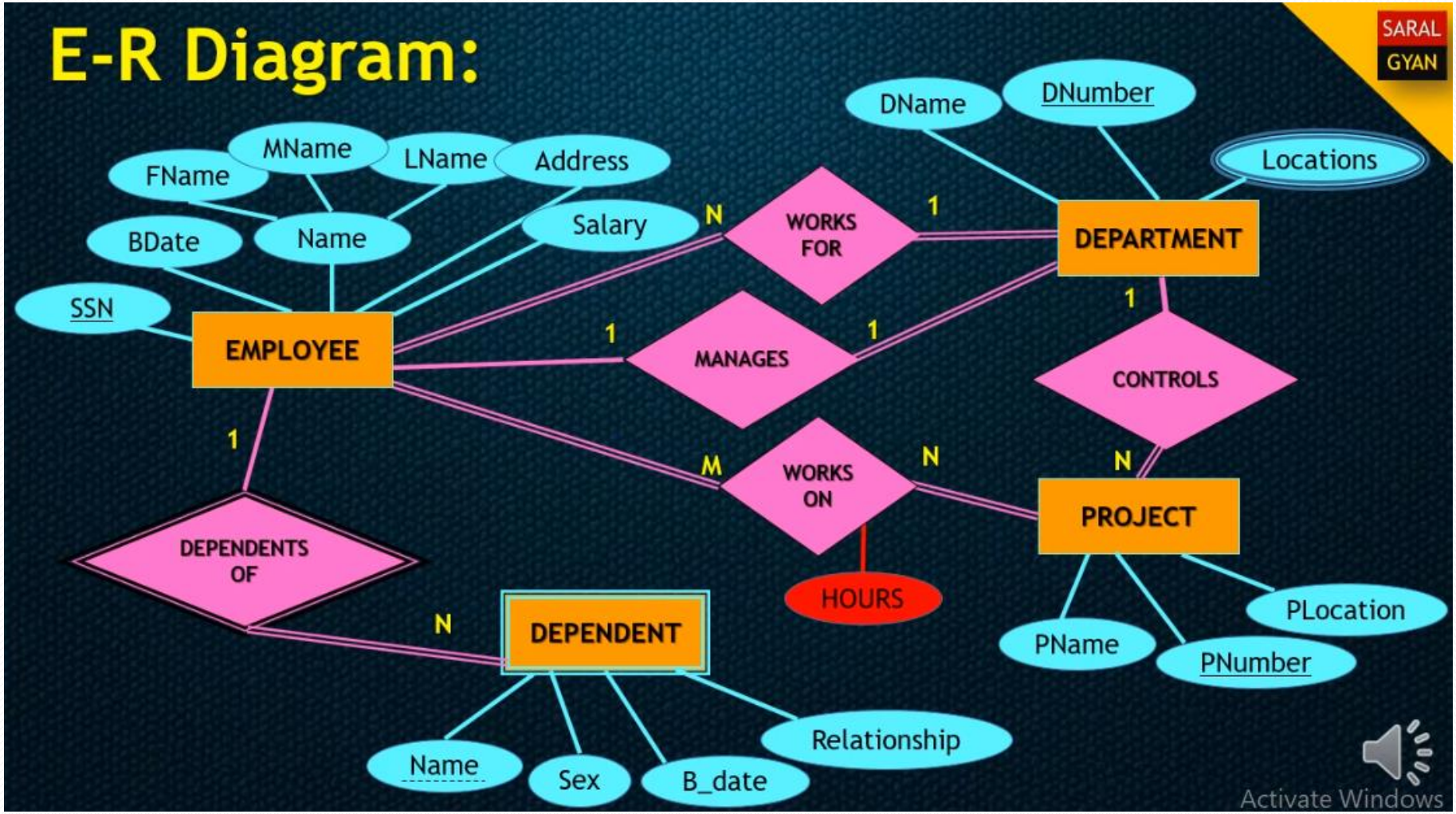

Step 5: Mapping M:N Relationship Types
Employee
SSN
Bdate
FName
MName
LName
Address
Salary
DNumber
DEPARTMENT
DName
DNumber
SSN
Project
PNumber
PName
PLocation
DNumber
WORKS_ON
SSN
PNumber
Hours
Dependent
SSN
Dp_Name
Sex
B_Date
Relation
Foreign Key

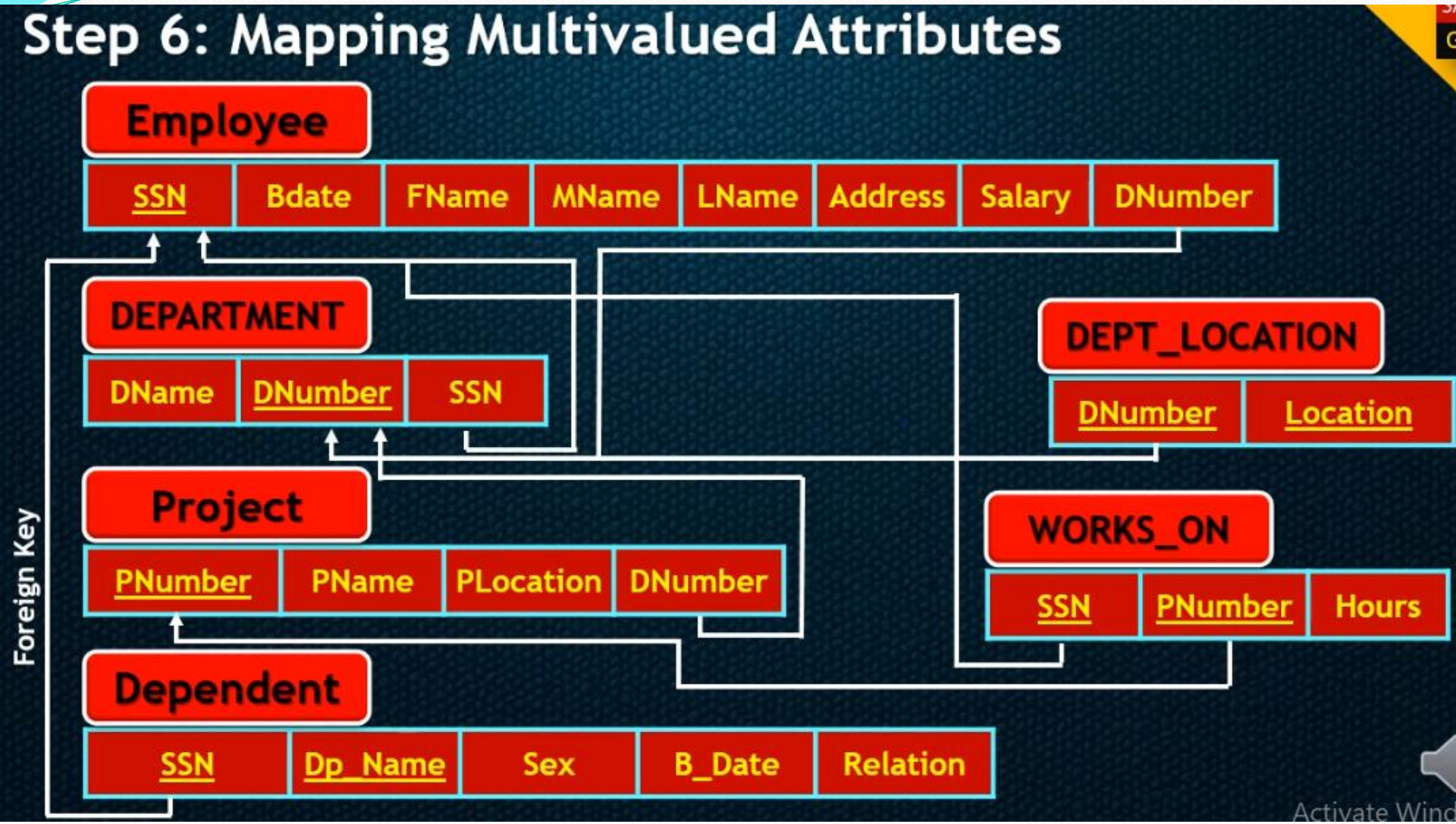
# Step 6: Mapping Multivalued Attributes

For each multivalued Attribute, Create a separate Relation

Add Primary Key of the Entity Set in new Relation as a Foreign Key

The Foreign Key attribute and Multivalued attribute will become Composite Key


E-R Diagram:
SARAL GYAN
DNumber
DName
Locations
MName
LName
Address
FName
Name
BDate
Salary
SSN
EMPLOYEE
WORKS FOR
DEPARTMENT
MANAGES
CONTROLS
WORKS ON
PROJECT
DEPENDENTS OF
DEPENDENT
HOURS
PLocation
PName
PNumber
Relationship
Name
Sex
B_date
1
N
M
Activate Windows


Step 6: Mapping Multivalued Attributes
Employee
SSN
Bdate
FName
MName
LName
Address
Salary
DNumber
DEPARTMENT
DName
DNumber
SSN
DEPT_LOCATION
DNumber
Location
Project
PNumber
PName
PLocation
DNumber
WORKS_ON
SSN
PNumber
Hours
Dependent
SSN
Dp_Name
Sex
B_Date
Relation
Foreign Key

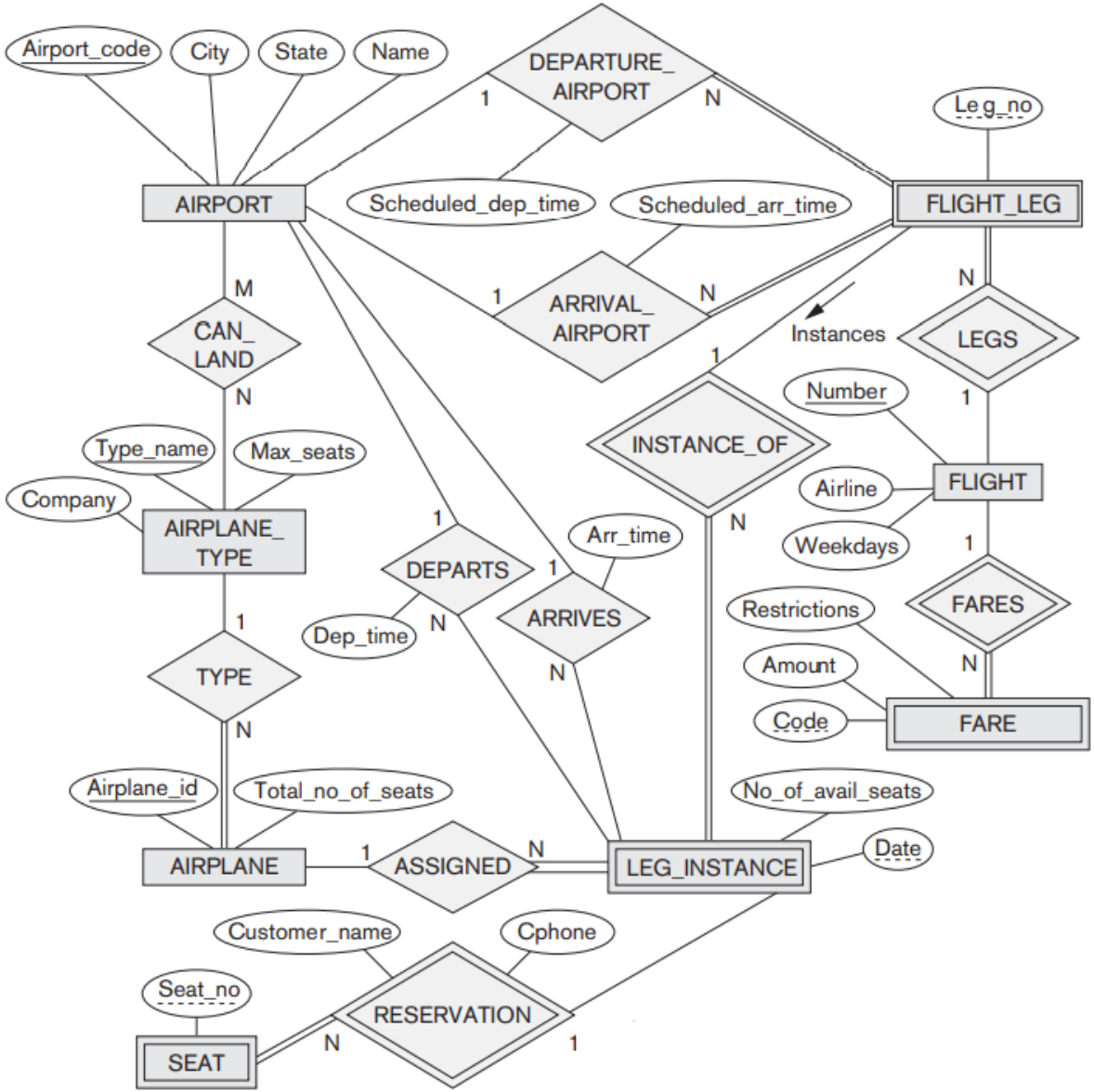
# Step 7: Mapping N-ary Relationships

For each N-ary Relationship, Create a Separate Relation

New Relation will Contain the Primary Keys of All Entity Sets forming N-ary Relationship

**Question: Map the given ER diagram to Relational schema.**

An ER diagram for an AIRLINE database schema.



**Relational database schema**

**AIRPORT**

| Airport_code | Name | City | State |
|---|---|---|---|

**FLIGHT**

| Flight_number | Airline | Weekdays |
|---|---|---|

**FLIGHT_LEG**

| Flight_number | Leg_number | Departure_airport_code | Scheduled_departure_time |
|---|---|---|---|
| | | Arrival_airport_code | Scheduled_arrival_time |

**LEG_INSTANCE**

| Flight_number | Leg_number | Date | Number_of_available_seats | Airplane_id |
|---|---|---|---|---|
| | | Departure_airport_code | Departure_time | Arrival_airport_code | Arrival_time |

**FARE**

| Flight_number | Fare_code | Amount | Restrictions |
|---|---|---|---|

**AIRPLANE_TYPE**

| Airplane_type_name | Max_seats | Company |
|---|---|---|

**CAN_LAND**

| Airplane_type_name | Airport_code |
|---|---|

**AIRPLANE**

| Airplane_id | Total_number_of_seats | Airplane_type |
|---|---|---|

**SEAT_RESERVATION**

| Flight_number | Leg_number | Date | Seat_number | Customer_name | Customer_phone |
|---|---|---|---|---|---|

**Figure 3.8**
The AIRLINE relational database schema.

**Question: Map the given ER diagram to Relational schema.**

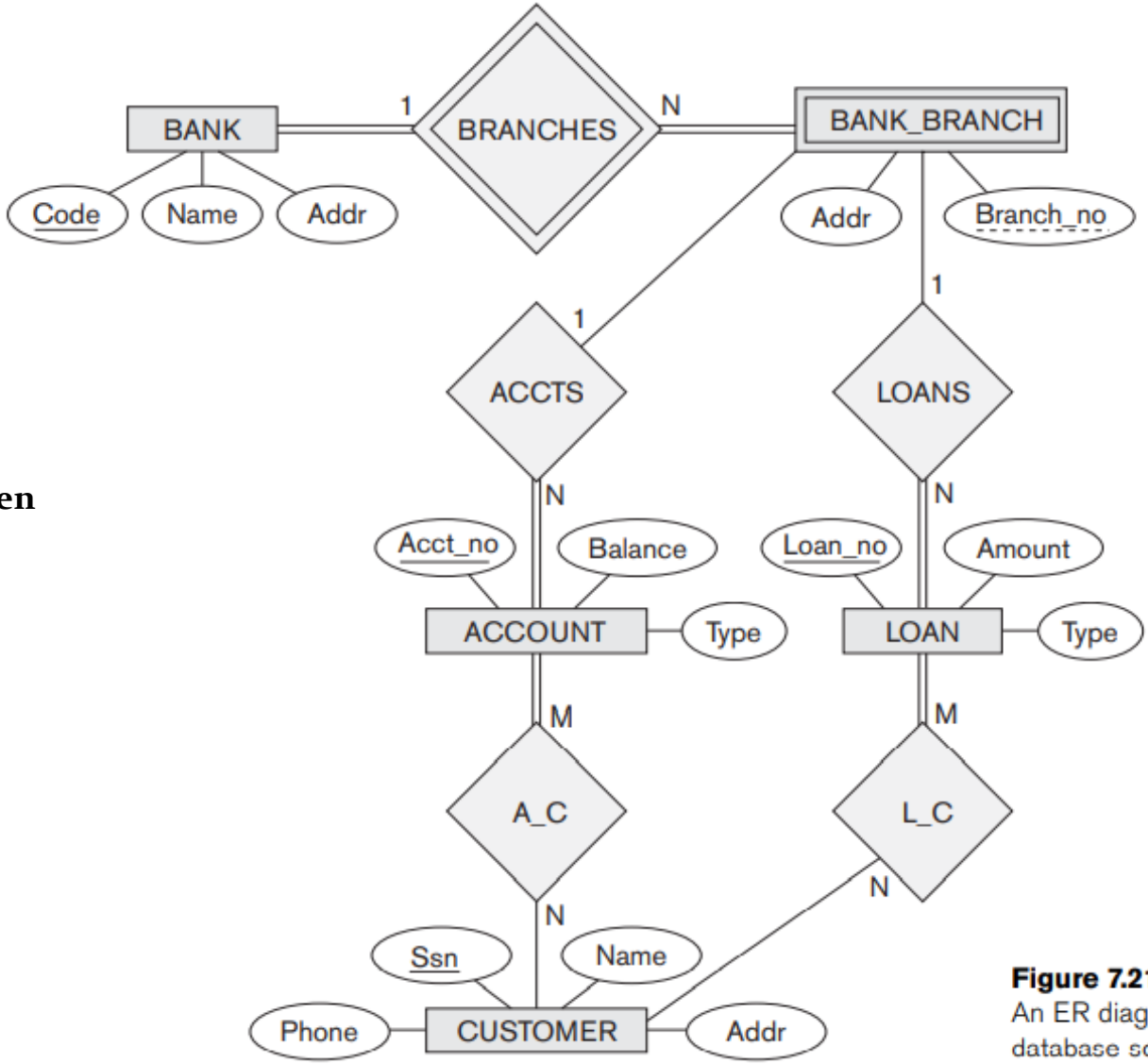
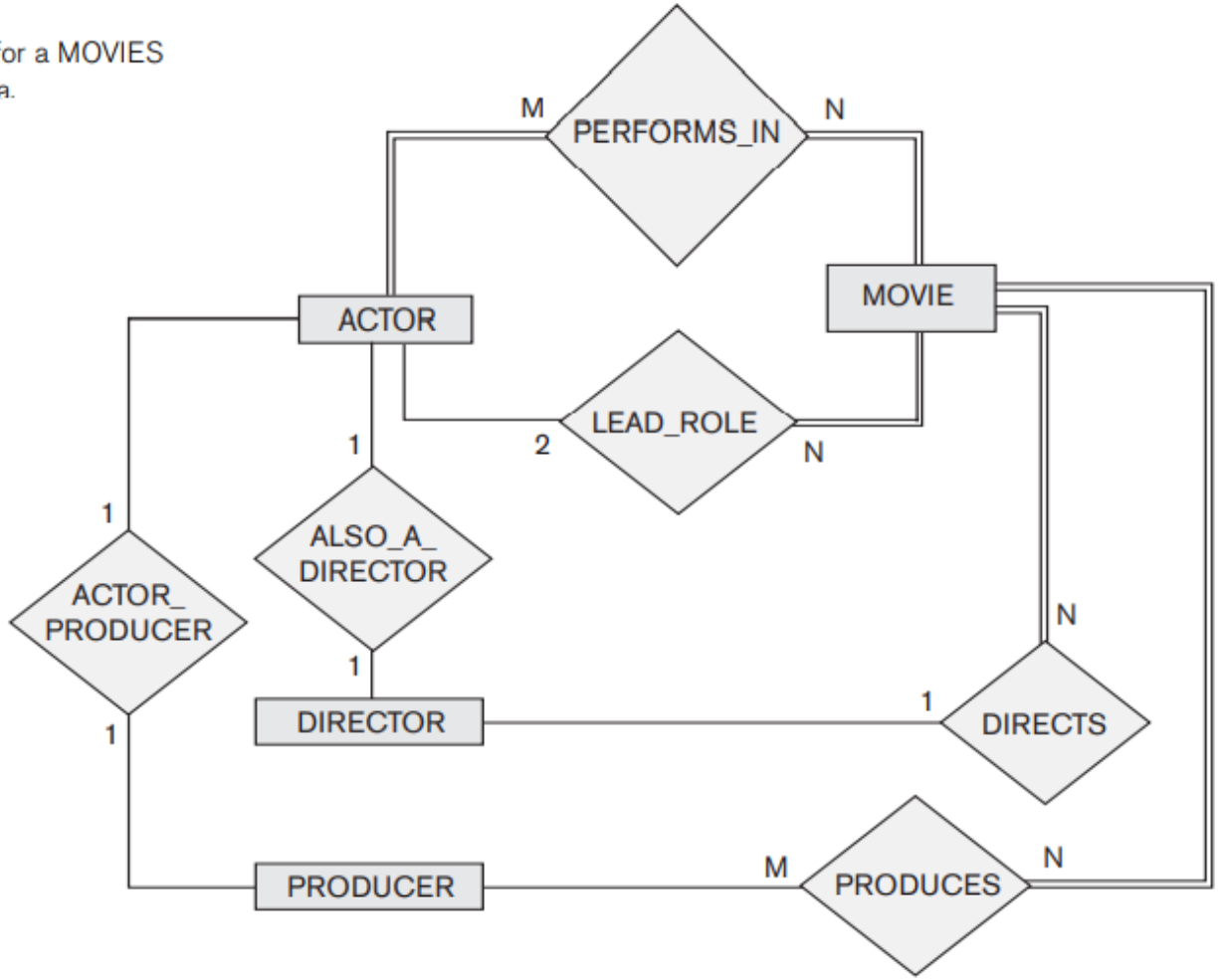



**Figure 7.21**
An ER diagram for a BANK database schema.

**Figure 7.24**
An ER diagram for a MOVIES database schema.




**Question: Map the given ER diagram to Relational schema.**