

# **Database Management Systems**

# Introduction

- **Data:**
- It is the measurement of any measurable physical or conceptual entity or process or fact.
- It has no meaning in itself.
- It has implicit meaning.
- Its context or environment of measurement gives its implicit meaning.
- Example:
  - $\{40.46, 56.32, 75.88\}$
  - weight of students =  $\{40.46, 56.32, 75.88\}$

# Introduction

- **Information:**

- It is the processed and interpreted data.
- Its meaning is explicit.
- Example:
  - weight of any 3 students = {40.46, 56.32, 75.88}
  - The weight of the heaviest among the 3 students = 75.88
  - Average weight of the above 3 student = 57.56
  - Rank of NIT-R in 2018-19 = 13
  - Campus Placement percentage of CSE NIT-R = 99%

# Introduction

- **Database:**
- Collection of related data representing some part of real world.
- Example:
- Bank Database, University Database, Facebook Database.
- Election Database, Income Tax Database etc.
- Each database has a definite use or application.

# Introduction

- **Database Management System:**
- Its is a software that is used to define, create, maintain and control access to database.
- Example: Oracle, MySQL, DB2, Postgress etc.

# Introduction

- **Database System:**
- It is a computerized system (hardware + software) that is used to store, retrieve and manipulate data in a database.
- **Hardware requirements:**
  - Secondary Storage like Disks and Tapes
  - Processors and associated Primary memory
- **Software requirements:**
  - DBMS and Application programs

# Introduction

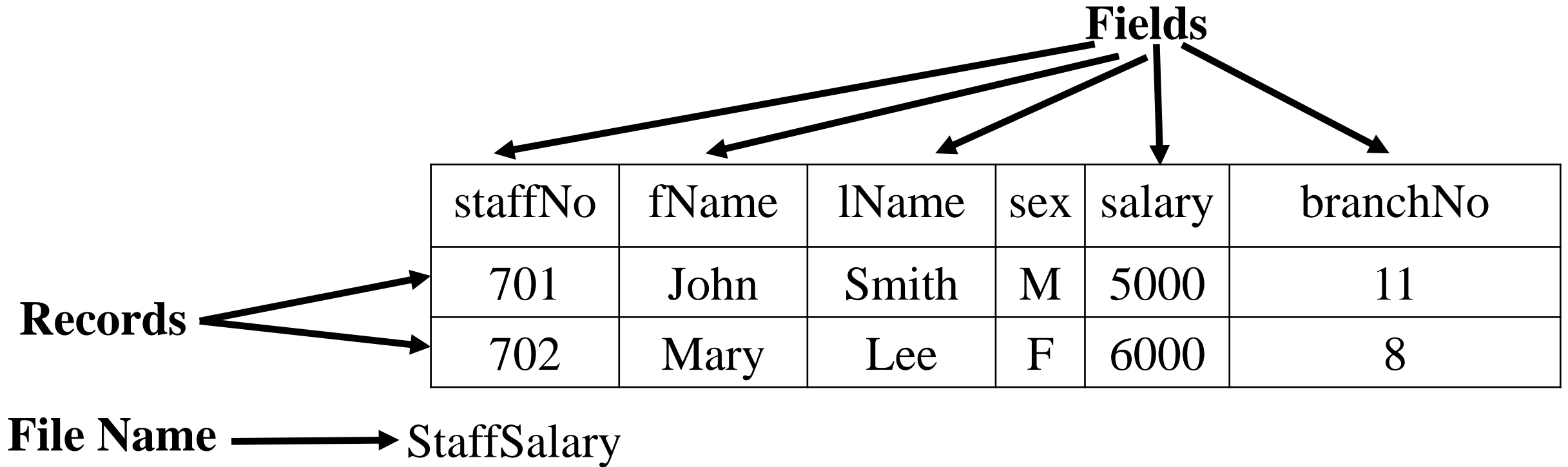
- **Two ways of storing, retrieving and managing data:**
  - 1. File based System
  - 2. DBMS based System

# Introduction

- **1. File based System:**
- **File:**
- It is a collection of records.
- Record is a set of logically related data.
- A file is represented as
- file-name(list of fields)
- Example:
- StaffSalary(staffNo, fName, lName, sex, salary, branchNo) in **Payroll Department**
- Staff(staffNo, fName, lName, position, sex, dateOfBirth, branchNo) in **Personnel Department**



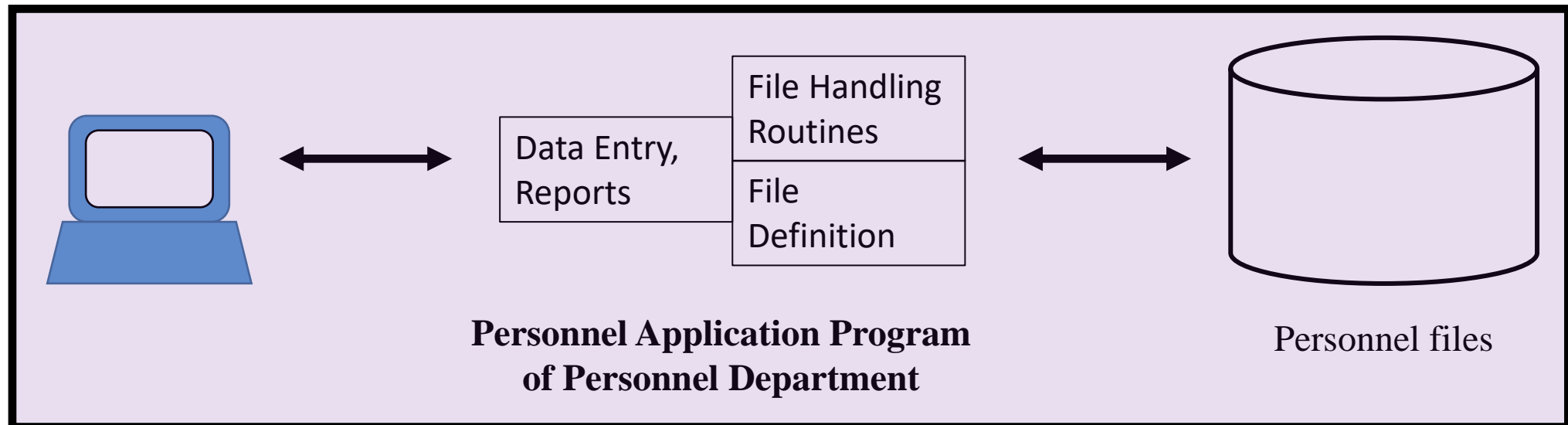
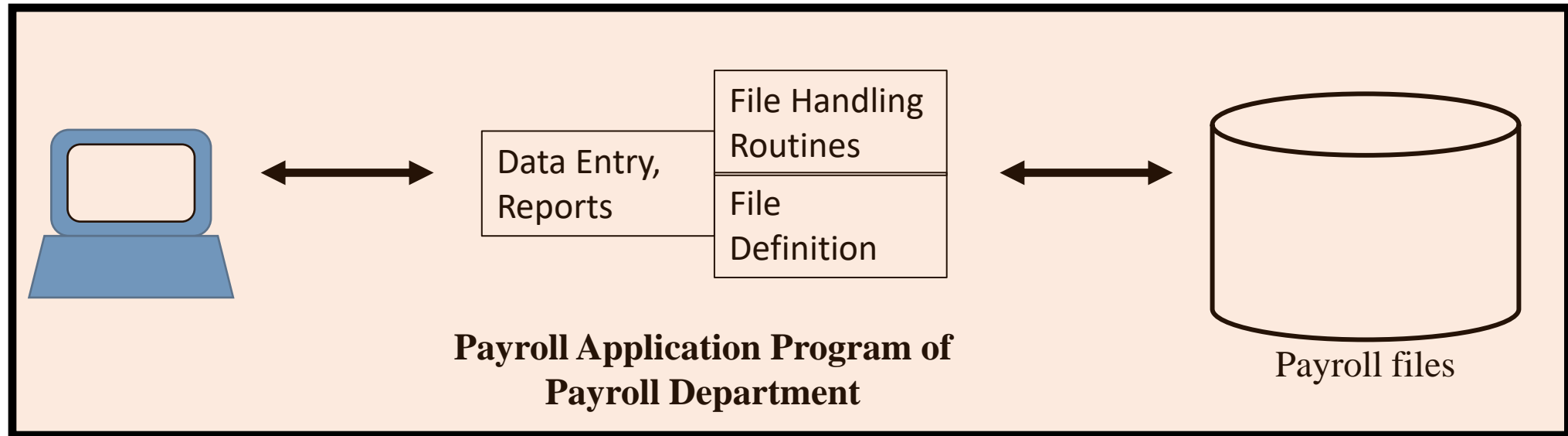
# Introduction



**Conceptual Structure of a file**

# Introduction

## File based Processing:



# Introduction

- **Limitations of File based System:**
- **Data Redundancy and Inconsistency:**
- Same data may be present in different files (causing more space cost), and may lead to inconsistent data when one instance of a data is modified, but its other instances in other files are not modified.
- **Atomicity Problems:**
- Sometimes it is needed to execute ALL or NONE of a group of queries / instructions. In file-processing system, maintaining this type of atomic nature of execution is not possible.

# Introduction

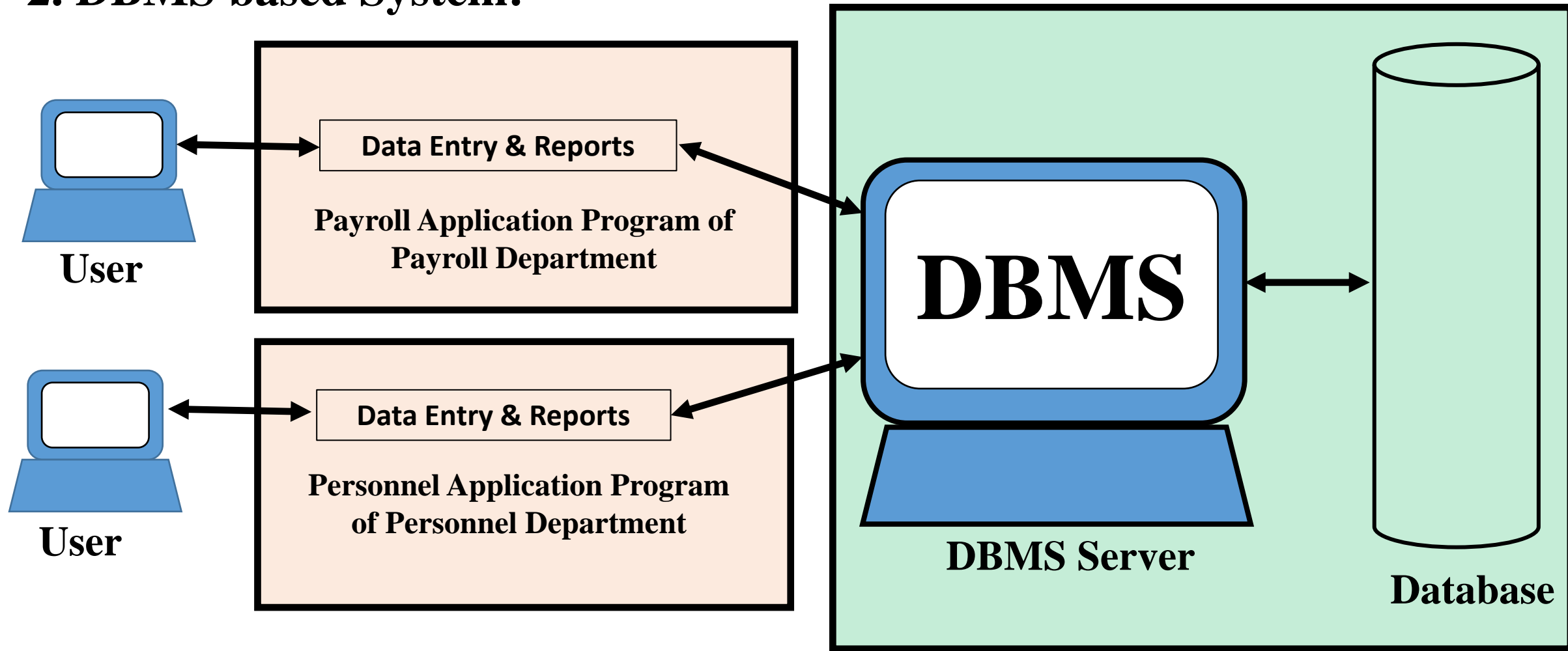
- **Integrity Problems:**
- Data stored may sometimes require a natural constraint (e.g. AGE cannot be a negative integer) or business-specific constraints (e.g., GRADE of a student can be E / A / B / C / D / F / I , and no other character) to be applied on them.
- Managing these integrity constraints on creation and during each modification of data is not possible through file-processing system.
- **Difficulty in Accessing Data:**
- In file-processing system, users are not abstracted from the physical access of data. Hence users should know the location of files to retrieve data. In course of time, if file structure changes, the users face trouble.

# Introduction

- **Concurrent Access Anomalies:**
- In desired scenario of multiple users accessing/altering same data, file-processing system may fail. Locks cannot be created between user access causing the problem.
- **Data Isolation:**
- Isolating a desired data from more than one file is challenging in file-processing system.
- **Security Problems:**
- All users are not designated to access all data. There is a logical grant / revoke of 'right of access' by business policy makers, which a database should adhere to. In file-processing system, there is no such scope.

# Introduction

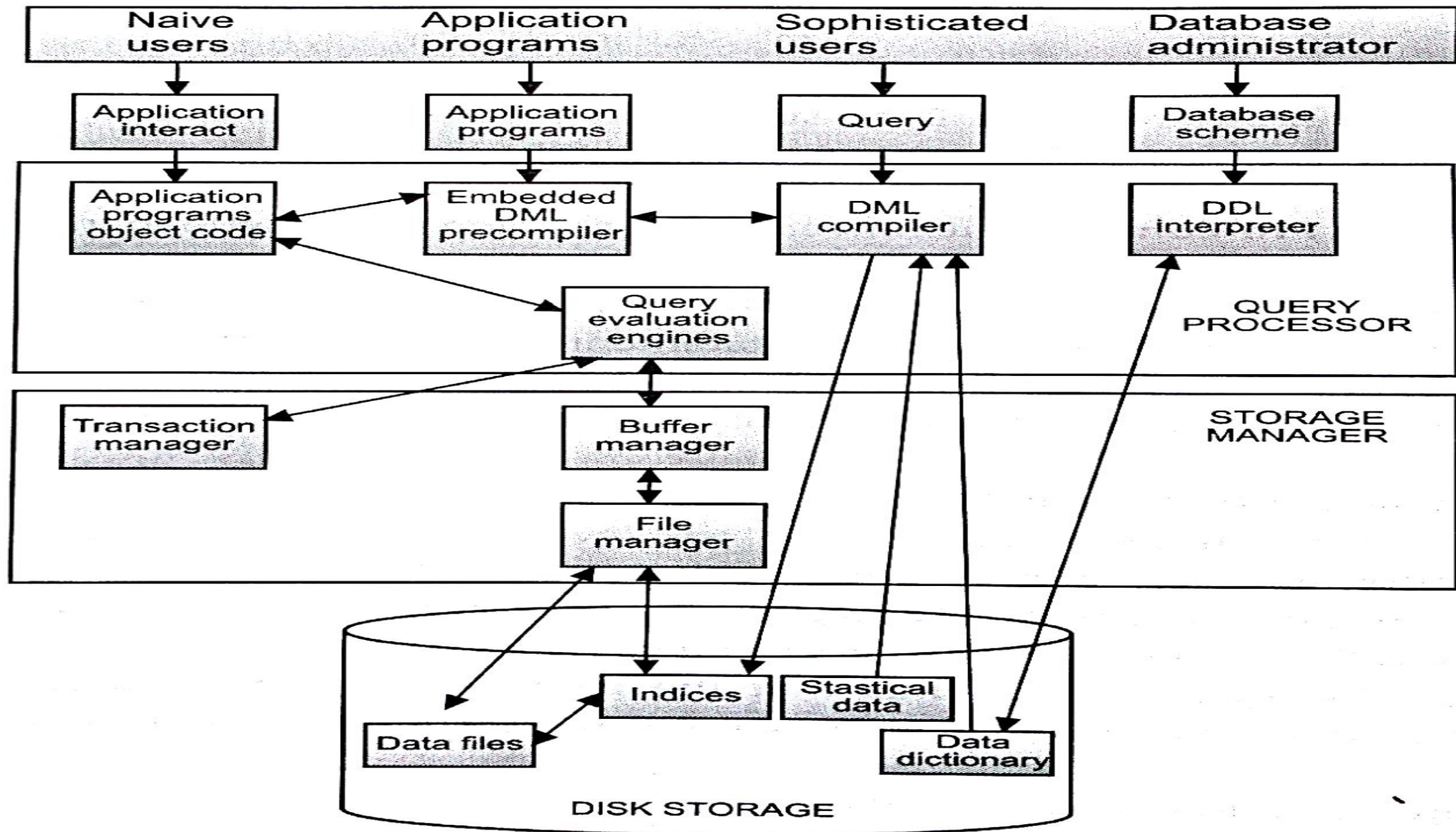
## 2. DBMS based System:



**DBMS based Processing:**

# Introduction

- **Components of DBMS:**





# Introduction

- **2. DBMS based System:**

- **Advantages:**

- Data duplication eliminated
- Controlled redundancy.
- Restricting unauthorized access.
- Providing backup and recovery.
- Providing Multiple User Interface.
- Enforcing Security and integrity constraints.
- Centralized Control and Data quality enhanced.

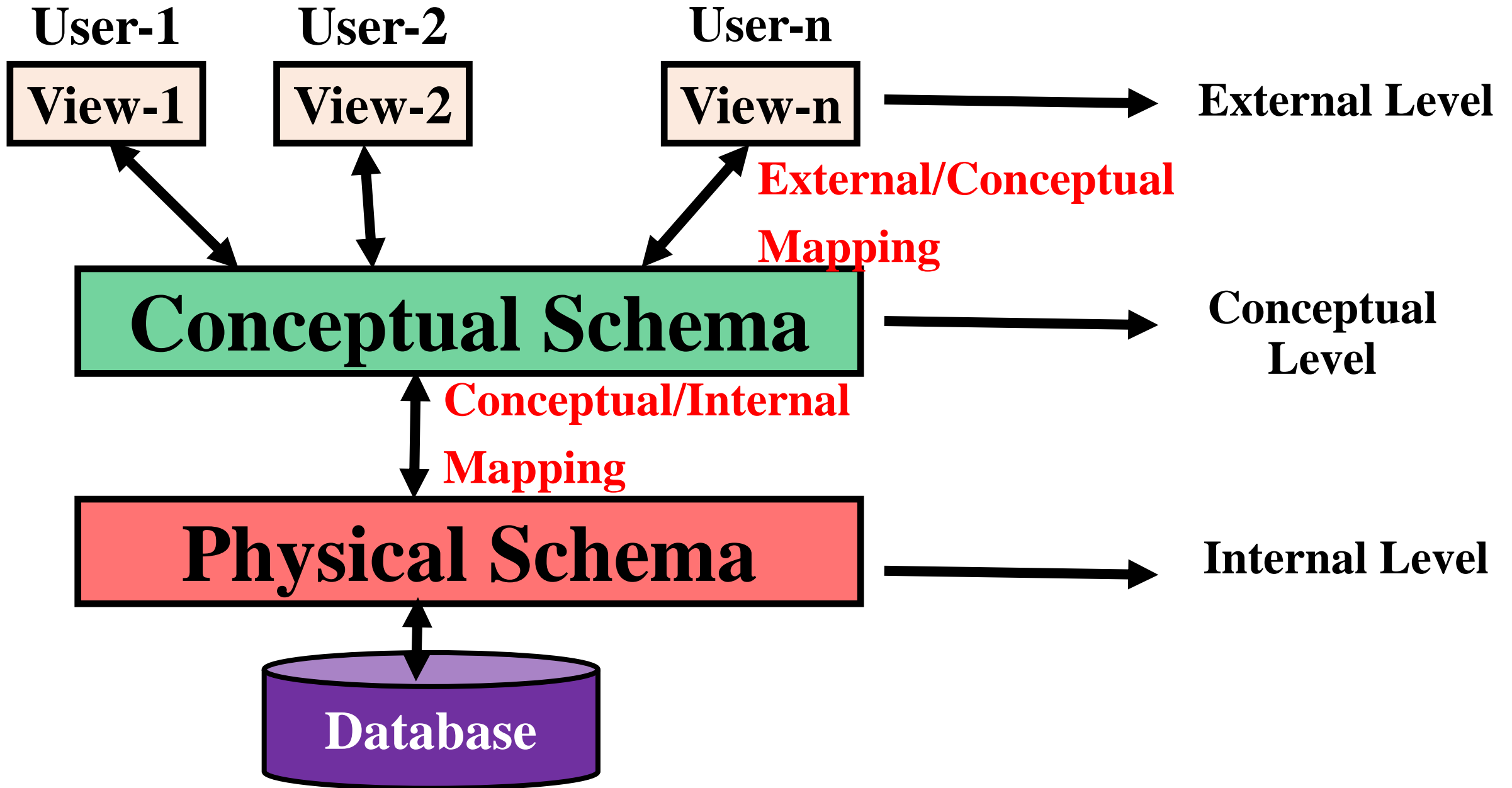
# Introduction

- **2. DBMS based System:**

- **Disadvantages:**

- Problems associated with centralization
- Cost of software/hardware and migration
- Complexity of backup and recovery
- Can't be used for many real time embedded systems

- **Levels of Abstraction in Database:**



# Levels of Abstraction in Database:

## External Level:

- It contains user views of the database.
- The user's view describes only that part of the database which is relevant to a user.
- Different user views have different representation of the same data according to user's need or comfort.
- It also provides a level of security by allowing certain group of users to access only certain types of data and not the entire database.
- Example:
  - Viewing one's bank transaction, balance etc.
  - Viewing one's online purchase order, exam result etc.

# Levels of Abstraction in Database:

## Conceptual Level:

- It contains the description of all the data in the database of an organization and the relationships among the data.
- It contains the logical structure of the entire database.
- It represents:
  - All entities, attributes and their relationships
  - Constraints of the data
  - Semantic information about the data
  - Security and Integrity information
- It should not contain any storage details about the data
- Example:
  - Student data is logically stored using relation schema as **Student**(RollNo, Name, Branch, DoB, Sex)

# Levels of Abstraction in Database:

## Internal Level:

- It describes how the data is physically stored in the database.
- It is concerned with:
  - Storage space allocation for data and indexes,
  - Record descriptions for storage
  - Record Placement
  - Data compression and encryption techniques

# **Levels of Abstraction in Database:**

## **Data Model:**

- It is a collection of high-level data description constructs that hide many low level storage details.
- Its description is more closer to how data is stored in a database, than to how a user thinks of the data with respect to enterprise it represents.
- Example: Relational Data Model

## **Semantic Data Model:**

- It is a more abstract, high-level data model that conceptually describes a real world enterprise more accurately in terms of how a user thinks of the enterprise.
- Example: ER diagram

# **Levels of Abstraction in Database:**

## **Schema:**

- It is a description of data using a data model.

## **Database Schema:**

- It is the description of a database using a data model.

## **Database Instance:**

- It is the data in the database at a specific given instance of time.



# **Levels of Abstraction in Database:**

## **Data Independence:**

- It is the main objective of the three-level architecture.
- Schema changes in the upper levels are unaffected by schema changes in the lower levels.

## **Types of Data Independence:**

1. Logical Data Independence
2. Physical Data Independence

# **Levels of Abstraction in Database:**

## **1. Logical Data Independence**

- It refers to the immunity of the external schemas to the changes in the conceptual schema.

## **2. Physical Data Independence**

- It refers to the immunity of the conceptual schemas to the changes in the physical schema.

# Database Languages:

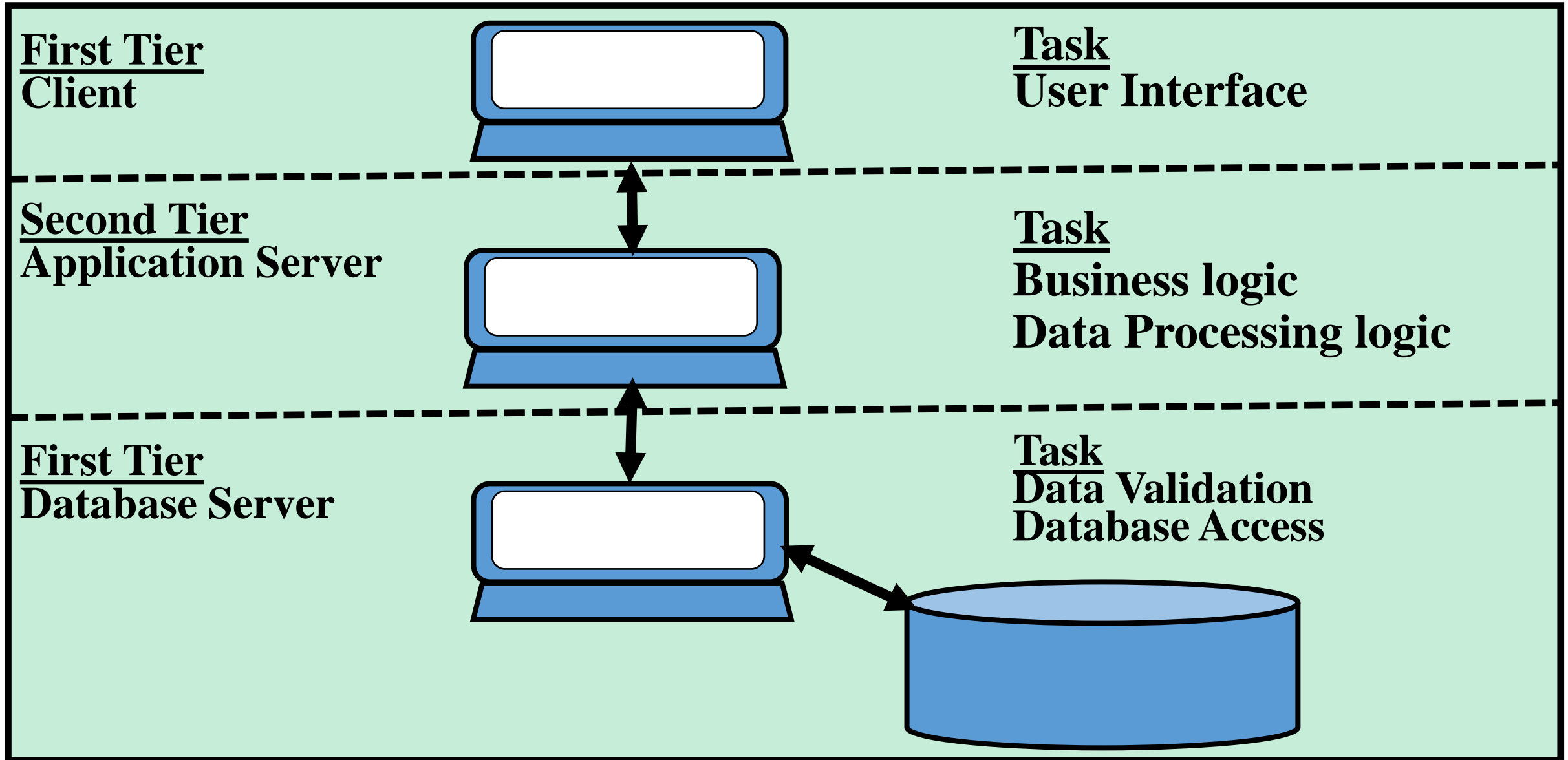
## 1. Data Definition Language (DDL)

- It is used to create conceptual and external schemas.

## 2. Data Manipulation Language (DML)

- It is used to manipulate the data stored in a database.
  - **Types of DML:**
    - **2.1 Procedural DML:**
      - describes how the data is to be obtained.
    - **2.2 Non-Procedural DML:**
      - describes only what data has to be obtained.

# Three-tier DBMS Architecture:



# Relational Model:

## Relation:

- It is a 2D table with set of rows and columns
- It present real world objects
- Columns(field) represent the attributes of real world objects
- Rows(tuple/record) represent instances of real world objects
- **Example:**

### Student

<b>RollNo</b>	<b>Name</b>	<b>Address</b>	<b>Sex</b>	<b>MobileNo</b>
2018001	John	Rourkela	M	98764000
2018022	Mark	Kolkata	M	78654387
2017061	Mary	Delhi	F	78905325

# **Relational Model:**

## **Degree of a Relation:**

- It is the number of attributes in a relation

## **Cardinality of a Relation:**

- It is the number of tuples in a relation.

## **Domain of an Attribute:**

- It is a set of values from which an attribute can be assigned with any one of its values.

## Relation Schema:

- Schema = logical representation, model or blue-print
- It is the logical representation of a relation
- It is represented by a relation schema name defined by a set of attribute and domain name pairs.

Example:

Student(RollNo: **number**, Name: **char**, CGPA: **number**, DOB: **date**)

## Relation Instance:

- It is the set of the rows of a relation at a specific instance of time.

## Relational Database Schema:

- It is the set of relation schemas forming the database

## **Structured Query Language (SQL) of Oracle:**

- It is used to create tables
- It is used to insert, delete, update tables.
- It is used to retrieve data from the tables.



## Command to Create Relation/Table:

- commands are **not case-sensitive**
- **create table** <table-name>(column\_name data type,.....);
- Example:
- **create table** student(rollno number, name char(20), CGPA number);

## Command to display the attributes and their domains of a table:

- **describe** <table-name>;
- Example:
- **describe** <student>;

## Command to insert values into a table:

- **insert into** <table-name(column names)> **values** (value1,...);
- Example:
- **insert into** student(rollno, name , CGPA) **values**(2017001, 'john', 8.5);
- **insert into** student(rollno, name , CGPA) **values**(2017024, 'mary', 7.5);
- **---OR-----**
- **insert into** <table-name> **values** (value1,...);
- Example:
- **insert into** student **values**(2017001, 'john', 8.5);

## Student

<b>rollno</b>	<b>name</b>	<b>CGPA</b>
2017001	john	8.5
2017024	mary	7.5

## Command to display all the rows & columns of a table:

- **select \* from** <table-name>;
- Example:
- **select \* from** student;

## Student

rollno	name	CGPA
2017001	john	8.5
2017024	mary	7.5

## Command to display all columns & specific rows of a table:

- **select \* from** <table-name> **where** <conditions>;
- Example:
- **select \* from** student **where** student.rollno=2017001;
- **select \* from** student **where** student.name='john';
- **select \* from** student **where** student.CGPA>7;

## Student

rollno	name	CGPA
2017001	john	8.5
2017024	mary	7.5

### Command to display specific columns & all rows of a table:

- **select** <column name(s)> **from** <table-name>;
- Example:
- **select** rollno, name **from** student;

### Command to display specific columns & rows of a table:

- **select** <column name(s)> **from** <table-name> **where** <conditions>;
- Example:
- **select** rollno, name **from** student **where** student.rollno=2017024;

## Command to delete a row(s) from a table:

- **delete from** <table-name> **where** <conditions>;
- Example:
- **delete from** student **where** student.rollno=2017001;

## Command to delete all rows of a table:

- **truncate table** <table-name>;
- Example:
- **truncate table** student;

### Student

rollno	name	CGPA
--------	------	------

## **Command to update the column value(s) in a table:**

- **update** <table-name> **set** <col-1=val-1,col-2=val-2,...>  
**where** <conditions>;
- Example:
- **update** student **set** student.CGPA=9.98 **where** student.roll=2017024;

## **Command to drop a column from a table:**

- **alter table** <table-name> **drop** (col-1);
- Example:
- **alter table** student **drop** (CGPA);

## **Command to drop multiple columns from a table:**

- **alter table** <table-name> **drop** (col-1,col-2...);
- Example:
- **alter table** student **drop** (name,CGPA);

## **Command to add a column to a table:**

- **alter table** <table-name> **add** (col-1 data type, col-2 data type);
- Example:
- **alter table** student **add** branch char(20);

## **Command to modify the column(s) in a table:**

- **alter table** <table-name> **modify** (col-1 data type, ...);
- Example:
- **alter table** student **modify** (rollno char(10));

## **Command to Rename a column in a table:**

- **alter table** <table-name> **rename column** col-1 to col-2;
- Example:
- **alter table** student **rename column** rollno to roll\_number;



## Command to rename a table:

- **alter table** <table-name> rename to <table-name-2>;
- Example:
- **alter table** student **rename to** student\_details;

## Command to delete a table:

- **drop table** <table-name>;
- The dropped tables retain their space in disk
- They are accessed via 'recycle bin'
- Example:
- **drop table** student;

## **Command to remove a dropped table from recycle bin:**

- **purge table** <table-name>;
- Example:
- **purge table** student;

## **Command to remove all dropped tables from recycle bin:**

- **purge** recycle bin;

## **Command to restore a dropped table from recycle bin:**

- **flashback table** <table-name> **to before drop**;
- Example:
- **flashback table** student **to before drop**;

## **Properties of a Relation:**

- The names of relations are distinct
- The names of the attributes in a relation is distinct
- Each attribute must be assigned with a single value from its domain
- The order of attributes does not matter
- Each tuple of a relation is distinct
- The order of tuples does not matter.

## **Super key of a Relation:**

- It is a single or a set of attributes that uniquely identifies a tuple within a relation.

## **Candidate key of Relation:**

- It is a super key with minimal number of attributes
- There can be many candidate keys for a relation

## **Primary key of Relation:**

- It is the candidate key that has been selected for a relation.
- Other candidate keys are called alternate keys.

## **Foreign key:**

- It a single or a set of attributes of a relation (child relation) that matches the primary/candidate key of another relation (parent relation) or the same relation.
- It is used to link two tables.

<u>rollno</u>	name
2017001	john
2017024	mary

Student

<u>Course-ID</u>	C-name
CS3002	DBMS
CS3015	OS

Course

<u>rollno</u>	<u>Course-ID</u>	Grade
2017001	CS3002	A
2017001	CS3015	B
2017024	CS3002	B
2017024	CS3015	A

Grade

## **Null Value:**

- Not applicable
- Unknown

# Relational Model Constraints:

## 1. Domain Constraint:

- It specifies that for each tuple the value of each of its attribute, A, must be an atomic value from the domain of A.

## 2. Entity Integrity:

- It specifies that in a base relation, no attribute(s) of a primary key can be null.

## 3. Referential Integrity:

- If a **foreign key** exists in a child relation, then either the foreign key value(s) must match the primary key value(s) of the parent relation or the foreign key value(s) must be null.

<b><u>FA-ID</u></b>	<b>Faculty-Advisor-name</b>
<b>FA976</b>	<b>Smith</b>
<b>FA764</b>	<b>Tomas</b>

**Faculty\_Advisor**

<b><u>rollno</u></b>	<b>name</b>	<b>FA-ID</b>
<b>2017001</b>	<b>john</b>	<b>FA976</b>
<b>2017021</b>	<b>james</b>	<b>FA764</b>
<b>2017024</b>	<b>mary</b>	<b>FA764</b>
<b>2017024</b>	<b>rose</b>	<b>NULL</b>

**student**



# Command to Specify Primary & Foreign Key for a table:

- **Create table** <table-name>  
( col-1 data-type,  
col-2 data-type, ...  
**constraint** <constraint\_name> **primary key** (col-1,col-2,...)  
);
- Example:
- **Create table** student  
(rollno **number**,  
name **char**(20),  
**constraint** student\_pk **primary key** (rollno)  
);

## **Command to Specify Primary Key using alter table command:**

- **Alter table** <table-name>  
**add constraint** <constraint\_name> **primary key** (col-1,col-2,...);
- Example:
- **alter table** student  
**add constraint** student\_pk **primary key** (rollno);

## **Command to drop Primary Key using alter table command:**

- **Alter table** <table-name> **drop constraint** <constraint\_name> ;
- Example:
- **alter table** student **drop constraint** student\_pk;

## **Disable Primary Key using alter table command:**

- **Alter table** <table-name>  
**disable constraint** <constraint\_name>;
- Example:
- **alter table** student  
**disable constraint** student\_pk;

## **Enable Primary Key using alter table command:**

- **Alter table** <table-name>  
**enable constraint** <constraint\_name> ;
- Example:
- **alter table** student  
**enable constraint** student\_pk;

# Command for Foreign Key constraint:

- **create table** <table-name>  
(  
  col-1 data-type,  
  col-2 data-type,  
  col-3 data-type,  
  **constraint** <foreign\_key\_name> **foreign key** (col-2)  
  **references** <parent\_table\_name(col-2)>,  
  **constraint** <foreign\_key\_name> **foreign key** (col-3)  
  **references** <parent\_table\_name(col-3)>,  
);

# Command for Foreign Key constraint:

- Example:

```
create table grade
```

```
(  roll number,
```

```
   courseID char(20),
```

```
   grade char(10),
```

```
constraint grade_pk primary key (roll,courseID),
```

```
constraint fk_student foreign key (roll) references student (roll),
```

```
constraint fk_course foreign key (courseID) references course(courseID)
```

```
);
```

- Foreign Key constraint can be added or dropped using alter table command
- **Data types in Oracle:**
  - Character (char)
  - Numeric (number)
  - Date
  - LOB
  - RAW and Long RAW
  - ROWID and UROWID

- **Character Data types**
  - 1. char
  - 2. varchar2 & varchar
  - 3. LOB character data types
  - 4. LONG data types

# 1. Char Data types

- It stores fixed-length character strings between 1 and 2000 bytes.
- Default length is one byte.
- If a string is shorter, then it is blank-padded to the fixed length.
- If a string is too large, Oracle Database returns an error.
- Ex. Name char( 20 byte) //default
- Name char( 20 char)

# 2. Varchar2 and varchar Data types

- varchar2 stores variable-length character strings between 1 and 4000 bytes.
- If a string is shorter, then it is not padded.
- If a string is too large, Oracle Database returns an error.
- Varchar is synonymous to varchar2.
- Varchar2 is advised to use over varchar.



### **3. LOB Character Data types**

- **LOB** = Large Object
- LOB datatypes for character data are CLOB and NCLOB.
- They can store up to 8 TB of data.

### **4. Long Data types (LONG)**

- It can store up to 2 GB of variable-length string.
- It is not recommended by oracle.

- **Numeric Data types**
  - **Number data type:**
  - Number datatypes store positive and negative **fixed** and **floating-point** numbers, **zero**, **infinity**, and values that are the undefined result of an operation (that is, is "not a number" or **NAN**).
  - **Syntax:**
  - Column-name **number**(precision, scale)
  - **Precision** is the total number of digits
  - **Scale** is the total number of digits to the right of the decimal point
  - If a precision is not specified, the column stores values as given.
  - If no scale is specified, the scale is zero.
  - **Example:**
  - Height number(8,4);

- **Date Data types**
  - 1. data
  - 2. timestamp
  - 3. timestamp with time zone
  - 4. timestamp with local time zone

- **1. Date Data type**

- It is used to represent a date with the default format '**yyyy-mm-dd**'.
- It also represents time.
- **Example:**
- create table student  
( roll **number**,  
name **char**(20 char),  
DOB **date**,  
**constraint** student\_pk **primary key** (roll)  
);
- **insert into** student **values**(111,'AA',**date** '1999-02-17');
- **insert into** student **values**(222,'bbb',**date** '2002-12-25');

- **Format date using to\_date( ) function:**
- **Syntax:**
- **To\_date(<string>,<format>)**
- Example:
- **insert into student values(111,'AA',to\_date('25-01-2017','dd-mm-yyyy'));**
  
- **Format date display using to\_char( ) function:**
- Example:
- **Select to\_char(DOB,'month-dy-yyyy') from student;**
- **Output = January-fri-2017**
- **Dates can be added and subtracted**
- **They can be compared**

- **2. timestamp Data type**
  - format **'yyyy-mm-dd:hh24:mi:ss.ff'**.
  - **Timestamp[fractional\_second\_precision]**
  - **fraction\_of\_second** can range from 0 to 9
  - Default value is 6
  - **Example:**
  - create table student  
( roll **number**,  
name **char**(20 char),  
DOB **timestamp**,  
**constraint** student\_pk **primary key** (roll)  
);
- **insert into student values(111,'AA', timestamp '2010-01-10 13:15:04.22');**

- **3. timestamp Data type with time zone**
  - **Example:**
  - create table student  
( roll **number**,  
name **char**(20 char),  
DOB **timestamp with time zone**,  
**constraint** student\_pk **primary key** (roll)  
);

- **4. timestamp Data type with local time zone**
  - **Example:**
  - create table student  
( roll **number**,  
name **char**(20 char),  
DOB **timestamp with time zone**,  
**constraint** student\_pk **primary key** (roll)  
);



- **LOB Data types**
  - LOB = large object
  - **1. BLOB**
  - **2. CLOB and NCLOB**
  - **3. BFILE**

# 1. BLOB

- The BLOB datatype stores unstructured **binary data** in the database.
- BLOBs can store up to 128 terabytes of binary data.
- Binary data like audio data, image data, etc.

# 2. CLOB and NCLOB

- The CLOB and NCLOB datatypes store up to 128 terabytes of **character data** in the database.
- CLOBs store database character set data
- NCLOBs store Unicode national character set data.

### 3. BFILE

- The BFILE datatype stores **unstructured binary data** in operating-system files **outside** the database of a DBMS.
- A BFILE column stores a file locator that points to an external file outside the database of a DBMS.
- The amount of BFILE data that can be stored is limited by the operating system.
- BFILEs are read only and it cannot be modified.
- The operating system must maintain the file integrity, security, and durability for BFILEs.

- **RAW and LRAW Data types**
- **Oracle Net Services and Import/Export** automatically convert CHAR, VARCHAR2, and LONG data between the database character set and the user session character set, if the two character sets are different.
- RAW and LRAW data types store binary data that should not be interpreted when moved between systems.
- RAW has maximum size of 2000 bytes
- LRAW has maximum size of 2 GB

- **ROWID and UROWID Data types**
- Internally, every database table has a **ROWID pseudocolumn**.
- It stores binary values called **rowids**.
- Each **rowid** represents the storage address of a row.
- A **physical rowid** identifies a row in an ordinary table.
- A **logical rowid** identifies a row in an index-organized table.
- The ROWID datatype can store only physical rowids.
- The UROWID (universal rowid) datatype can store physical, logical, or foreign (non-Oracle) rowids.
- **Example:**
- Select **rowid**, name from student where student.roll=11;

- **Substring pattern matching:**
- **Select \* from student s where s.name like ‘%ra%’;**
- It will retrieve all the student with names like Ram, Raman, Ramesh, Raju, Ranjit.
-

- **Order by clause:**
- An ORDER BY clause allows you to specify the order in which rows appear in the result set.
- **ORDER BY** < column-Name | ColumnPosition | Expression >  
[ **ASC** | **DESC** ] [ **NULLS FIRST** | **NULLS LAST** ]
- **Example:**
- **Select \* from student order by CGPA;**
- **Select roll, name from student order by name ASC;**
- **Select roll, name from student order by name ASC NULLS FIRST;**

- **Select name, roll from student order by name DESC, roll ASC;**

<u>Name</u>	<u>Roll</u>
Zukar	111
Zukar	112
Young	113
Mark	114

- **Select name, roll from student order by name DESC, roll DESC;**

<u>Name</u>	<u>Roll</u>
Zukar	112
Zukar	111
Young	113
Mark	114



- **Group by clause:**
- A GROUP BY clause groups a result into subsets that have matching values for one or more columns.
- In each group, no two rows have the same value for the grouping columns.
- NULLs are considered equivalent for grouping purposes.
- **Group by** <col-name(s),function(s)> **from** <table\_name>  
    **where** <conditions> **group by** <col-name(s)>

<u>Roll</u>	<u>Name</u>	<u>Gender</u>	<u>State</u>
111	John	M	MP
112	Moses	M	OD
113	john	M	TN
114	Mary	F	WB
115	Alia	F	MP

**Example:**

**Select name from student group by name;**

**Name**

John

Moses

Mary

Alia

<u>Roll</u>	<u>Name</u>	<u>Gender</u>	<u>State</u>
111	John	M	MP
112	Moses	M	OD
113	john	M	OD
114	Mary	F	OD
115	Alia	F	WB

**Example:**

**Select gender from student group by gender;**

**Gender**

M

F

<u>Roll</u>	<u>Name</u>	<u>Gender</u>	<u>State</u>
111	John	M	MP
112	Moses	M	OD
113	john	M	OD
114	Mary	F	OD
115	Alia	F	WB

**Example:**

**Select state from student group by state;**

**State**

MP

OD

WB

<u>Roll</u>	<u>Name</u>	<u>Gender</u>	<u>State</u>
111	John	M	MP
112	Moses	M	OD
113	john	M	OD
114	Mary	F	OD
115	Alia	F	WB

### Example:

**Select** state, count(\*) as No\_Students **from** student **group by** state;

<u>State</u>	<u>No_Students</u>
MP	1
OD	3
WB	1

<u>Roll</u>	<u>Name</u>	<u>CGPA</u>	<u>State</u>
111	John	9.9	MP
112	Moses	7.5	OD
113	john	8.65	OD
114	Mary	6.78	OD
115	Alia	8.26	WB

### Example:

**Select** state, max(CGPA),min(CGPA),avg(CGPA) **from** student **group by** state;

<u>State</u>	<u>Max(CGPA)</u>	<u>Min(CGPA)</u>	<u>Avg(CGPA)</u>
MP	9.9	9.9	9.9
OD	8.65	6.78	7.643
WB	8.26	8.26	8.26

<u>Roll</u>	<u>Name</u>	<u>Gender</u>	<u>State</u>
111	John	M	MP
112	Moses	M	OD
113	john	M	OD
114	Mary	F	OD
115	Alia	F	WB

**Example:**

**Select** state, count(\*) as No\_Students **from** student  
**group by** state **order by** state **ASC**;

<u>State</u>	<u>No_Students</u>
OD	3
MP	1
WB	1

<u>Roll</u>	<u>Name</u>	<u>Gender</u>	<u>State</u>
111	John	M	MP
112	Moses	M	OD
113	john	M	OD
114	Mary	F	OD
115	Alia	F	WB

### Example:

**Select** gender, count(\*) as No\_Students **from** student s

**where** s.state = 'OD'

**group by** gender **order by** gender **ASC**;

<u>Gender</u>	<u>No_Students</u>
---------------	--------------------

F	1
---	---

M	2
---	---



## Having Clause:

- A HAVING clause restricts the results of a GROUP BY.
- The HAVING clause is applied to each group of the table.
- If there is no GROUP BY clause, the HAVING clause is applied to the entire result as a single group.
- **HAVING** <group-conditions>

<u>Roll</u>	<u>Name</u>	<u>CGPA</u>	<u>State</u>
111	John	9.9	MP
112	Moses	7.5	OD
113	john	8.65	OD
114	Mary	6.78	OD
115	Alia	8.26	WB

**Example:**

**Select state, count(\*) from student group by state having count(\*)=1;**

**State    Count(\*)**

MP            1

WB            1

<u>Roll</u>	<u>Name</u>	<u>CGPA</u>	<u>State</u>
111	John	9.9	MP
112	Moses	7.5	OD
113	john	8.65	OD
114	Mary	6.78	OD
115	Alia	8.26	WB

**Example:**

**Select** state, count(\*) **from** student **group by** state **having** min(CGPA)>8;

**State    Count(\*)**

MP            1

WB            1

<u>Roll</u>	<u>Name</u>	<u>Gender</u>	<u>State</u>
111	John	M	MP
112	Moses	M	OD
113	john	M	OD
114	Mary	F	OD
115	Alia	F	WB

**Example:**

**Select** state, count(\*) **from** student **where** state = 'OD'  
**group by** gender **having** count(\*) = 1;

<u>Gender</u>	<u>Count(*)</u>
F	1

<u>Roll</u>	<u>Name</u>	<u>Gender</u>	<u>State</u>
111	John	M	MP
112	Moses	M	OD
113	john	M	OD
114	Mary	F	OD
115	Alia	F	WB

**Example:**

**Select state from student;**

**State**

MP

OD

OD

OD

WB

- Duplicates are present

<u>Roll</u>	<u>Name</u>	<u>Gender</u>	<u>State</u>
111	John	M	MP
112	Moses	M	OD
113	john	M	OD
114	Mary	F	OD
115	Alia	F	WB

**Example:**

**Select Distinct(state) from student;**

**State**

MP

OD

WB

- No Duplicates are present

## **Deletion Effect on child table:**

### **1. On delete cascade:**

- if a record in the parent table is deleted, then the corresponding records in the child table will automatically be deleted.

# 1. On delete cascade:

- Example:

<u>rollno</u>	name
2017001	john
2017024	mary

Student

<u>Course-ID</u>	C-name
CS3002	DBMS
CS3015	OS

Course

<u>rollno</u>	<u>Course-ID</u>	Grade
2017001	CS3002	A
2017001	CS3015	B
2017024	CS3002	B
2017024	CS3015	A

Grade



# 1. On delete cascade:

- **Syntax:**

- CREATE TABLE table\_name

- ( col-1 datatype,

- col-2 datatype,

- ...

- CONSTRAINT <fk\_name> FOREIGN KEY (col-1, ...)

- REFERENCES <parent\_table> (col-1, ...) ON DELETE CASCADE );

# 1. On delete cascade:

- Example:

```
create table grade
(roll    number(5),
 courseID varchar2(20),
 CGPA    number(6,4),
constraint grade_pk primary key (roll,courseID),
constraint fk_student foreign key (roll) references student(roll)
on delete cascade,
constraint fk_course  foreign key (courseID) references course(courseID)
on delete cascade
);
```

# **Deletion Effect on child table:**

## **2. On delete set null:**

- If a record in the parent table is deleted, then the corresponding records in the child table will have the foreign key fields set to null.
- The records in the child table will not be deleted.

## **3. On delete no action:**

- It prevent deleting a row in the parent table is deleted, when that row is referenced in the child table.
- This the default action of delete command.

## **Update Effect on child table:**

- Update in parent table(candidate key) is allowed if the statement does not leave any rows in the child table without a referenced parent key value.
- Update of primary key of parent table are not allowed.

## **Set Operations:**

1. Union
2. Intersection
3. Set difference
4. IN, NOT IN
5. ANY
6. ALL

# 1. Union operation:

<u>rollno</u>	name
2017001	john
2017024	mary

Student

<u>rollno</u>	<u>Course-ID</u>	Grade
2017001	CS3002	A
2017001	CS3015	B
2017024	CS3002	B
2017024	CS3015	A

Grade

- **Syntax:**
- $\langle \text{Set-1} \rangle \text{ union } \langle \text{Set-2} \rangle$ ;
- The resultant set contains the elements of set-1 and set-2 without duplicates.

## Example:

**Select rollno from student union select rollno from grade;**

**Rollno**

2017001

2017024

# 1. Union operation:

- **Syntax:**

- **<Set-1> union all <Set-2>;**

- The resultant set contains the elements of set-1 and set-2 with duplicates.

## **Example:**

**Select rollno from student union select rollno from grade;**

### **Rollno**

2017001

2017024

2017001

2017001

2017024

2017024

## 2. Intersect operation:

<u>rollno</u>	name
2017001	john
2017024	mary

Student

<u>rollno</u>	<u>Course-ID</u>	Grade
2017001	CS3002	A
2017001	CS3015	B
2017024	CS3002	B
2017024	CS3015	A

Grade

- **Syntax:**
- **<Set-1> intersect <Set-2>;**
- The resultant set contains only those elements that are present in both set-1 and set-2.

### Example:

**Select rollno from student intersect select rollno from grade;**

**Rollno**

2017001

2017024

### 3. Set difference operation:

<u>rollno</u>	name
2017001	john
2017024	mary

<u>rollno</u>	<u>Course-ID</u>	Grade
2017001	CS3002	A
2017001	CS3015	B

- **Syntax:**  

StudentGrade
- $\langle \text{Set-1} \rangle \text{ minus } \langle \text{Set-2} \rangle$ ;
- The resultant set contains only those elements which are present in set-1, but are not present in set-2.

#### Example:

**Select** rollno **from** student **minus** **select** rollno **from** grade;

**Rollno**

2017024



## 4. Set Member operation (IN):

<u>rollno</u>	name
2017001	john
2017024	mary

<u>rollno</u>	<u>Course-ID</u>	Grade
2017001	CS3002	A
2017001	CS3015	B

- **Syntax:**  

StudentGrade
- **Select** col-1,col-2 **from** <table\_name> **where** col-1 **IN** (Set\_of\_Values);

**Example:**

**Select** rollno,name **from** student **where** rollno **IN** (select rollno **from** grade);

<u>Rollno</u>	<u>Name</u>
2017001	john

## 4. Set Member operation (NOT IN):

<u>rollno</u>	name
2017001	john
2017024	mary

<u>rollno</u>	<u>Course-ID</u>	Grade
2017001	CS3002	A
2017001	CS3015	B

- **Syntax:**  

StudentGrade
- **Select** col-1,col-2 **from** <table\_name> **where** col-1 **IN** (Set\_of\_Values);

### Example:

**Select** rollno,name **from** student **where** rollno **NOT IN** (select rollno **from** grade);

<u>Rollno</u>	<u>Name</u>
2017024	mary

## 5. ALL :

<u>rollno</u>	name
2017001	john
2017024	mary

<u>rollno</u>	<u>Course-ID</u>	Grade
2017001	CS3002	A
2017001	CS3015	B

- **Syntax:**  

StudentGrade
- **Select** col-1,col-2 **from** <table\_name> **where** col-1 **op** **ALL**(Set\_of\_Values);
- Where **op** = {>,<,>=,!=}

### Example:

**Select** rollno,name **from** student **where** rollno > **ALL** (1997011,2017020);

<u>Rollno</u>	<u>Name</u>
2017024	mary

5. ALL :

<u>rollno</u>	name
2017001	john
2017024	mary

Student

<u>rollno</u>	<u>Course-ID</u>	Grade
2017024	CS3002	A
2017024	CS3015	B

Grade

Example:

Select rollno,name from student where rollno = ALL (select rollno from grade);

<u>Rollno</u>	<u>Name</u>
2017024	mary

## 6. ANY :

<u>rollno</u>	name
2017001	john
2017024	mary

Student

<u>rollno</u>	<u>Course-ID</u>	Grade
2017001	CS3002	A
2017001	CS3015	B

Grade

- **Syntax:**
- **Select** col-1,col-2 **from** <table\_name> **where** col-1 **op** **ANY**(Set\_of\_Values);
- Where **op** = {>,<,>=,=,!=}

### Example:

**Select** rollno,name **from** student **where** rollno > **ANY** (1997011,2017020);

<u>Rollno</u>	<u>Name</u>
2017001	john
2017024	mary

## 7. Exists, Not Exists :

<u>rollno</u>	name
2017001	john
2017024	mary

Student

<u>MscID</u>	Rollno	Instru
M001	2017001	Drum
M002	2017022	Piano
M003	2017001	Bass

Music

**Example:**

**Select \* from student s where exists (select \* from music m  
where s.rollno=m.rollno);**

<u>Rollno</u>	<u>Name</u>
2017001	john

**Select \* from student s where not exists (select \* from music m  
where s.rollno=m.rollno);**

<u>Rollno</u>	<u>Name</u>
2017024	mary

# **Joins :**

- Two or more tables can be joined based on some conditions.
- **Types of Join:**
- **1. Inner Join**
  - 1.1. Equi Join
  - 1.2. Self Join
  - 1.3. Natural join
  - 1.4. Theta join
- **2. Outer Join**
  - 2.1. Left outer join
  - 2.2. Right outer join
  - 2.3. Full outer join
- **3. Cartesian Product**

## Equi-Join:

<u>rollno</u>	name
2017001	john
2017024	mary

Student

<u>rollno</u>	<u>Course-ID</u>	Grade
2017001	CS3002	A
2017024	CS3002	Ex
2017001	CS1000	B
2017024	CS1000	C

Grade

- It joins two tables using the **equal(=) comparison** operator in the join condition.

### Example:

Select \* from student s, grade g where **s.rollno = g.rollno**;

<u>Rollno</u>	<u>Name</u>	<u>Rollno</u>	<u>Course-ID</u>	<u>Grade</u>
2017001	john	2017001	CS3002	A
2017001	john	2017001	CS1000	B
2017024	mary	2017024	CS3002	Ex
2017024	mary	2017024	CS1000	C



# Equi-Join:

## Example:

Select s.rollno, Course-ID, grade from student s, grade g where **s.rollno = g.rollno**;

<u>Rollno</u>	<u>Course-ID</u>	<u>Grade</u>
2017001	CS3002	A
2017001	CS1000	B
2017024	CS3002	Ex
2017024	CS1000	C

## Equi-Join:

<u>rollno</u>	name
2017001	john
2017024	mary

Student

<u>rollno</u>	<u>Course-ID</u>	Grade
2017001	CS3002	A
2017024	CS3002	Ex
2017001	CS1000	B
2017024	CS1000	C

Grade

- Equi-join can also be done using key words **join** with **on(join conditions)**.

### Example:

Select \* from student s **join** grade g **on** (s.rollno = g.rollno);

<u>Rollno</u>	<u>Name</u>	<u>Rollno</u>	<u>Course-ID</u>	<u>Grade</u>
2017001	john	2017001	CS3002	A
2017001	john	2017001	CS1000	B
2017024	mary	2017024	CS3002	Ex
2017024	mary	2017024	CS1000	C

# Self-Join:

- Self-join causes a table to be joined to itself.

<u>EmpID</u>	Emp-name	MngID
2017001	john	NULL
2017024	mary	2017001
2017022	leo	2017026
2017026	Gita	NULL

<u>EmpID</u>	Emp-name	MngID
2017001	john	NULL
2017024	mary	2017001
2017022	leo	2017026
2017026	Gita	NULL

• Example: Employee as m

Employee as e

Select \* from employee m **join** employee e on (m.**EmpID** = e.**MngID**);

<u>EmpID</u>	Emp-name	MngID	<u>EmpID</u>	Emp-name	MngID
2017001	john	NULL	2017024	mary	2017001
2017026	Gita	NULL	2017022	leo	2017026

# Self-Join:

- Compute the list of managers and their employees

<u>EmpID</u>	Emp-name	MngID
2017001	john	NULL
2017024	mary	2017001
2017022	leo	2017026
2017026	Gita	NULL

Employee as m

<u>EmpID</u>	Emp-name	MngID
2017001	john	NULL
2017024	mary	2017001
2017022	leo	2017026
2017026	Gita	NULL

Employee as e

- **Example:**

Select m.name as Manager, e.name as Employee from employee m

join employee e on (m.EmpID = e.MngID);

Manager      Employee

john

mary

Gita

leo

## Natural Join:

- In equi-join the common columns of the two tables were duplicated.
- Natural join removes these duplicate columns.
- **Natural join implicitly involves equal operator on common columns of the tables.**
- So, equal operator and common columns should not be specified.
- It is done using the key word “**natural join**”.

**Natural Join:**

<u>rollno</u>	name
2017001	john
2017024	mary

**Student**

<u>rollno</u>	<u>Course-ID</u>	Grade
2017001	CS3002	A
2017024	CS3002	Ex
2017001	CS1000	B
2017024	CS1000	C

**Grade**

**Example:**

Select \* from student **natural join** grade;

<u>Rollno</u>	<u>Name</u>	<u>Course-ID</u>	<u>Grade</u>
2017001	john	CS3002	A
2017001	john	CS1000	B
2017024	mary	CS3002	Ex
2017024	mary	CS1000	C

# Theta Join:

- It is join operation in which the join operator is **not equal operator**.

<u>rollno</u>	name
2017001	john
2017024	mary

Student

<u>rollno</u>	<u>Course-ID</u>	Grade
2017001	CS3002	A
2017024	CS3002	Ex
2017001	CS1000	B
2017024	CS1000	C

Grade

- Example:
- **Select \* from** student s **join** grade g **on** (s.rollno < g.rollno);

<u>Rollno</u>	<u>Name</u>	<u>Rollno</u>	<u>Course-ID</u>	<u>Grade</u>
2017001	john	201724	CS3002	A
2017001	john	201724	CS1000	C

## Left Outer Join:

- In **inner joins** only the rows **that satisfied the join conditions** are joined.
- In **outer joins** not only the rows **that satisfy the join conditions** are joined, but also, the rows that **do not satisfy the join conditions** are joined.
- A new table is produced by first joining the rows of the left table of a join that satisfy the join conditions with the rows of the right table and then by joining the rows of the left table that do not satisfy the join conditions with the rows of the right table.
- But, in such rows of the new table, the columns of the right table are assigned with NULL values.



# Left Outer Join:

<u>rollno</u>	name
2017001	john
2017024	mary
2017022	Gita
2017030	Mark

Student

<u>rollno</u>	<u>Course-ID</u>	Grade
2017001	CS3002	A
2017024	CS3002	Ex

Grade

- Example:
- **Select \* from student s left outer join grade g on (s.rollno =g.rollno);**

<u>Rollno</u>	<u>Name</u>	<u>Rollno</u>	<u>Course-ID</u>	<u>Grade</u>
2017001	john	2017001	CS3002	A
2017024	mary	2017024	CS3002	Ex
2017022	Gita	NULL	NULL	NULL
2017030	Mark	NULL	NULL	NULL

## Right Outer Join:

<u>rollno</u>	<u>Course-ID</u>	Grade
2017001	CS3002	A
2017024	CS3002	Ex

Grade

<u>rollno</u>	name
2017001	john
2017024	mary
2017022	Gita
2017030	Mark

Student

- Example:
- **Select \* from grade g right outer join student s on (g.rollno = s.rollno);**

<u>Rollno</u>	<u>Course-ID</u>	<u>Grade</u>	<u>Rollno</u>	<u>Name</u>
2017001	CS3002	A	2017001	john
2017024	CS3002	Ex	2017024	mary
NULL	NULL	NULL	2017022	Gita
NULL	NULL	NULL	2017030	Mark

## Full Outer Join:

<u>SptrID</u>	Rollno	Game
S001	2017001	Soccer
S002	2017024	Volley

Sports

<u>MscID</u>	Rollno	Instru
M001	2017001	Drum
M002	2017022	Piano

Music

- Example:
- **Select \* from sports s full outer join music m on (s.rollno = m.rollno);**

<u>SptrID</u>	<u>Rollno</u>	<u>Game</u>	<u>MscID</u>	<u>Rollno</u>	<u>Instru</u>
S001	2017001	Soccer	M001	2017001	Drum
S002	2017024	Volley	NULL	NULL	NULL
NULL	NULL	NULL	M002	2017022	Piano

# Cartesian Product:

- Every row of the left table is joined with all the rows of the right table.
- In this join there is no join condition
- Key word is “**cross join**”

Sports			Music		
<u>SptrID</u>	Rollno	Game	<u>MscID</u>	Rollno	Instru
S001	2017001	Soccer	M001	2017001	Drum
S002	2017024	Volley	M002	2017022	Piano

<u>SptrID</u>	Rollno	Game	<u>MscID</u>	Rollno	Instru
S001	2017001	Soccer	M001	2017001	Drum
S001	2017001	Soccer	M002	2017022	Piano
S002	2017024	Volley	M001	2017001	Drum
S002	2017024	Volley	M002	2017022	Piano

# Cartesian Product:

- Example:
- **Select \* from** sports, music;
- **Select \* from** sports **cross join** music;

## Other Constraints:

- 1. **Not null**
- 2. **Unique**
- 3. **Check(expression)**

### Example:

```
create table election
(  eid    number(5),
   ssn    number(5) unique,
   name varchar2(20 char) default 'xxxx' not null,
   age number(4,2) default 0 not null,
   constraint election_pk primary key (eid),
   constraint check_age check(age >= 18)
);
```

## Other Constraints:

- 4. Trigger

## View:

- It is a subset of one or more base tables
- It is a way to provide relevant data to appropriate users.
- It provides a level of security.
- **Types:** 1. Virtual views    2. Materialized views.
- **Virtual views** are never stored in the database, but their definitions are stored.
- **Materialized views** are stored in the database along with their definitions.

**View:**

<u>SptrID</u>	Rollno	Game
S001	2017001	Soccer
S002	2017024	Volley

**Sports**

<u>SptrID</u>	Game
S001	Soccer
S002	Volley

<u>SptrID</u>	Rollno
S001	2017001
S002	2017024

Rollno	Game
2017001	Soccer
2017024	Volley

**Some views of the Sports Table**



**View:**

<u>MscID</u>	Rollno	Instru
M001	2017001	Drum
M002	2017022	Piano

**Music**

<u>SptrID</u>	Rollno	Game
S001	2017001	Soccer
S002	2017024	Volley

**Sports**

<u>SptrID</u>	Game	<u>MscID</u>	Instru
S001	Soccer	M001	Drum

**View**

# View:

- Syntax:
- **create view** <view\_name> **as select** col-1,col-2 **from** <tab-1,tab-2>
- **where** <conditions>;
- Example:
- **Create view** music\_view **as select** mscid, instru **from** music;
- **Create view** music\_view **as select** mscid, instru **from** music  
**where** roll= 2017022;

# View:

- Example:

- **Create view** music\_sport\_view **as**

```
select mscid, sprtsid,game,instru
```

```
from music m, sports s
```

```
where (m.roll = s.roll);
```

- Views can be queried like a normal table.
- Views that are created from a single table, can be **updated, inserted and deleted**.
- Views having aggregates, group by, union all, subquery in the select clause etc can not be updated.

# Materialized View:

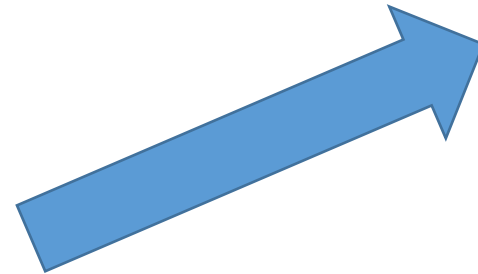
- Syntax:
  - **create materialized view** <view\_name>  
    **as select** col-1,col-2 **from** <tab-1,tab-2> **where** <conditions>;
- Example:
- **Create materialized view** music\_view **as select** mscid, instru **from** music;
- **Create materialized view** music\_view **as select** mscid, instru **from** music  
    **where** roll= 2017022;

# Relational Model of a Table in a Database:

- A **table** is represented as a **relation** in relational model.
- A relation,  $r$ , over a collection of sets  $= \{D_1, D_2, \dots, D_n\}$ , is a **subset** of their **Cartesian Product**  $= D_1 \times D_2 \times \dots \times D_n$ .
- The set  $D_i$  is a set of values representing a domain.
- Domain  $D_i$  is specified by the data types provided by a DBMS.
- So, a relation,  $r$ , is a set of  $n$ -tuples  $\{d_1, d_2, \dots, d_n\}$  where  $d_i \in D_i$ .

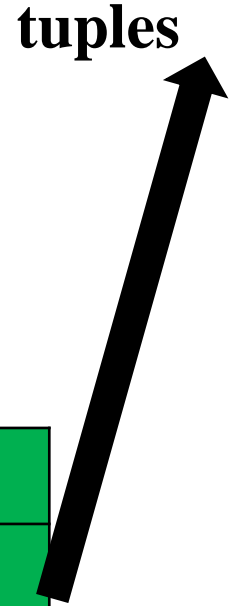
# Relational Model of a Table in a Database:

- Example:
- Let there be two sets
  - **roll**={001,002}
  - **name**={john, mary}
- Their Cartesian product is
  - **roll** x **name** = {(001, john),
  - (001, mary),
  - (002, john),
  - (002, mary)}
- 



roll	name
001	john
002	mary

Student relation



tuples

# Relational Model of a Table in a Database:

- **Relation Schema:**

- It is the logical structure of a relation.
- $R(A_1:D_1, A_2:D_2, A_3:D_3, \dots, A_n:D_n)$
- **R** is the name of the Schema
- $A_i$  is a column/attribute
- $D_n$  is the domain of the column/attribute

- **Example:**

- STUDENT(Roll:Number(6), Name:Varchar2(10 char))
- A collection of relation schema is called **relational database schema**.

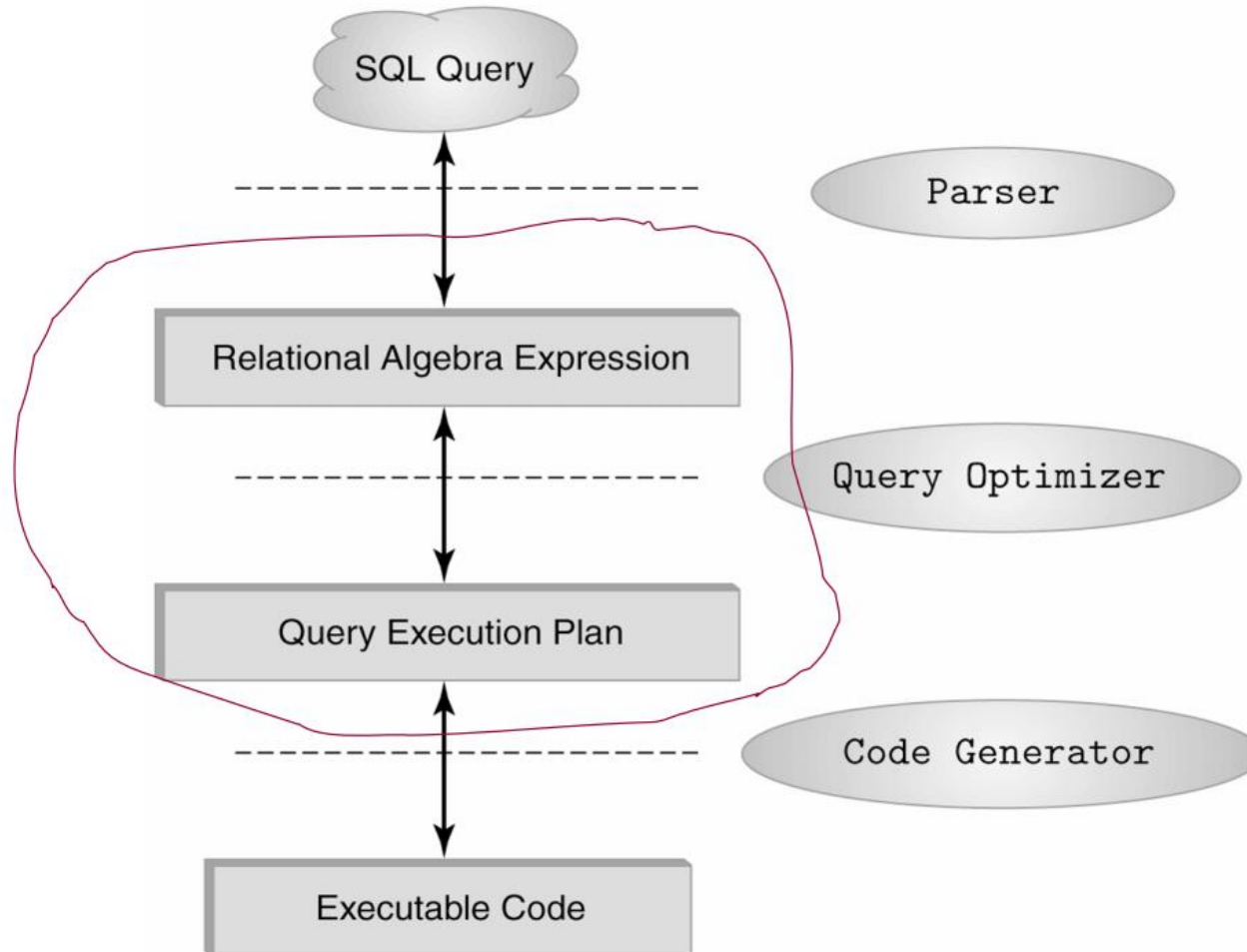
# Relational Query Languages:

- 1. Relational Algebra
- 2. Relational Calculus



# Relational Algebra:

## The Role of Relational Algebra in a DBMS



- **Relational Algebra** is a procedural query language that enables users to manipulate and retrieve data from a relational database.
- **Relational Algebra Operators:**
  - 1. select operator( $\sigma$ )
  - 2. project operator( $\pi$ )
  - 3. Cartesian/Cross product ( $\times$ )
  - 4. rename operator( $\rho$ )
  - 5. set difference (-)
  - 6. union ( $\cup$ )
  - 7. intersection( $\cap$ )
  - 8. join ( $\bowtie$ )
  - 9. division( $\div$ )

- **select operator( $\sigma$ )**

- It is used to produce a **new relation** by selecting a set of tuples from a relation that satisfies some selection conditions specified on its attributes.

- It is represented by the Greek symbol  $\sigma$  (sigma)

- **General representation:**

- $\sigma_{\text{<selection\_conditions>}}(\text{Relation})$

- Example:

- $\sigma_{\text{roll}=101}(\text{STUDENT})$

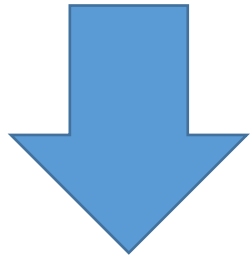
- $\sigma_{\text{Dept}=\text{'CS'} \text{ AND } \text{state}=\text{'OD'}}(\text{STUDENT})$

- $\sigma_{\text{state} \neq \text{'OD'}}(\text{STUDENT})$

- **In SQL, operator( $\sigma$ ) is specified in the WHERE clause of a query.**

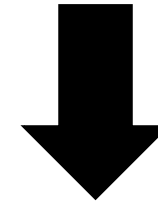
- Example:

- **Select \* from student where state='OD';**



- **$\sigma_{\text{state}=\text{'OD'}}(\text{student})$**

Roll	Name	Gender	State
111	John	M	MP
112	Moses	M	OD
113	john	M	OD
114	Mary	F	OD
115	Alia	F	WB



**Select operation**

Roll	Name	Gender	State
112	Moses	M	OD
113	john	M	OD
114	Mary	F	OD

- select operator( $\sigma$ ) is commutative
- $\sigma_{\text{condition-2}}(\sigma_{\text{condition-1}}(R)) = \sigma_{\text{condition-1}}(\sigma_{\text{condition-2}}(R))$

- Example:

- 1. Select \* from student s where s.state = 'OD' AND s.gender = 'M';
- 2. Select \* from student s where s.gender = 'M' AND s.state = 'OD';
- In the above query 1 & 2 are equivalent.



- $\sigma_{\text{state='OD'}}(\sigma_{\text{gender='M'}}(\text{student})) = \sigma_{\text{gender='M'}}(\sigma_{\text{state='OD'}}(\text{student}))$

- **project operator( $\pi$ )**

- It is used to produce a **new relation** by selecting some attributes, specified by a user, of a relation.
- It is represented by the Greek symbol  $\pi$  (pi)
- **General representation:**

- $\pi_{\text{<attributes\_list>}}(\text{Relation})$

- Example:

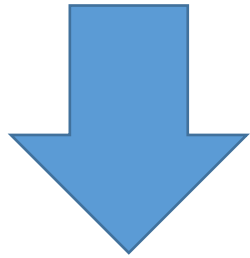
- $\pi_{\text{roll,name}}(\text{STUDENT})$

- $\pi_{\text{roll}}(\text{STUDENT})$

- **In SQL, operator( $\pi$ ) is specified in the SELECT clause of a query.**

- Example:

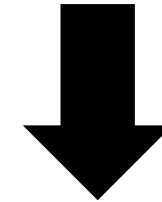
- **Select** roll, name **from** student;



$\pi_{\text{name}}(\text{student})$

- 

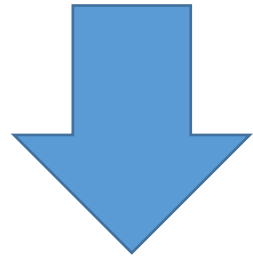
Roll	Name	Gender	State
111	John	M	MP
112	Moses	M	OD
113	John	M	OD
114	Mary	F	OD
115	Alia	F	WB



**Project operation**

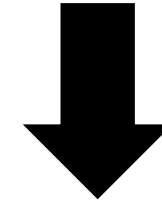
Roll	Name
111	John
112	Moses
113	John
114	Mary
115	Alia

- Example:
- **Select distinct(name) from student;**



- $\pi_{\text{name}}(\text{student})$
- Projection removes duplicate tuples

Roll	Name	Gender	State
111	John	M	MP
112	Moses	M	OD
113	John	M	OD
114	Mary	F	OD
115	Alia	F	WB



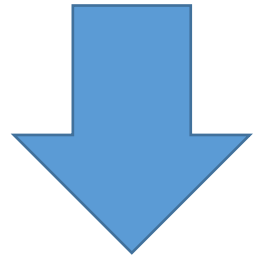
**Project operation**

Name
John
Moses
Mary
Alia



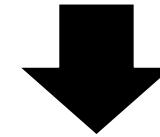
- Example:

- **Select** roll, name **from** student **where** state='OD';



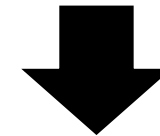
- $\pi_{\text{roll, name}}(\sigma_{\text{state='OD'}}(\text{student}))$

Roll	Name	Gender	State
111	John	M	MP
112	Moses	M	OD
113	john	M	OD
114	Mary	F	OD
115	Alia	F	WB



**1. Select operation**

Roll	Name	Gender	State
112	Moses	M	OD
113	john	M	OD
114	Mary	F	OD

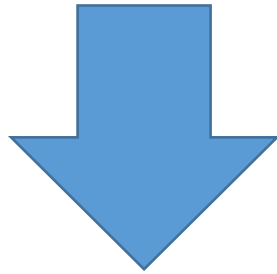


**2. Project operation**

Roll	Name
112	Moses
113	john
114	Mary

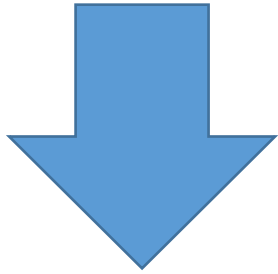
- **Rename operator( $\rho$ ):**
- It is used to give new names to attributes and relations.
- It is represented by the Greek symbol  $\rho$  (rho)
- **General representation:**
  - $\rho_{s(b1,b2...bn)}(\mathbf{Relation})$
  - $\rho_s(\mathbf{Relation})$
  - $\rho_{(b1,b2...bn)}(\mathbf{Relation})$ 
    - where  $s$  is the new name of a relation
    - $b1, b2...$  are the new name of the attributes
- Example:
- $\rho_{\text{student\_details}(\text{roll\_number}, \text{full\_name}, \text{department})}(\mathbf{student})$

- In SQL, operator( $\rho$ ) is accomplished using the key word 'AS' 'rename to'.
- Example:
- **alter table** student **rename to** student\_details;



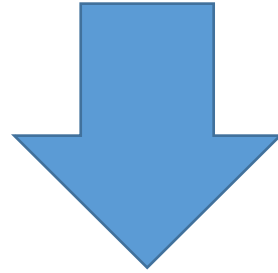
$\rho_{\text{student\_details}}(\text{student})$

- Example:
- **Select** roll as **roll\_number**, name as **student\_name**, state as **state\_code**  
**from** student s **where** s.state = 'OD';



$\rho_{(\text{roll\_number}, \text{student\_name}, \text{state\_code})}(\sigma_{\text{state}='OD'}(\text{student}))$

- Example:
- **insert into** **odiya\_students** (**roll\_number, student\_name, state\_code**)  
**select** roll, name, state **from** student s **where** s.state = 'OD';



$\rho$  **odiya\_students**(**roll\_num, student\_name, state\_code**)( $\sigma_{\text{state}='OD'}(\text{student})$ )



**odiya\_students**(**roll\_number, student\_name, state\_code**)  $\leftarrow (\sigma_{\text{state}='OD'}(\text{student}))$

- The symbol( $\leftarrow$ ) also means assignment

# Cartesian/Cross Product ( $\times$ ):

- A new relation is produced by joining each tuple of the left relation with all the tuples of the right relation.
- It is represented by symbol  $\times$ .
- $M \times N$
- It is commutative
- $M \times N = N \times M$
- It is associative
- $(M \times N) \times R = N \times (M \times R)$
- **Example:**
- sports  $\times$  music
- Expensive Operation

**Sports**

<u>SptrID</u>	Game
S001	Soccer
S002	Volley

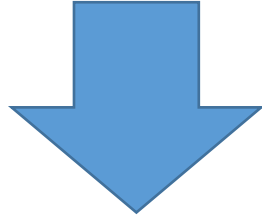
**Music**

<u>MscID</u>	Instru
M001	Drum
M002	Piano

**sports  $\times$  music**

<u>SptrID</u>	Game	<u>MscID</u>	Instru
S001	Soccer	M001	Drum
S001	Soccer	M002	Piano
S002	Volley	M001	Drum
S002	Volley	M002	Piano

- Example:
- **Select \* from sports, music;**
- **Select \* from sports cross join music;**



- **sports × music**

# Union ( $\cup$ ):

- Two relations  $M(A_1, A_2 \dots A_n)$  and  $N(B_1, B_2 \dots B_n)$  are union compatible,
- if they have same degree and if  $\text{domain}(A_i) = \text{domain}(B_i)$ .
- Union, intersection & set difference are defined on union compatible relations.
- The **new relation** produced by  $(M \cup N)$  includes all the tuples of relation M & relation N.
- But duplicate tuples are eliminated.
- **Representation:**
- $M \cup N$
- Union is **commutative**
- $M \cup N = N \cup M$
- Union is **associative**
- $(M \cup N) \cup R = M \cup (N \cup R)$

**CSE**

<u>Roll</u>	Game
111	Soccer
222	Volley

**ECE**

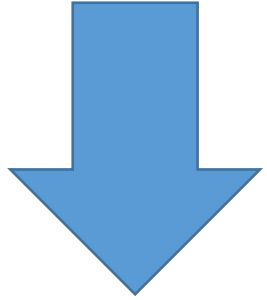
<u>Roll</u>	Game
323	Tennis
214	TT

**CSE  $\cup$  ECE**

<u>Roll</u>	Game
111	Soccer
222	Volley
323	Tennis
214	TT



- Example:
- **Select \* from CSE union Select \* from ECE;**



- **CSE  $\cup$  ECE**

# Intersection ( $\cap$ ):

- The **new relation** produced by  $(M \cap N)$  includes only those tuples which are in both the relations.
- **Representation:**
- $M \cap N$
- It is **commutative**
- $M \cap N = N \cap M$
- It is **associative**
- $(M \cap N) \cap R = M \cap (N \cap R)$

**CSE**

<u>CourseID</u>	Name
111	DSA
222	DBMS

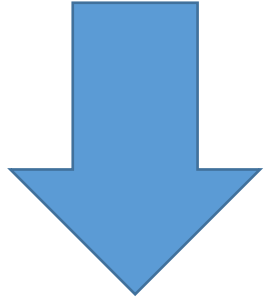
**ECE**

<u>CourseID</u>	Name
111	DSA
214	ED

**$CSE \cap ECE$**

<u>CourseID</u>	Name
111	DSA

- Example:
- **Select \* from CSE intersect Select \* from ECE;**



- **$\text{CSE} \cap \text{ECE}$**

# Set Difference (—):

- The **new relation** produced by  $(M - N)$  includes those tuples which are only in relation M and not in N.
- **Representation:**
- $M - N$
- It is **not commutative**
- $M - N \neq N - M$
- It is **not associative**
- $(M - N) - R \neq M - (N - R)$

**CSE**

<u>CourseID</u>	Name
111	DSA
222	DBMS

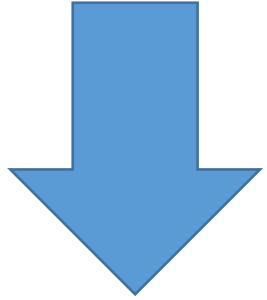
**ECE**

<u>CourseID</u>	Name
111	DSA
214	ED

**CSE — ECE**

<u>CourseID</u>	Name
222	DBMS

- Example:
- **Select \* from CSE minus Select \* from ECE;**



- **CSE – ECE**

# **Join( $\bowtie$ ):**

- **1. Inner Join**

- 1.1. Equi-Join
- 1.2. Natural Join
- 1.3. Theta Join

- **2. Outer Join**

- 2.1. Left outer Join
- 2.2. Right outer Join
- 2.3. Full outer Join

- **3. Semi Join**

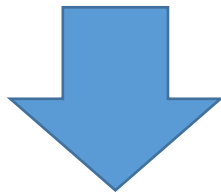
- 3.1. Left semi join
- 3.2. Right semi join

- **4. Anti semi join or Anti join**

- 4.1. Left Anti Join
- 4.2. Right Anti Join
-

- **Equi-Join:**
- A new relation is produced by joining the tuples of two relations using equality comparison operator **explicitly** on their common attributes in the join conditions.
- The new relation contains all the attributes of the two relations.
- It is represented as  $M \bowtie_{\langle equality\_join\_conditions \rangle} N$
- Example:

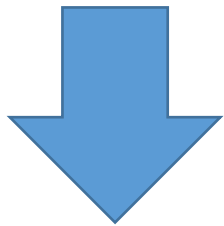
**Select \* from student s join grade g on (s.rollno = g.rollno);**



**student**  $\bowtie_{\langle student.roll=grade.roll \rangle}$  **grade**

- **Natural Join:**
- A new relation is produced by joining the tuples of two relations using equality comparison operator **implicitly** on their common attributes in the join conditions.
- The new relation contains a single copy of the common attributes along with other attributes of the two relations.
- It is represented as  $M *_{\langle equality_{join\_conditions} \rangle} N$
- It is also represented as  $M * N$
- Example:

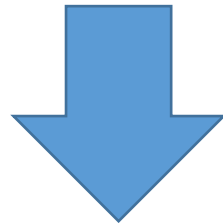
**Select \* from student natural join grade;**



student \* grade



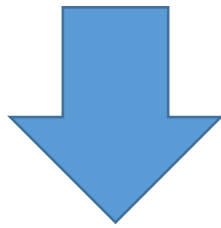
- **Theta Join:**
- A new relation is produced by joining the tuples of two relations using other comparison operators, except equality operator, **explicitly** on the attributes of the two relations in the join conditions.
- It is represented as  $M \bowtie_{\langle join\_conditions \rangle} N$
- It is also represented as  $M \bowtie_{\theta} N$ , where  $\theta$  represents any comparison operator, except the equality operator.
- Example:
- **Select \* from student s join grade g on (s.rollno < g.rollno);**



student  $\bowtie_{s.rollno < g.rollno}$  grade

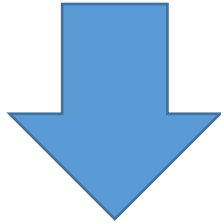
- Inner Joins are commutative and associative.

- **Left Outer Join:**
- A new relation is produced by first joining the tuples of the left relation of a join that satisfy the join conditions with the tuples of the right relation and then by joining the tuples of the left relation that do not satisfy the join conditions with the tuples of the right relation.
- But, in such tuples of the new relation, the columns of the right relation are assigned with NULL values.
- It is represented as  $M \bowtie_{\langle \text{join\_conditions} \rangle} N$
- Example:..
- **Select \* from student s left outer join grade g on (s.rollno =g.rollno);**



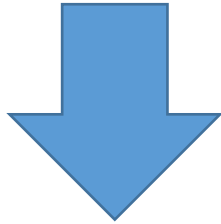
student  $\bowtie_{\langle \text{s.rollno=g.rollno} \rangle}$  grade

- **Right Outer Join:**
- It is represented as  $M \bowtie N$
- Example:.
- **Select \* from student s right outer join grade g on (s.rollno =g.rollno);**



student  $\bowtie_{\langle s.rollno=g.rollno \rangle}$  grade

- **Full Outer Join:**
- It is represented as  $M \bowtie N$
- Example:.
- **Select \* from student s full outer join grade g on (s.rollno =g.rollno);**



student  $\bowtie_{\langle \text{s.rollno}=\text{g.rollno} \rangle}$  grade

- Outer joins are commutative and associative.

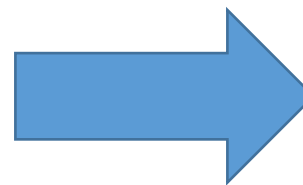
- **Semi Join:**
- The new relation produced by a semi-join between two relations contains only one copy of tuples of one relation that have one or more matches (according to join condition) with the tuples of the other relation.
- **Left Semi Join:**
- It is represented as  $M \bowtie_{\langle \text{join\_conditions} \rangle} N$ .
- The new relation produced by  $M \bowtie N$  contains only one copy of the tuples of relation M that satisfy some join conditions with the tuples of relation N.
- Example:

A	B
1	2
3	2
5	6

Relation M

B	C
2	3
2	55
54	66

Relation N

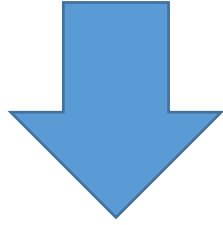


A	B
1	2
3	2

$M \bowtie_{\langle M.B=N.B \rangle} N$

- Example:.

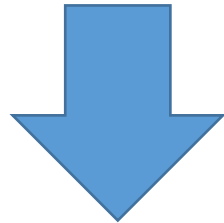
**Select \* from student s where exists (select \* from music m  
where s.rollno=m.rollno);**



Student ⋈<sub><s.rollno=m.rollno></sub> Music

- **Right Semi Join:**
- It is represented as  $M \bowtie_{\langle \text{join\_conditions} \rangle} N$ .
- The new relation produced by  $M \bowtie N$  contains only one copy of the tuples of relation N that satisfy some join conditions with the tuples of relation M.
- Example:.

**Select \* from music m where exists (select \* from student s  
 where s.rollno=m.rollno);**



**Student  $\bowtie_{\langle \text{s.rollno=m.rollno} \rangle}$  Music**

- Semi joins are not commutative and associative



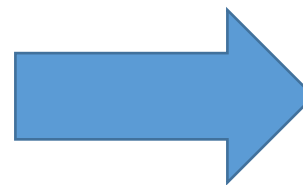
- **Anti join or Anti Semi Join:**
- The new relation produced by a anti-join between two relations contains only one copy of tuples of one relation that have one or more non-matches (according to join conditions) with the tuples of the other relation.
- **Left Anti Join:**
- It is represented as  $M \triangleright_{\langle \text{join\_conditions} \rangle} N$ .
- The new relation produced by  $M \triangleright N$  contains only one copy of the tuples of relation M that do not satisfy some join conditions with the tuples of relation N.
- Example:

A	B
1	2
3	7
5	6

Relation M

B	C
2	3
2	55
54	66

Relation N



A	B
3	7
5	6

$M \triangleright_{\langle M.B=N.B \rangle} N$

- Example:.

**Select \* from student s where not exists (select \* from music m  
where s.rollno=m.rollno);**



Student  $\triangleright$   $\langle s.rollno=m.rollno \rangle$  Music

- **Right Anti Join:**
- It is represented as  $M \Join_{\langle \text{join\_conditions} \rangle} N$ .
- The new relation produced by  $M \Join N$  contains only one copy of the tuples of relation N that do not satisfy some join conditions with the tuples of relation M.
- Example:.

**Select \* from music m where not exists (select \* from student s  
where s.rollno=m.rollno);**



**Student  $\Join_{\langle \text{s.rollno=m.rollno} \rangle}$  Music**

- Anti joins are not commutative and associative

- **Division Operator:**
- It is not an operator and is not directly supported.
- But this concept helps to answer queries which involve the term “ALL”.
- It is represented as  $M \div N$ .

Roll	Sub
101	DAA
101	DBMS
102	DAA
103	DBMS
104	DBMS
104	DAA

**Relation M**

Sub
DAA
DBMS

**Relation N**



Roll
101
104

**$M \div N$**

- **Example:**
- Q. Let M be the student registration relation.
- Let N be the mandatory subjects for this semester of CSE.
- **\* Find the students who have registered for all the mandatory subject for this semester of CSE?**

- **Solution:**
- **Step-1:**
- Identify the students who have registered in any of the subjects.
- $T1 \leftarrow \pi_{roll}(M)$

Roll	Sub
101	DAA
101	DBMS
102	DAA
103	DBMS
104	DBMS
104	DAA

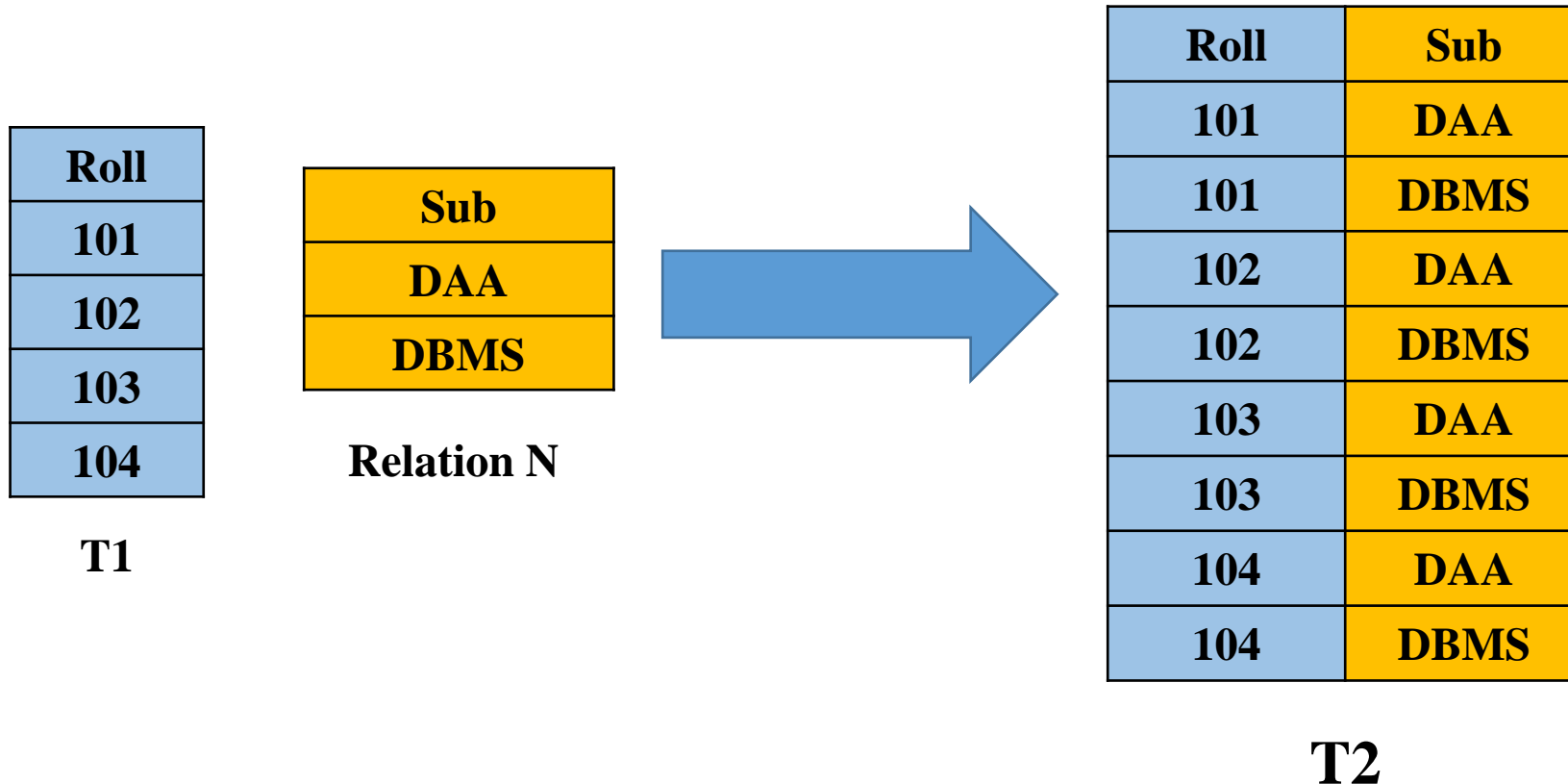
**Relation M**



Roll
101
102
103
104

**T1**

- **Step-2:**
- Assume that the registered students have registered for all the mandatory subjects, and according compute such a relation.
- $T2 \leftarrow (T1 \times N)$



- **Step-3:**
- T2 contains false data.
- So, identify those students who did not register for any of the mandatory subjects.
- $T3 \leftarrow (T2 - M)$

Roll	Sub
101	DAA
101	DBMS
102	DAA
102	DBMS
103	DAA
103	DBMS
104	DAA
104	DBMS

**T2**

Roll	Sub
101	DAA
101	DBMS
102	DAA
103	DBMS
104	DBMS
104	DAA

**Relation M**



Roll	Sub
102	DBMS
103	DAA

**T3**



- **Step-4:**
- Identify the roll number of those students who did not register for any of the mandatory subjects.
- $T4 \leftarrow \pi_{roll}(T3)$

Roll	Sub
102	DBMS
103	DAA

**T3**



Roll
102
103

**T4**

- **Step-5:**
- Identify the roll number of those students who have registered for all the mandatory subjects of CSE for this semester.
- $T5 \leftarrow (T1 - T4)$

Roll
101
102
103
104

**T1**

Roll
102
103

**T4**



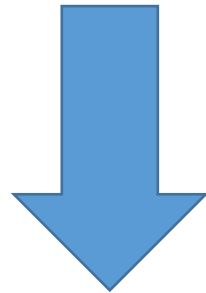
Roll
101
104

**T5**

- **Generalized Projection:**

- It extends the projection operation by allowing functions of attributes to be included in the projection list.
- It is represented as
- $\pi_{F1, F2, \dots, Fn}(R)$
- Where F1, F2 ...,Fn are functions over the attributes of a relation 'R'.
- Example:

**Select** eid, salary, 0.15\*salary as Tax\_Amount **from** employee;



$$\rho_{(eid, salary, tax\_amount)}(\pi_{(eid, salary, 0.15 * salary)}(employee))$$

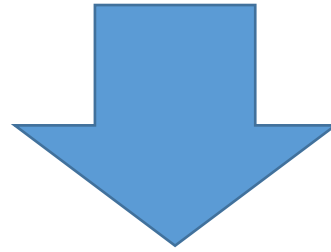
- **Aggregate functions and Grouping:**

- It is represented as:

- $\langle grouping\_attributes \rangle \mathfrak{F} \langle function\_list \rangle (Relation)$

**Example:**

**Select** state, count(\*) **as** student\_strength **from** student **group by** state;



$\rho_{state, student\_strength} (state \mathfrak{F}_{state, count(*)} (student) )$

# **Tuple Relational Calculus:**

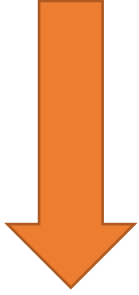
- 1. Tuple Relational Calculus
- 2. Domain Relational Calculus

# Tuple Relational Calculus:

- A relational calculus expression does not specify how data has to be retrieved, but it specifies what data is to be retrieved.
- So, it is a non-procedural language.
- The expressive power of relational algebra and calculus are identical.
- **Tuple Variable:**
  - It is a variable that takes on tuples of a particular relation schema values.
  - It ranges over a particular database relation.

- **General form** of tuple relational calculus is:
  - $\{ T \mid \text{Conditional\_Expression}( T ) \}$
- T is the Target Tuple variable.
- T ranges over tuples of values.
- Conditional\_Expression( T ) is the body of the query.
- It evaluates to TRUE or FALSE when T is substituted with some tuple values.
- It can involve other tuple variables as well.

- **Example:**
- **Select \* from student;**



- $\{ t \mid \text{student}(t) \}$
- Relation **student** is called the **range relation** of **tuple variable t**.
- **Student( t )** is True if the tuple assigned to 't' is a tuple of relation student.



- **Select** roll, name **from** student;



- $\{ t.\text{roll}, t.\text{name} \mid \text{student}(t) \}$
- **Or**
- $\{ t \mid \text{student}(s) \text{ AND } t.\text{roll}=s.\text{roll} \text{ AND } t.\text{name}=s.\text{name} \}$
- **Select** roll, name **from** student **where** state='OD';



- $\{ t.\text{roll}, t.\text{name} \mid \text{student}(t) \text{ AND } t.\text{state} = \text{'OD'} \}$
- Student( t) is True if 't' is a tuple of student relation.
- t.state = 'OD' is true if the state attribute of t is 'OD'.

- **Select** s.roll, s.name **from** student s, music m **where** s.roll=m.roll;



- { s.roll,s.name | student(s) AND music(m) AND s.roll=m.roll }

- **Select** s.roll, s.name **from** student s, grade g **where** (s.roll=g.roll and  
g.cgpa>=9.0) ;



- { s.roll,s.name | student(s) AND grade(g) AND s.roll=g.roll AND  
g.cgpa>=9.0 }

- **Query Conditions/Formula:**

- **Atomic Conditions:**

- $R(T)$  identifies the range of the tuple variable  $T$  and evaluates to true if  $T$  is a tuple in relation  $R$ .
- $(T.a \text{ **comparison\_operator** } S.b)$  or  $(T.a \text{ **comparison\_operator** } \text{constant})$ , where  $T$  and  $S$  are relation names, 'a' and 'b' are the attributes of relation  $T$  and  $S$  respectively.

- **General Conditions:**

- If  $C1$  and  $C2$  are conditions then  $(C1 \text{ AND } C2)$ ,  $(C1 \text{ OR } C2)$ , and  $\text{NOT } C1$  are also conditions.

- **Example:**

- **Select \* from student;**



- $\{t \mid (\exists \mathbf{t})(\text{student}(t))\}$

- The condition  $(\exists \mathbf{t})(\text{student}(t))$  will be true if there is at least one or some tuple in student relation.

- **Select** m.roll, m.name **from** music m **where** m.instru\_name='Drum';



- {m.roll, m.name |  $(\exists \mathbf{m})(\text{music}(\mathbf{m}) \text{ AND } \text{m.instru\_name}=\text{'Drum'})$ }

- **Select** s.roll, sp.game\_name, m.instru\_name **from** student s, sports sp, music m **where** s.roll=sp.roll and s.roll=m.roll;



- {s.roll, sp.game\_name, m.instru\_name | student(s) AND  $(\exists \mathbf{m})(\text{music}(\mathbf{m}) \text{ AND } (\exists \mathbf{sp})(\text{sports}(\mathbf{sp}) \text{ AND } \text{s.roll}=\text{sp.roll} \text{ AND } \text{s.roll}=\text{m.roll}))$ }

- or

- {s.roll, sp.game\_name, m.instru\_name | student(s) AND  $(\exists \mathbf{m})(\exists \mathbf{sp})(\text{music}(\mathbf{m}) \text{ AND } \text{sports}(\mathbf{sp}) \text{ AND } \text{s.roll}=\text{sp.roll} \text{ AND } \text{s.roll}=\text{m.roll})$ }

- **Select** sp.roll, sp.game\_name **from** sports sp, music m **where** sp.roll=m.roll and m.instru\_name =‘Drum’;



- {sp.roll, sp.game\_name | sports(sp) AND ( $\exists m$ )(music(m) AND sp.roll=m.roll AND m.instru\_name=‘Drum’)}

- **select** dept\_name **from** student s, department d **where** d.did=s.did and s.gender!=‘F’;



- {d.dept\_name | department(d) AND ( $\exists s$ )(student(s) AND s.did=d.did AND s.gender≠‘F’)}

Student

Roll	Name
101	John
102	James
103	Tom
104	Mary

Course

SubID	SubName
CS2055	DAA
CS2043	DBMS
CS1000	DS

Registration

Roll	SubID	Grade
101	CS2055	Ex
101	CS2043	Ex
101	CS1000	Ex
102	CS2055	A
102	CS2043	A
103	CS1000	B
104	CS2055	Ex
104	CS2043	A
104	CS1000	C

- Find the names of the students who have registered for all the subjects of the course.
- $\{s.name \mid \text{student}(s) \text{ AND } (\forall c)(\text{course}(c) \text{ AND } (\exists r)(\text{registration}(r) \text{ AND } r.\text{subID} = c.\text{subID} \text{ AND } s.\text{roll} = r.\text{roll} ))\}$

Student

Roll	Name
101	John
102	James
103	Tom
104	Mary

Course

SubID	SubName
CS2055	DAA
CS2043	DBMS
CS1000	DS

Registration

Roll	SubID	Grade
101	CS2055	Ex
101	CS2043	Ex
101	CS1000	Ex
102	CS2055	A
102	CS2043	A
103	CS1000	B
104	CS2055	Ex
104	CS2043	A
104	CS1000	C

- Find the names of the students who have got Ex for all the subjects of the course.
- $\{s.name \mid \text{student}(s) \text{ AND } (\forall c)(\text{course}(c) \text{ AND } (\exists r)(\text{registration}(r) \text{ AND } r.\text{subID} = c.\text{subID} \text{ AND } s.\text{roll} = r.\text{roll} \text{ AND } r.\text{grade} = 'Ex')))\}$



- Find the names of the students who have got Ex in DBMS subject.
- $\{s.name \mid \text{student}(s) \text{ AND } (\forall c)((\text{course}(c) \text{ AND } c.subName='DBMS') \rightarrow ((\exists r)(\text{registration}(r) \text{ AND } r.subID = c.subID \text{ AND } s.roll = r.roll \text{ AND } r.grade='Ex'))))\}$
- Find the names of the students who have register for only CS2055 & CS2043.
- $Temp = \{c.subID \mid (\exists c)(\text{course}(c) \text{ AND } (c.subID='CS2055' \text{ OR } c.subID='CS2043'))\}$
- $\{s.name \mid \text{student}(s) \text{ AND } (\forall c)((Temp(c) \text{ AND } (\exists r)(\text{registration}(r) \text{ AND } r.subID = c.subID \text{ AND } s.roll = r.roll)))\}$

# Domain Relational Calculus:

- **Domain Variable:**

- It is a variable that takes a value from a domain of a particular attribute of a relational schema.

- **Example:**

- **Select \* from student;**



- $\{ \langle R, N \rangle \mid \text{student}(R, N) \}$

- Here, R and N are domain variables.

**Student**

<b>Roll</b>	<b>Name</b>
101	John
102	James
103	Tom
104	Mary

- **Select** roll, name **from** student **where** roll=104 ;



- $\{ \langle R, N \rangle \mid \text{student}(R, N) \text{ AND } R=104 \}$

- **Select** name **from** student **where** roll=104;



- $\{ \langle N \rangle \mid \text{student}(R, N) \text{ AND } R=104 \}$

- **Select** s.name, m.instru\_name **from** student s, music m **where** s.roll=m.roll;



- $\{ \langle N, C \rangle \mid \text{student}(R, N) \text{ AND } \text{music}(A, B, C) \text{ AND } R=B \}$

**Music**

<b>M-ID</b>	<b>Roll</b>	<b>Instru-name</b>
M801	101	Drum
M802	104	Guitar
M803	103	Sitar

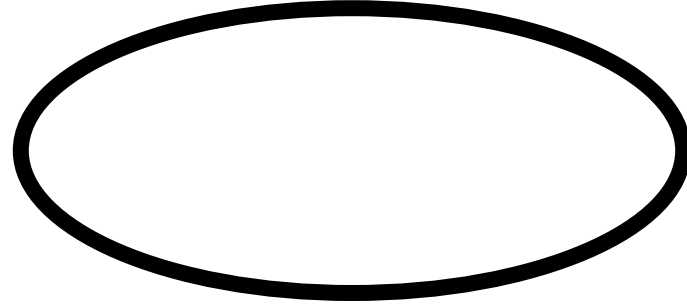
- **Database Design:**

- **Step-1:** Construct a conceptual model of the Problem using High level data model like ER diagram.
- **Step-2:** Convert the conceptual model into a logical model using relational data model.
- **Step-3:** Tune the logical model using normalization
- **Step-4:** Implement the logical model using oracle
- **Step-5:** Physical tuning for performance

- **Step-1:** Construct a conceptual model of the Problem using High level data model like ER diagram.
- **Entity Relationship (ER) Diagram:**
- It is a data model that is used to build a conceptual schema by mapping the essential real world objects and their relationships.
- **Entity:**
- It may be an object with physical/conceptual existence in the real world.
- Example: person, car, hospital, job etc.
- Symbol:



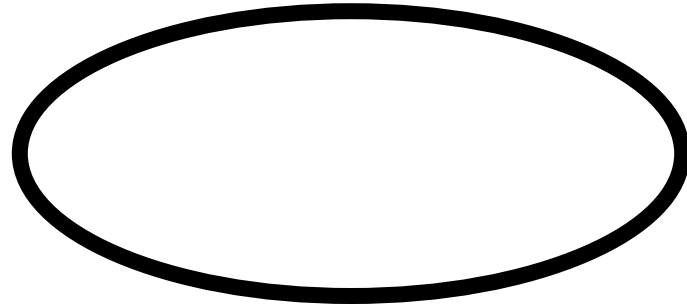
- **Attribute:**
- It is the properties that describe an entity.
- Example: person's name, ID, height, marital status, address etc.



- **Types of Attribute:**

- 1. Atomic attribute or single valued attribute
- 2. Composite attribute
- 3. Multivalued attribute
- 4. Derived attribute
- 5. Complex attribute

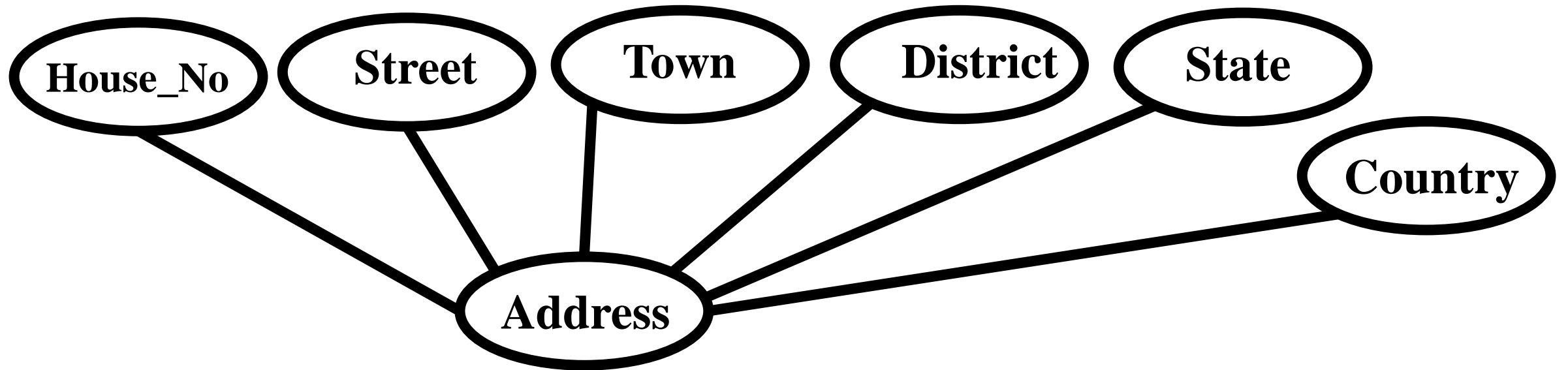
- **Atomic/single valued Attribute:**
- It cannot be divided or decomposed into any other attributes.
- It can be assigned with only one value at a time.
- Example: ID, First\_Name, Last\_Name, height, House\_No, Lane\_No, age, etc.



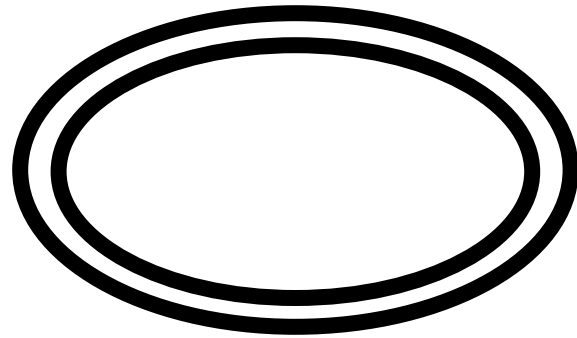


- **Composite Attribute:**

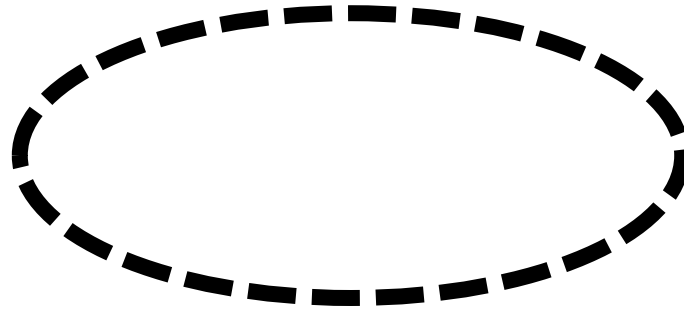
- It is composed by combining many atomic attributes.
- It can be divided or decomposed into many other atomic attributes.
- Example: Name, Address, Mobile\_No, etc



- **Multi valued Attribute:**
- It can be assigned with a set of values.
- Example: Shirt\_Color, Mobile\_No, Office\_Address, Hobby, etc.



- **Derived Attribute:**
- It is can be derived from the attributes that are stored in DBMS.
- Example: age, class\_strength, average\_CGPA, etc.

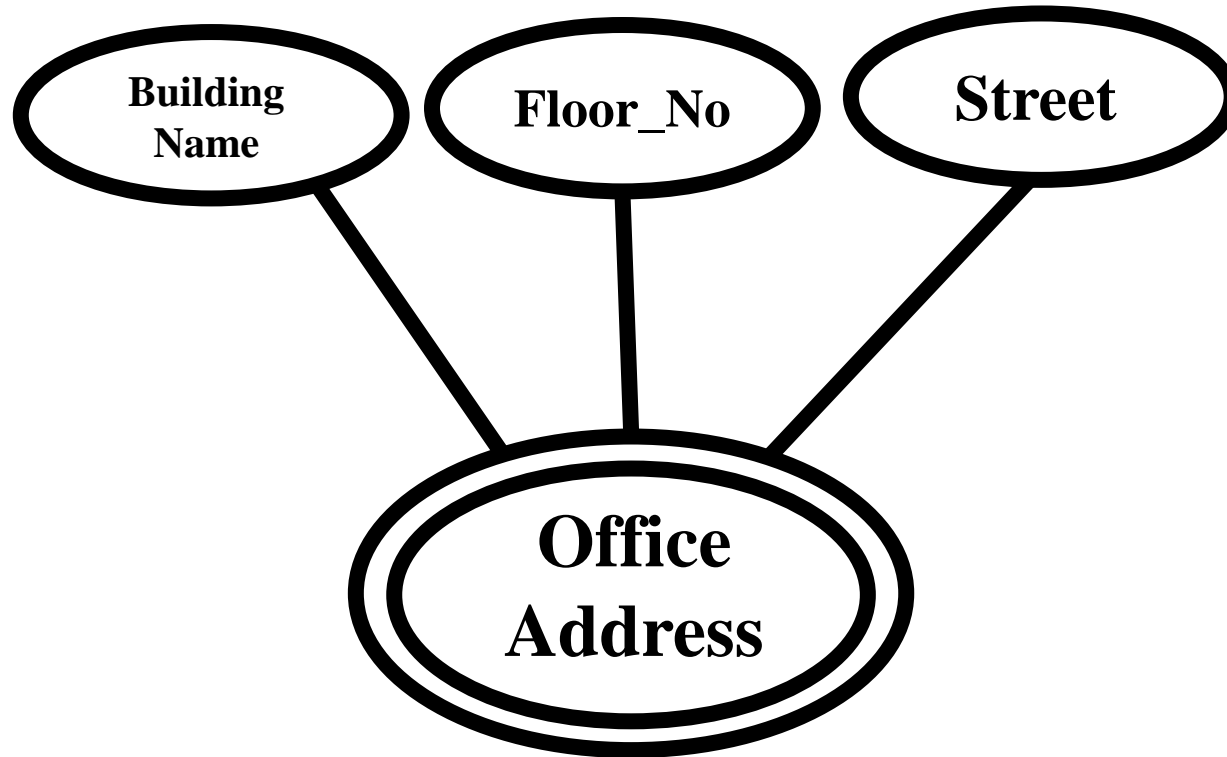


- **Complex Attribute:**

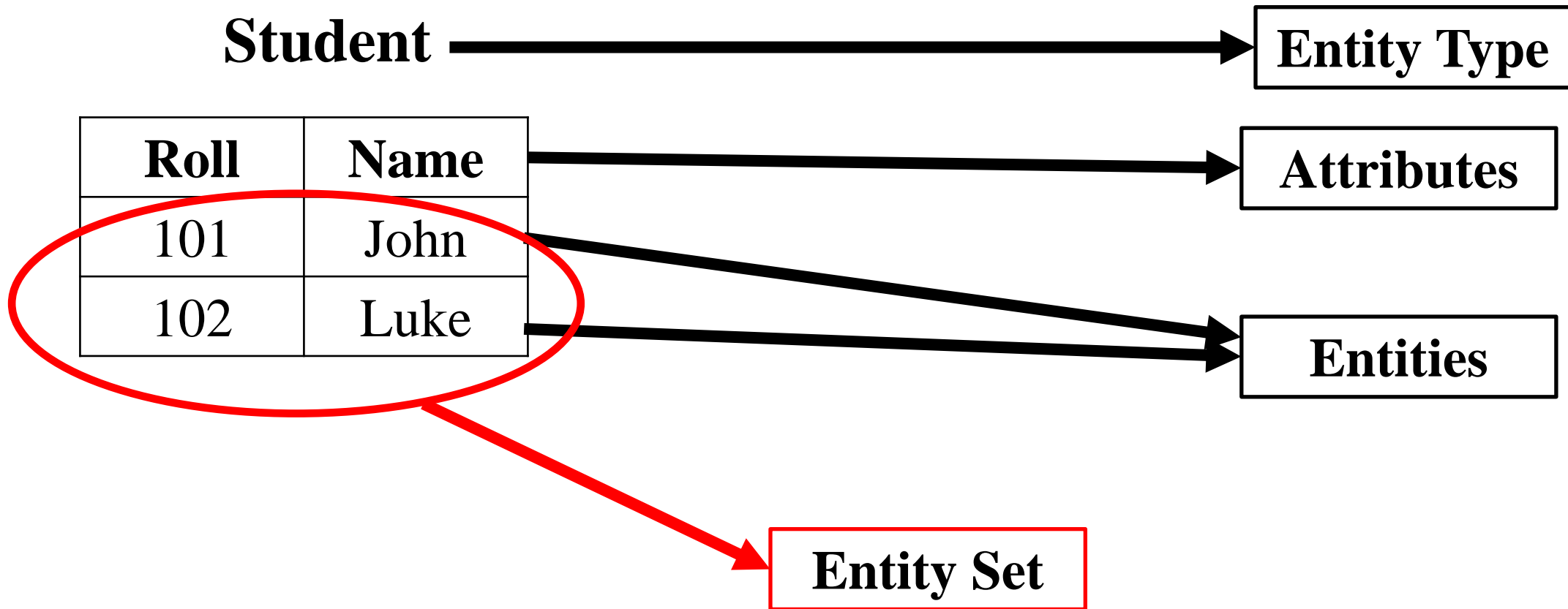
- It is a nesting of composite and multi-valued attributes.

- Example:

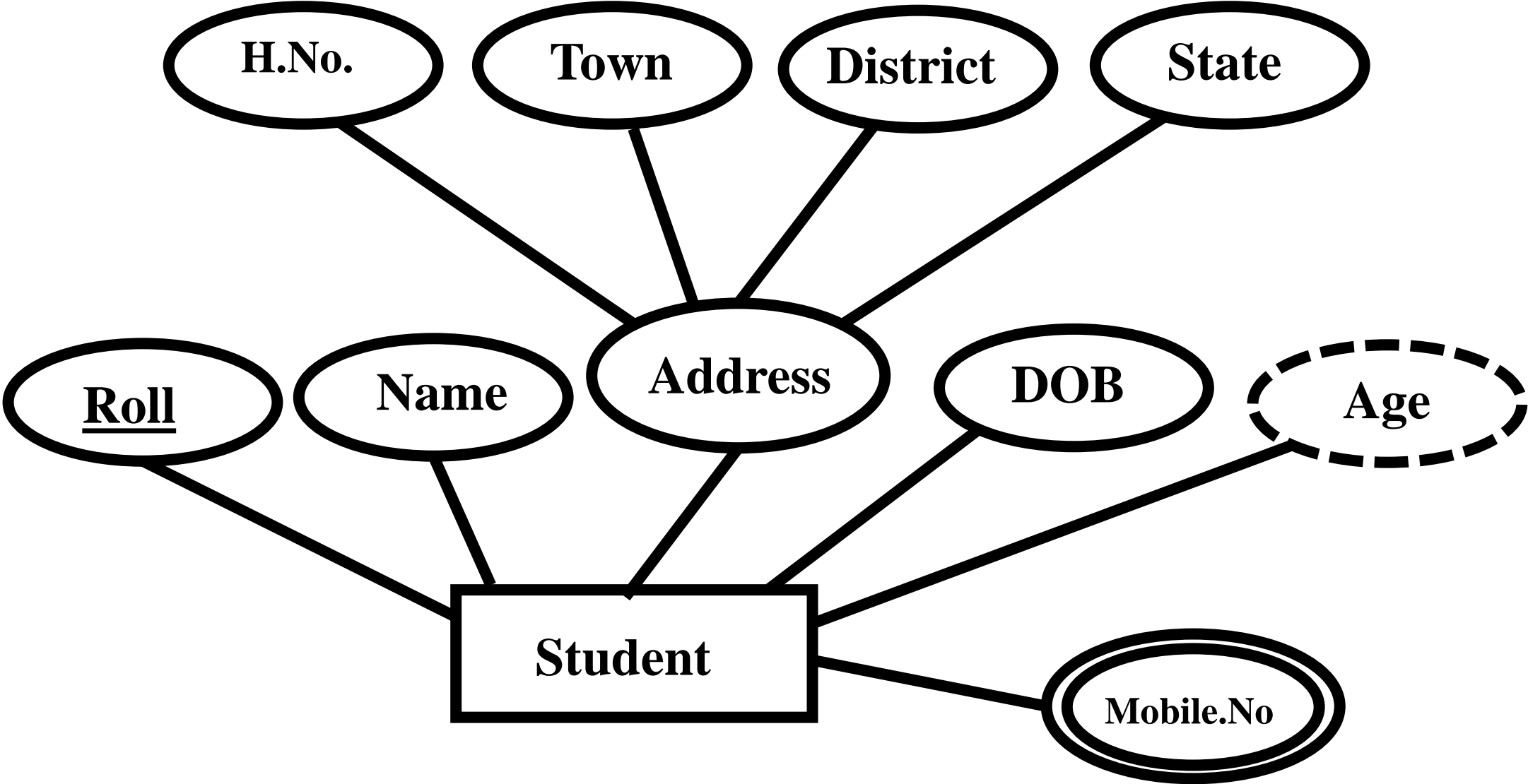
- {Office\_Address(Buiding\_Name, Floor\_No, Street, Lane, Dist, State, Country)}



- **Entity Type:**
- It is a set of entities with same attributes.
- Example: Student entity type, Car entity type

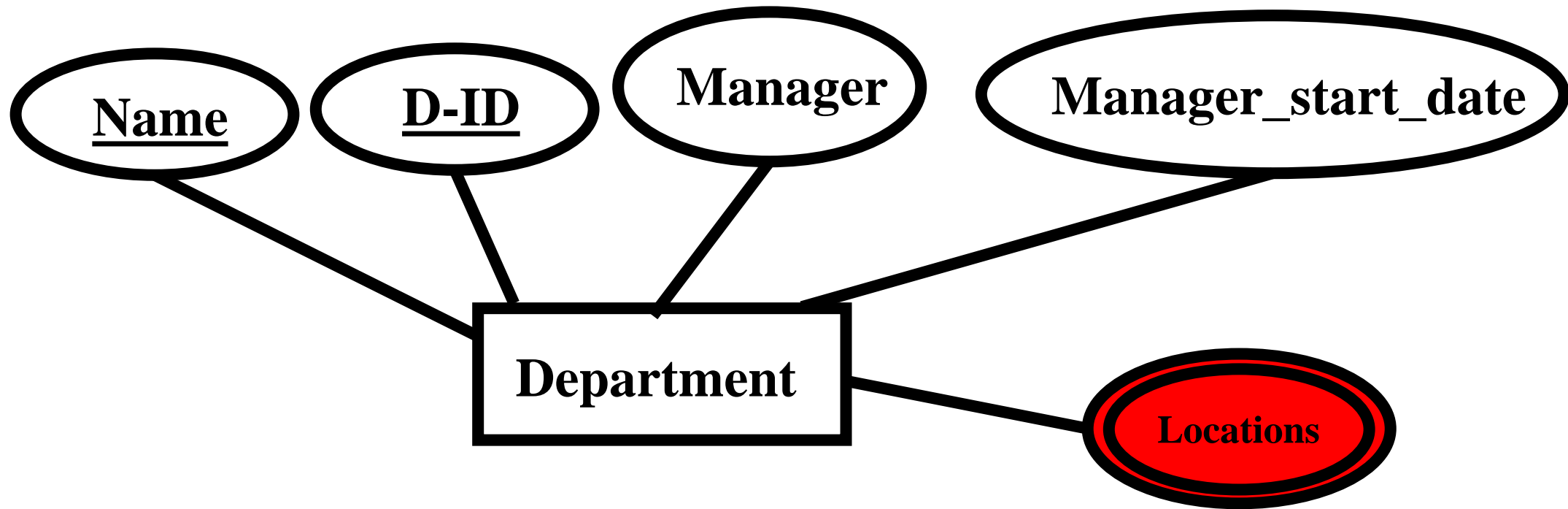


**Entity Type:**



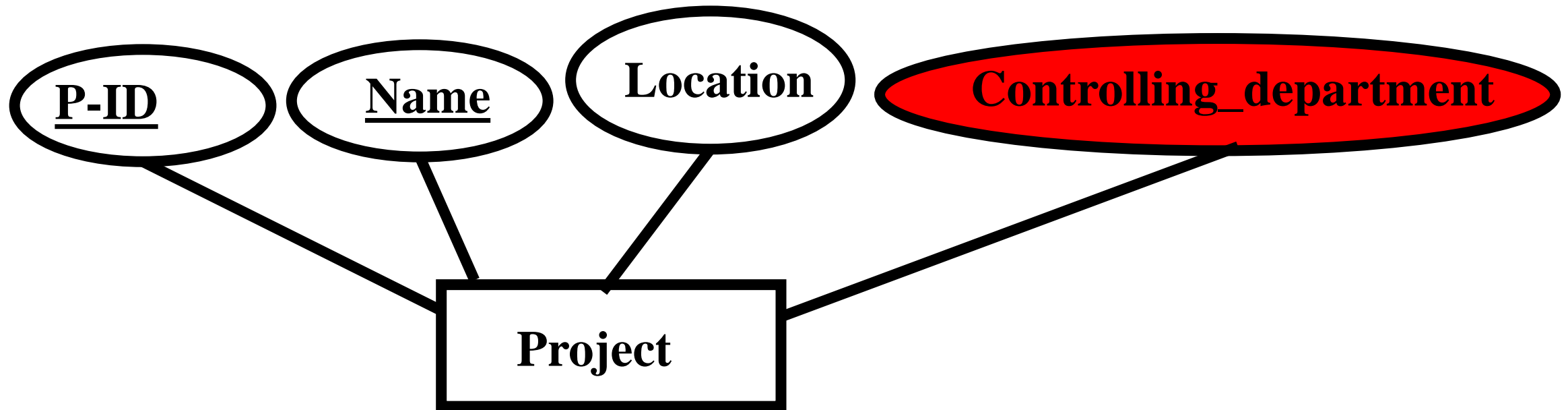
**Example:**

**Entity Types for a Dummy Company:**



**Example:**

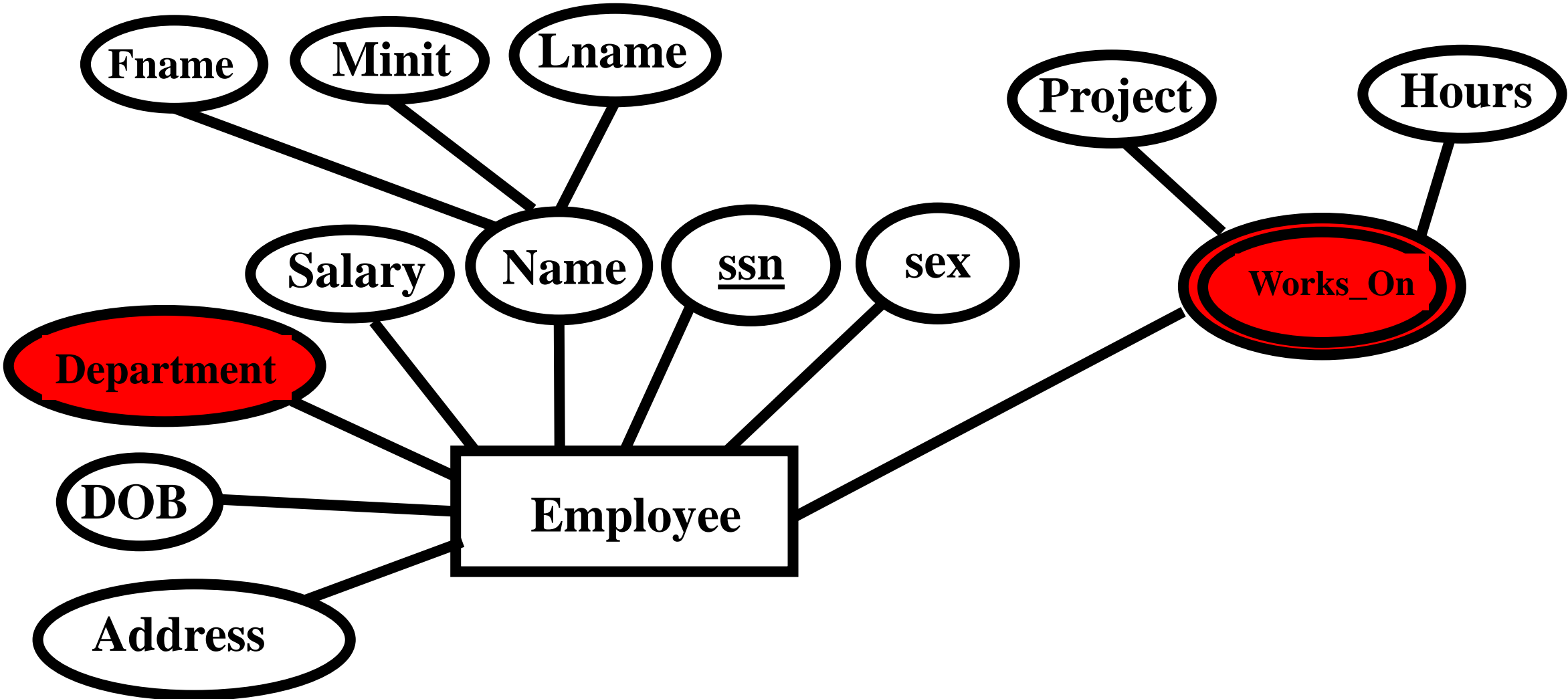
**Entity Types for a Dummy Company:**





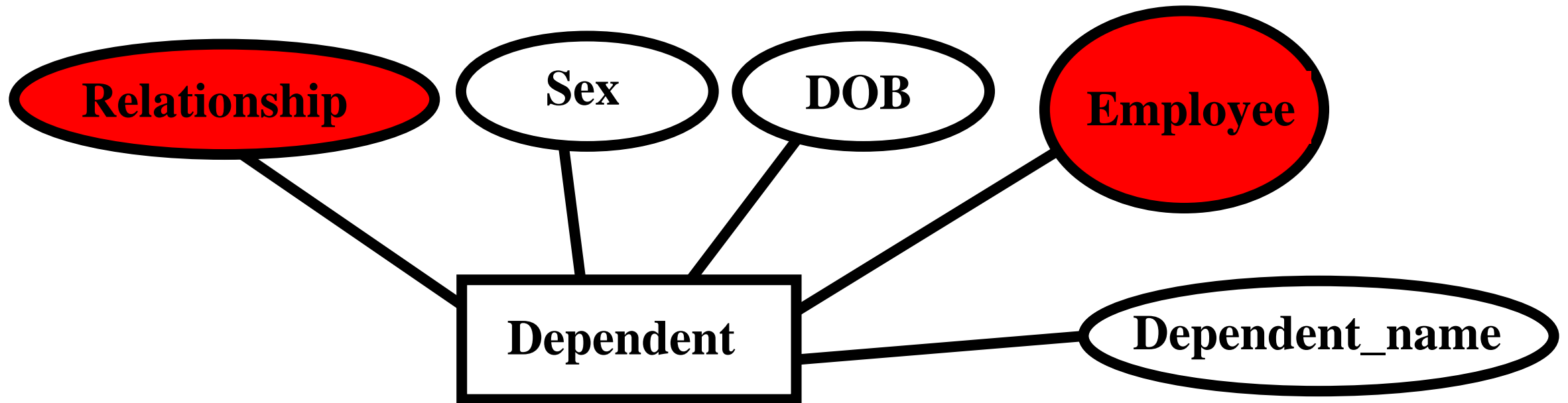
**Example:**

**Entity Types for a Dummy Company:**

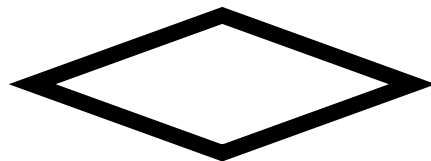


**Example:**

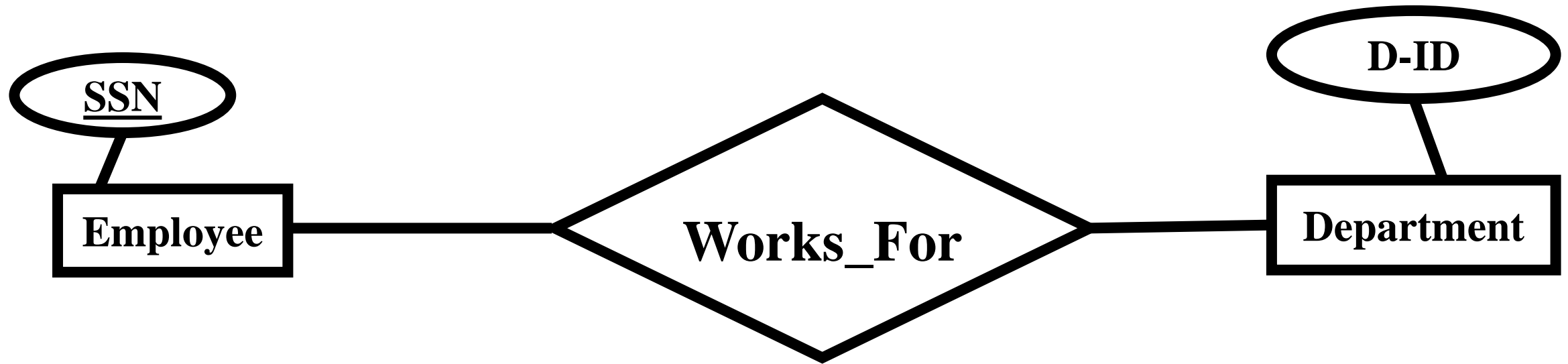
**Entity Types for a Dummy Company:**



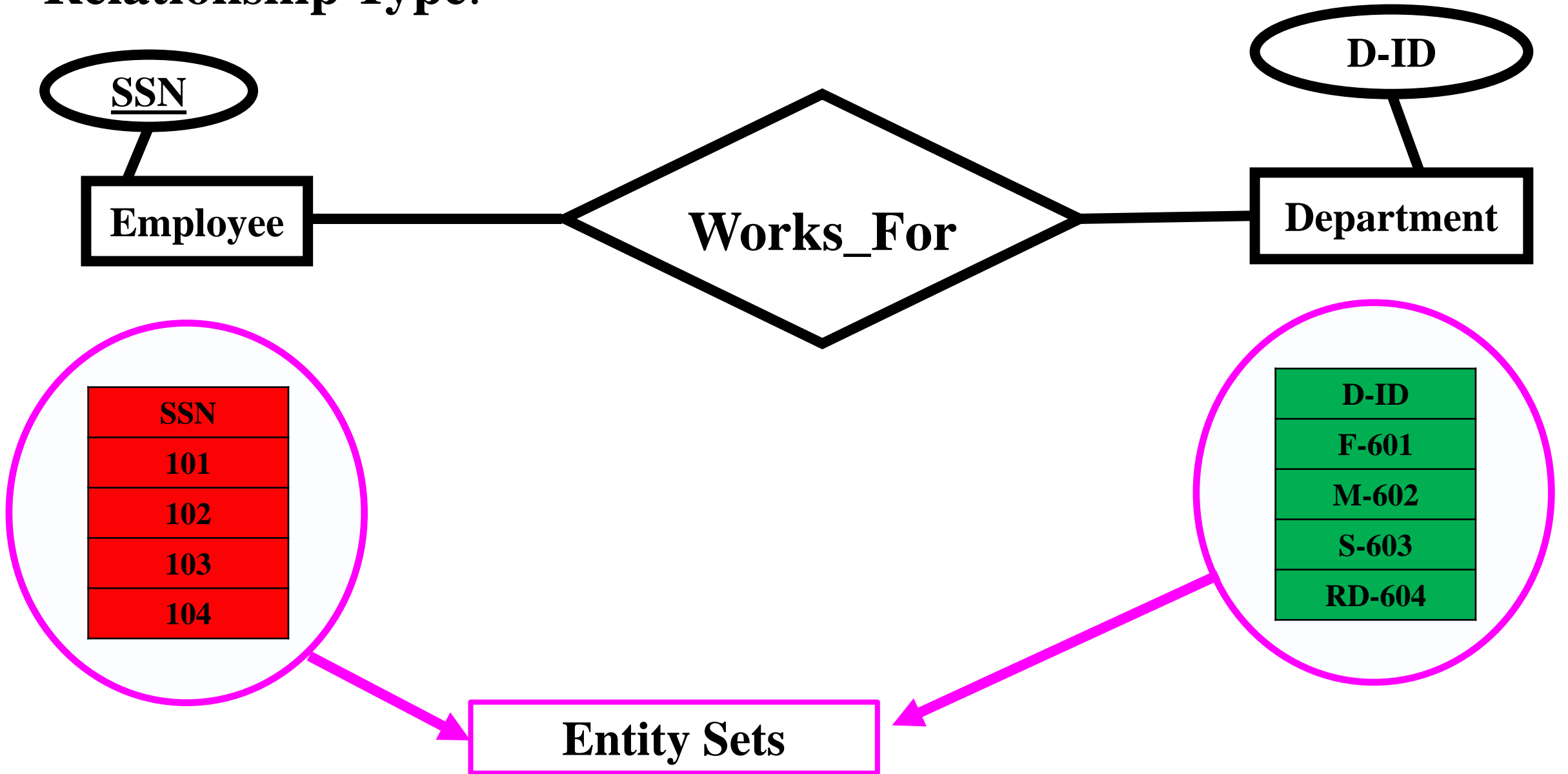
- Entity types of a particular eco-system or mini-world are related to one another.
- Relationships among the entity types are generally captured as attributes in the Initial design.
- Relationship among entity types are represented as **Relationship Type**.
- A set of associations/relationships that exists among a collection of entity types is called a **Relationship Type**.
- Symbol:



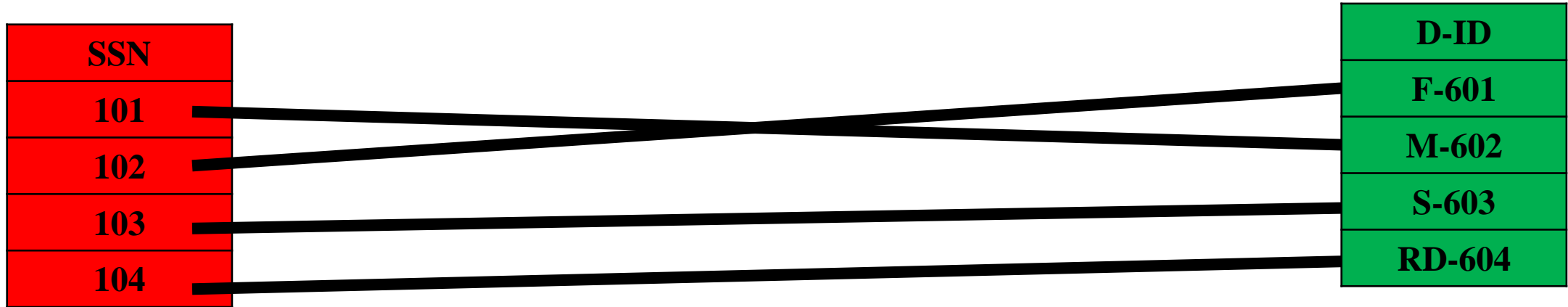
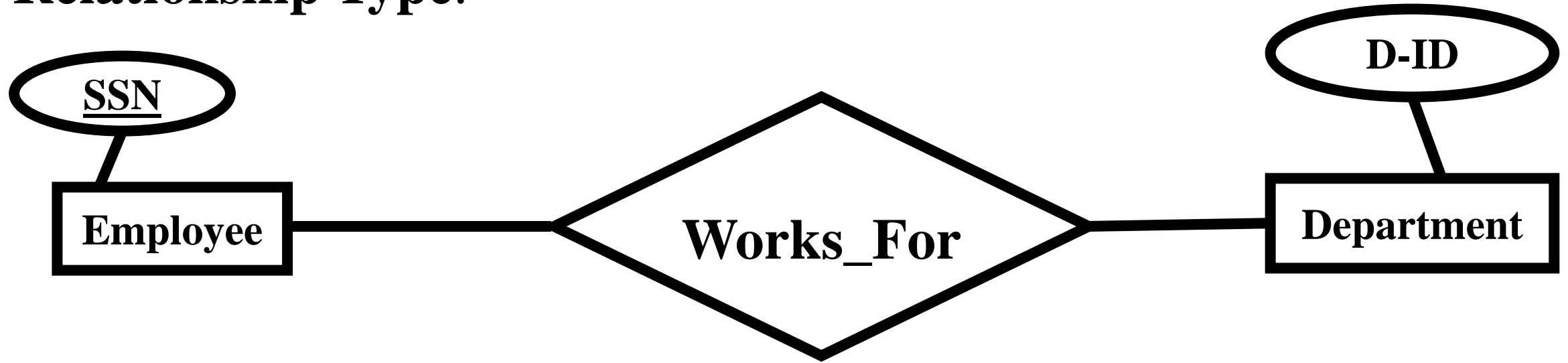
- **Relationship Type.**



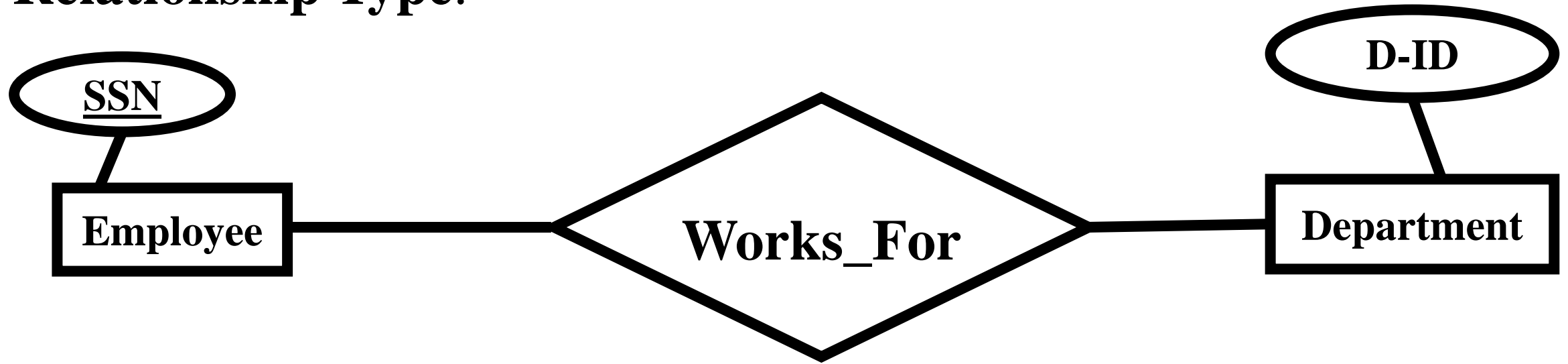
- **Relationship Type.**



- **Relationship Type.**



- **Relationship Type.**

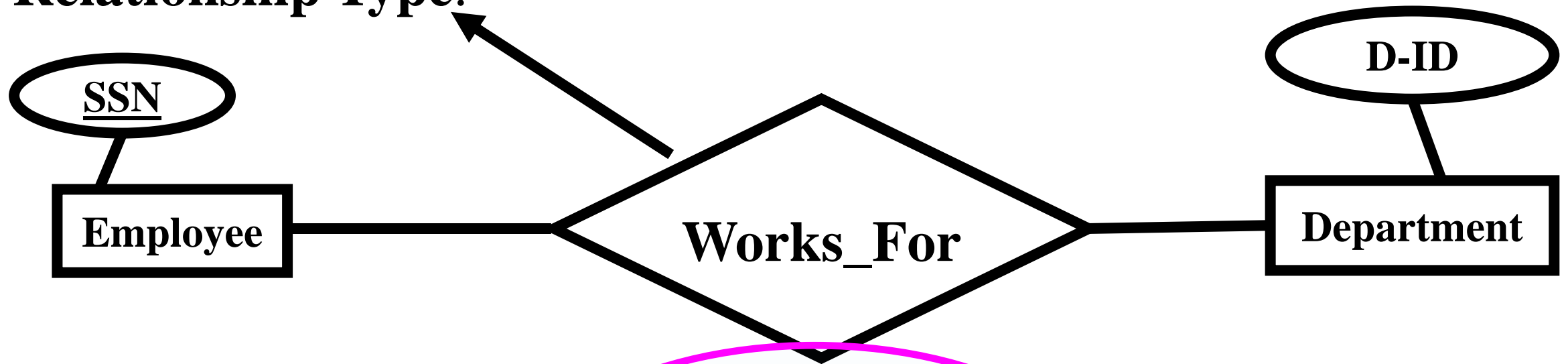


SSN
101
102
103
104

SSN	D-ID
101	M-602
102	F-601
103	S-603
104	RD-604

D-ID
F-601
M-602
S-603
RD-604

- **Relationship Type.**



SSN
101
102
103
104

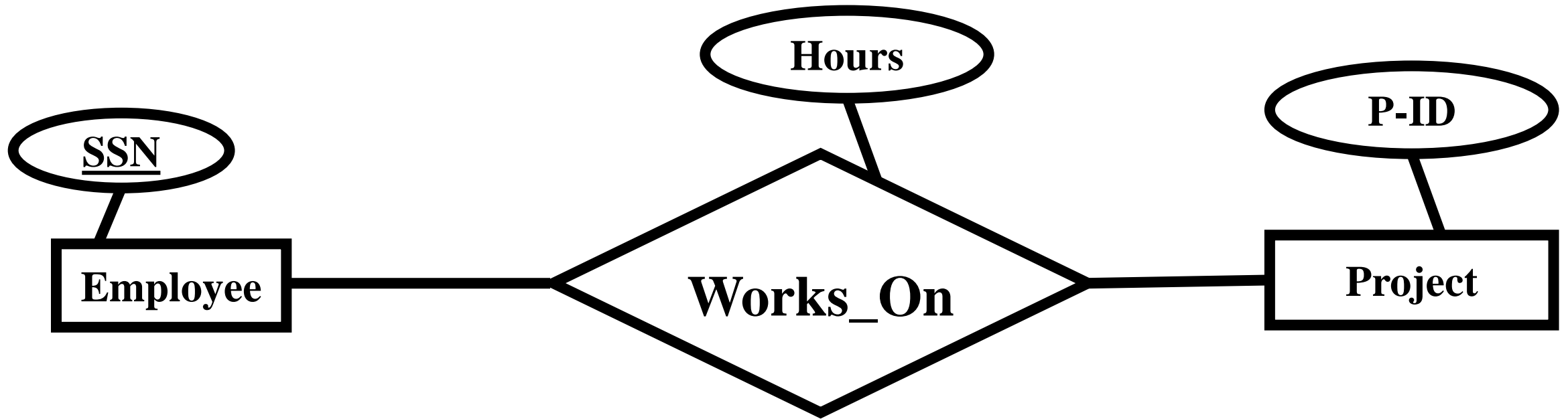
SSN	D-ID
101	M-602
102	F-601
103	S-603
104	RD-604

D-ID
F-601
M-602
S-603
RD-604

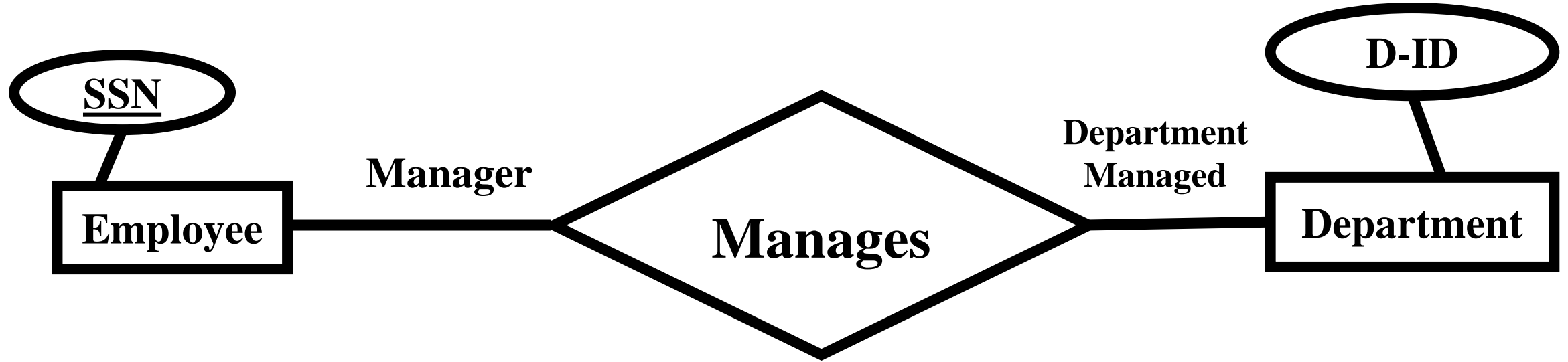
**Relationship Set**



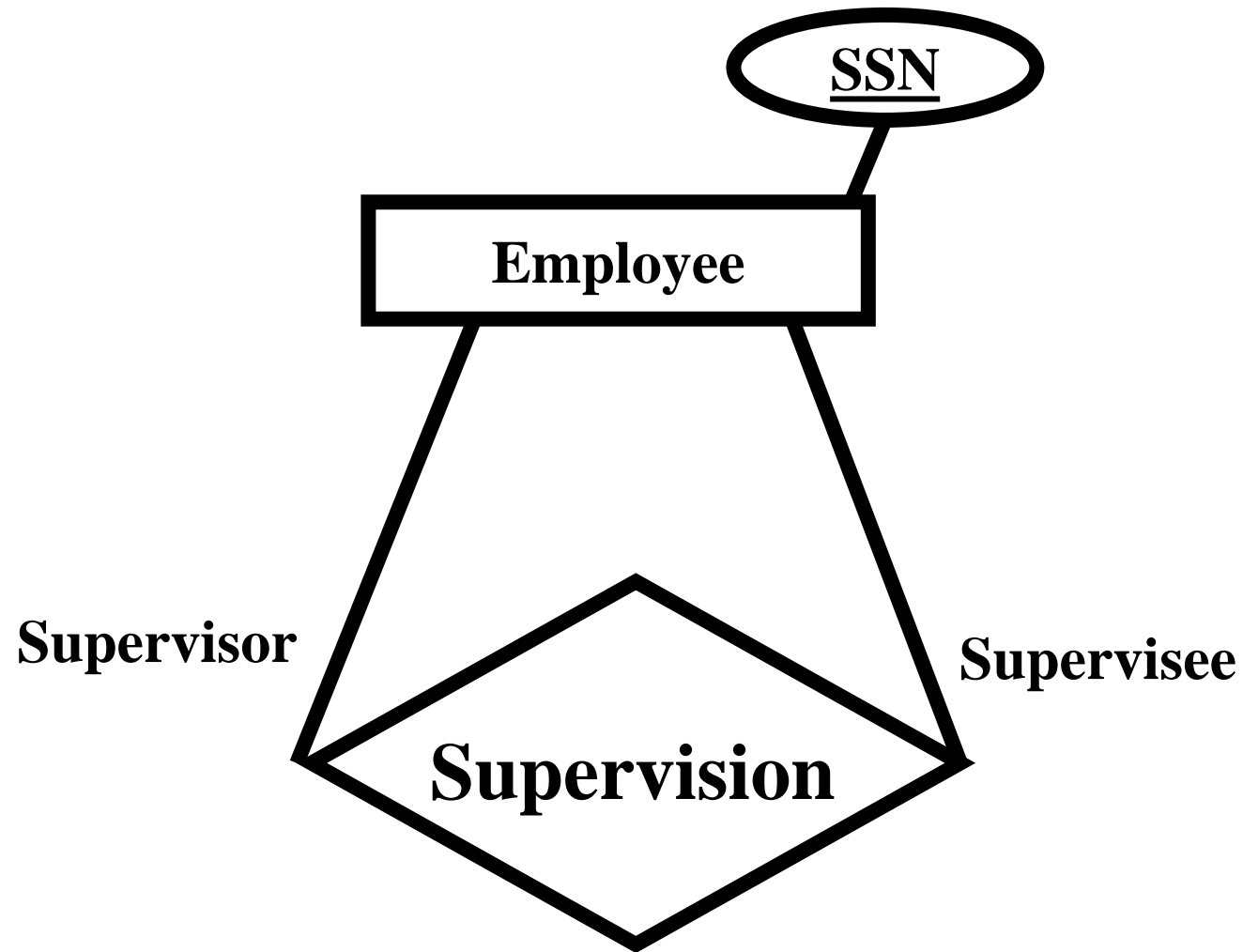
- **Degree of Relationship Type:**
- is the number of entity types participating in a Relationship Type.
- **Attributes of Relationship Type:**
- A Relationship type can have attributes.



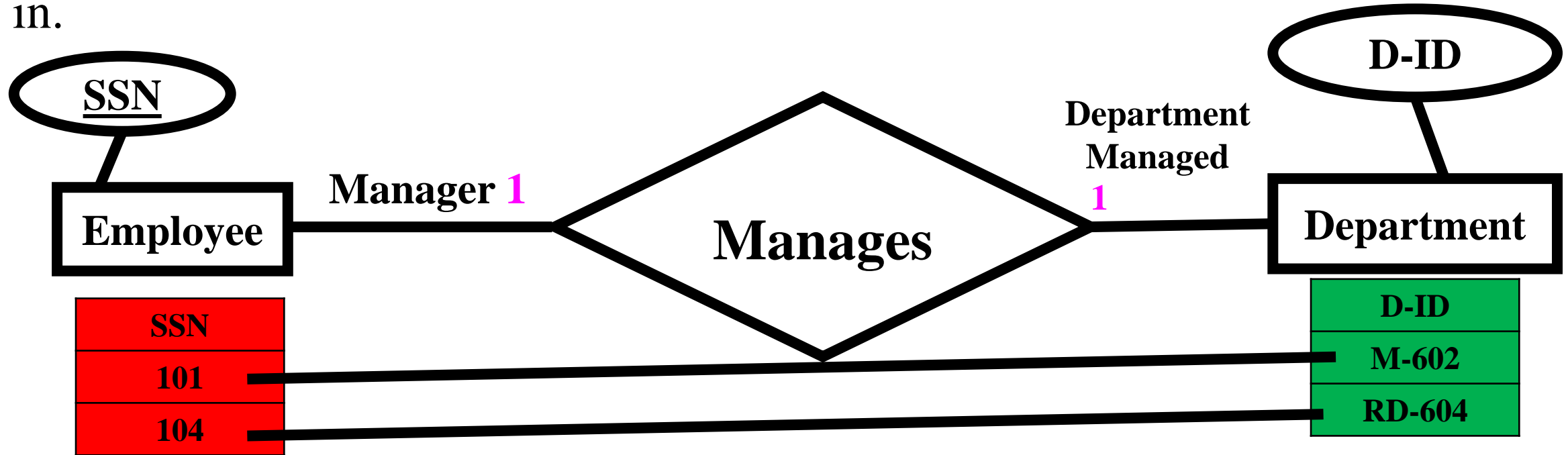
- **Role Names:**
- Specify the role of the participating entity type.



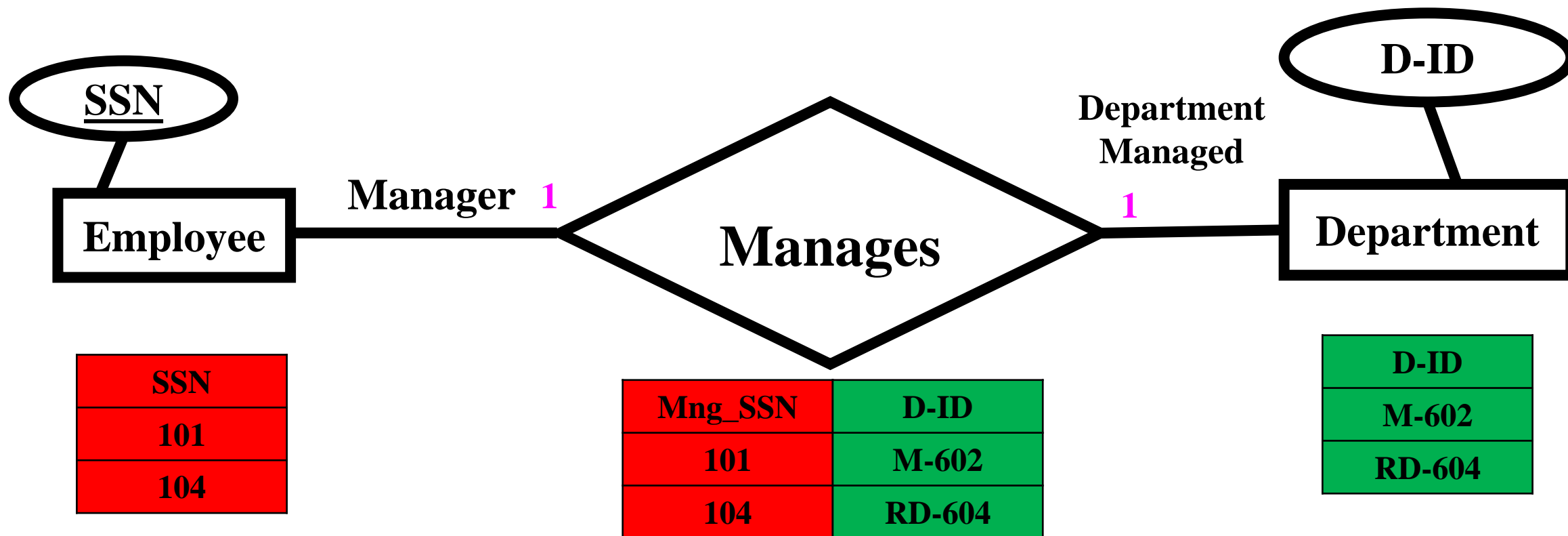
- Role Names are important in **Recursive relationship types**.



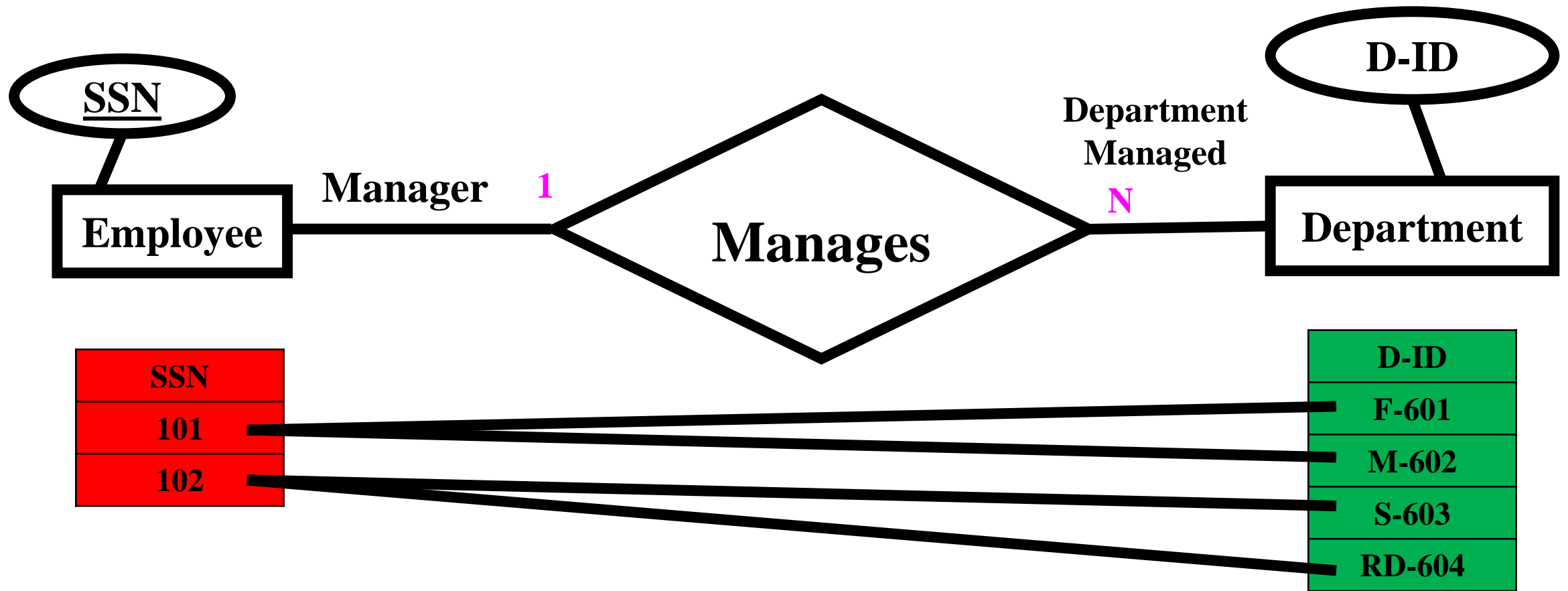
- **Constraints on Binary Relationship Types:**
- **Cardinality ratio:**
- It specifies the **maximum** number of relationship that an entity can participate in.



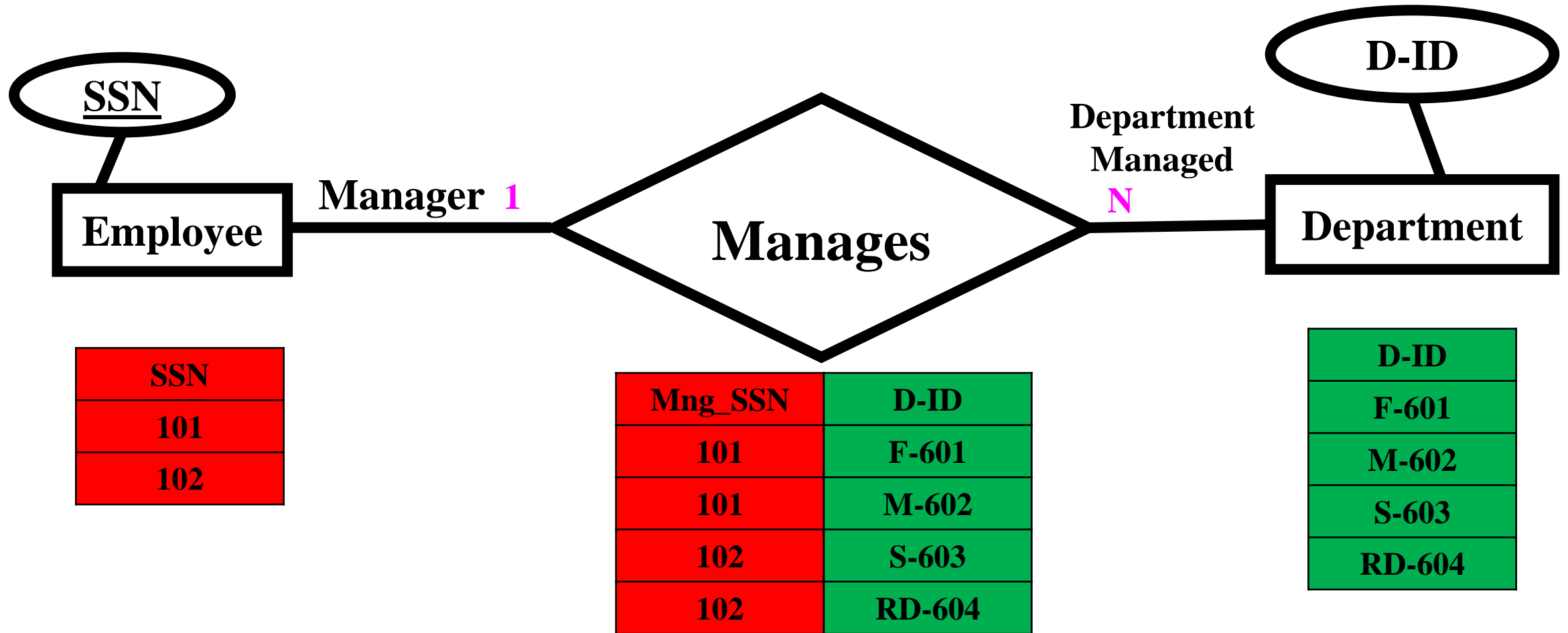
- Here, manages is a One-to-One (1:1) relationship



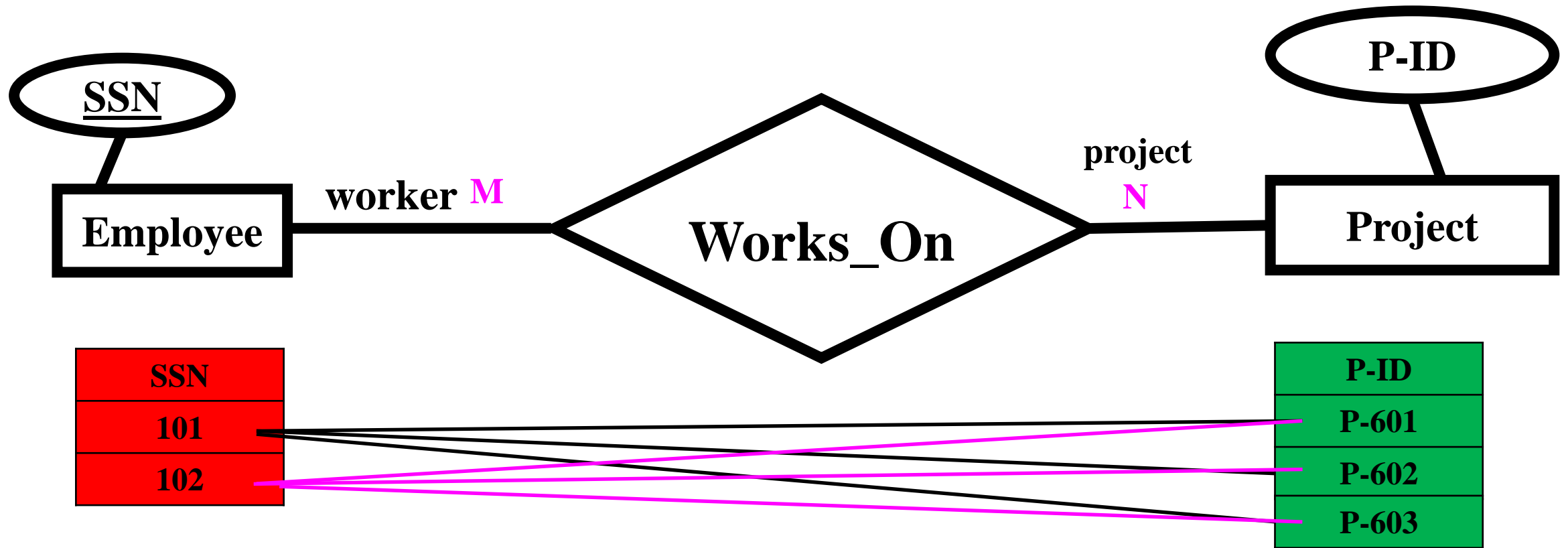
- Here, manages is a One-to-One (1:1) relationship



- Here, manages is a One-to-Many (1:N) relationship
- Where, N=2

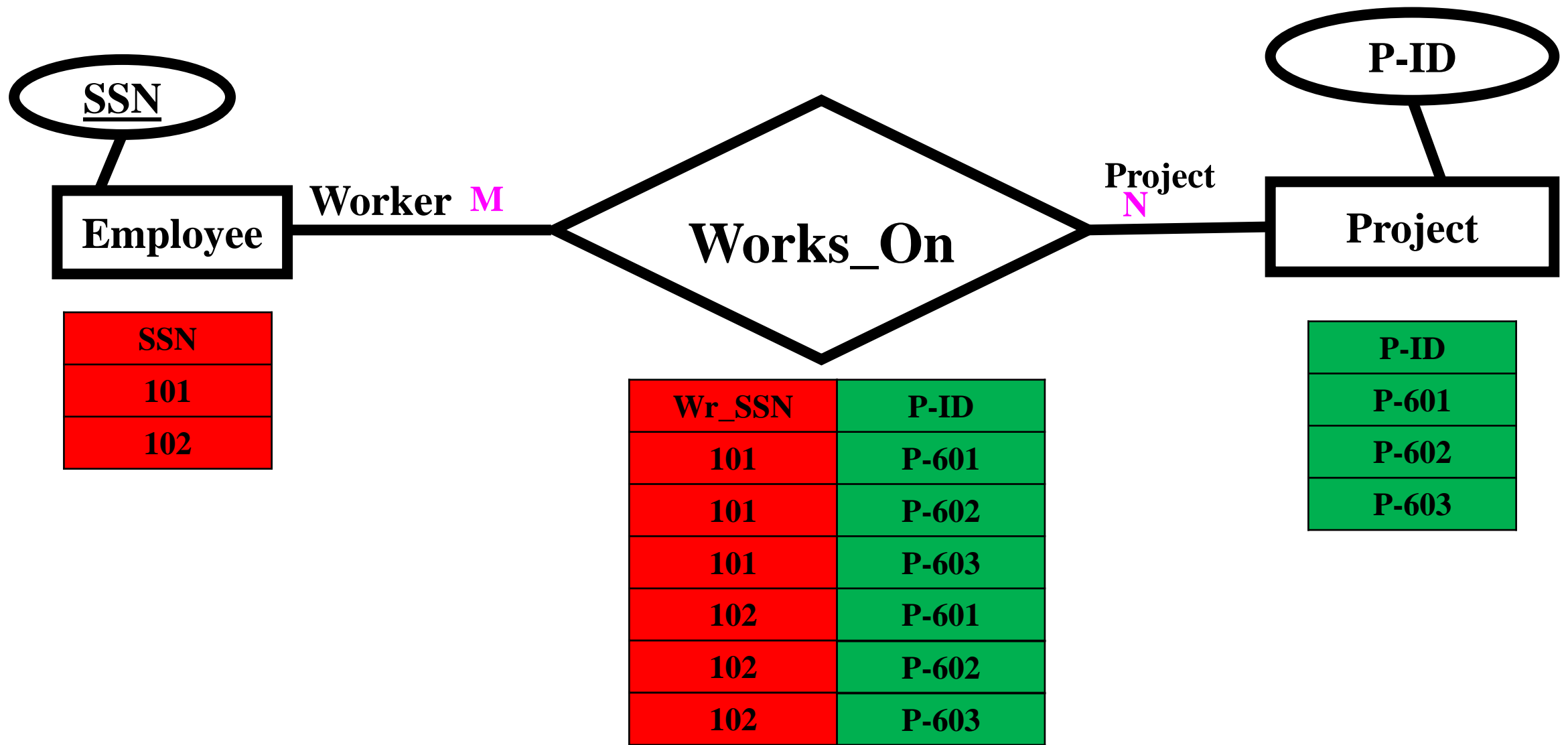


- Here, manages is a One-to-Many (1:N) relationship
- Where, N=2



- Here, Works\_On is a Many-to-Many (M:N) relationship
- Where, M=2:N=3

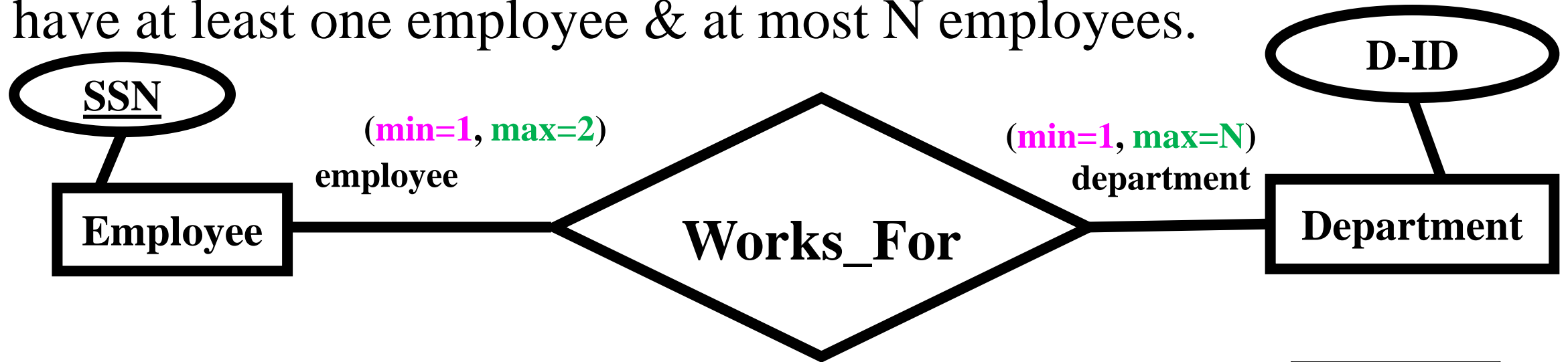




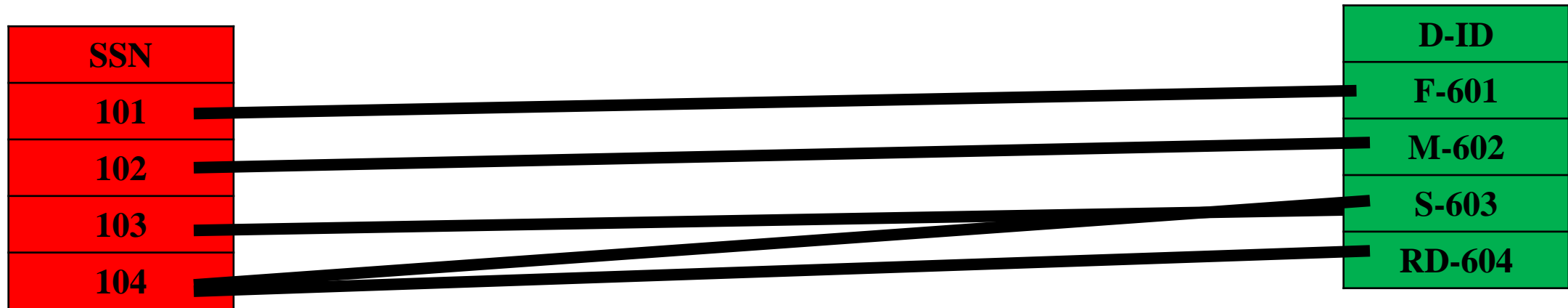
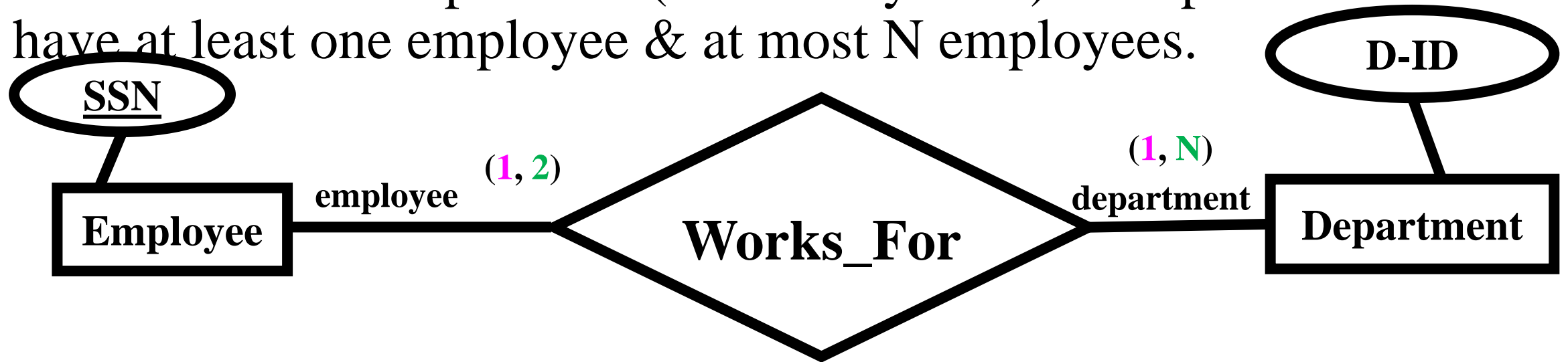
- Here, Works\_On is a Many-to-Many (M:N) relationship
- Where, M=2:N=3

- **Constraints on Binary Relationship Types:**
- **Participation/minimum cardinality constraint:**
  - It specifies the **minimum** number of relationship that each entity can participate in.
- **Types of participation constraint:**
  - 1. total participation
  - 2. partial participation

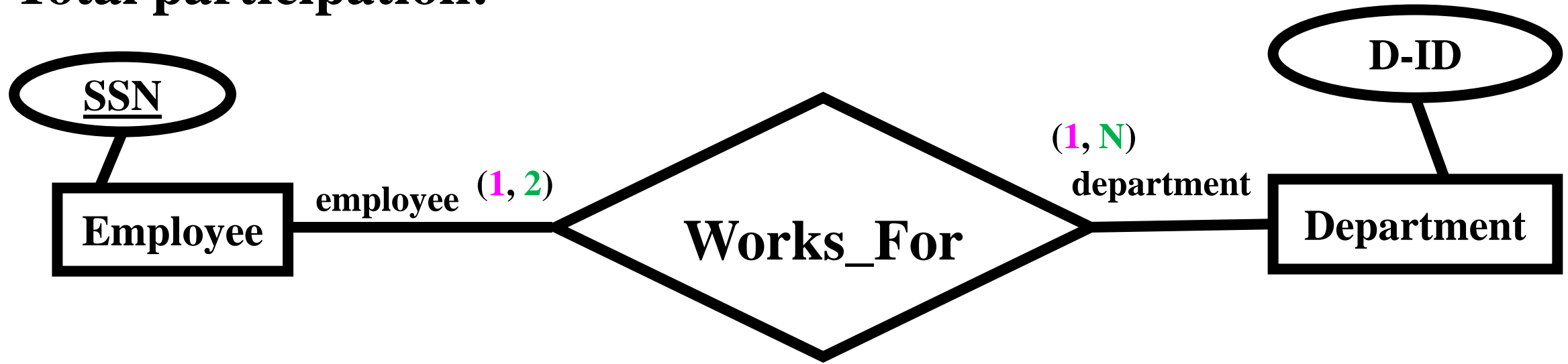
- **Total participation:**
- Constraint: Every employee must work for at least one department and at most two department(cardinality ratio). A department can have at least one employee & at most N employees.



- **Total participation:**
- Constraint: Every employee must work for at least one department and at most two department(cardinality ratio). A department can have at least one employee & at most N employees.



- **Total participation:**

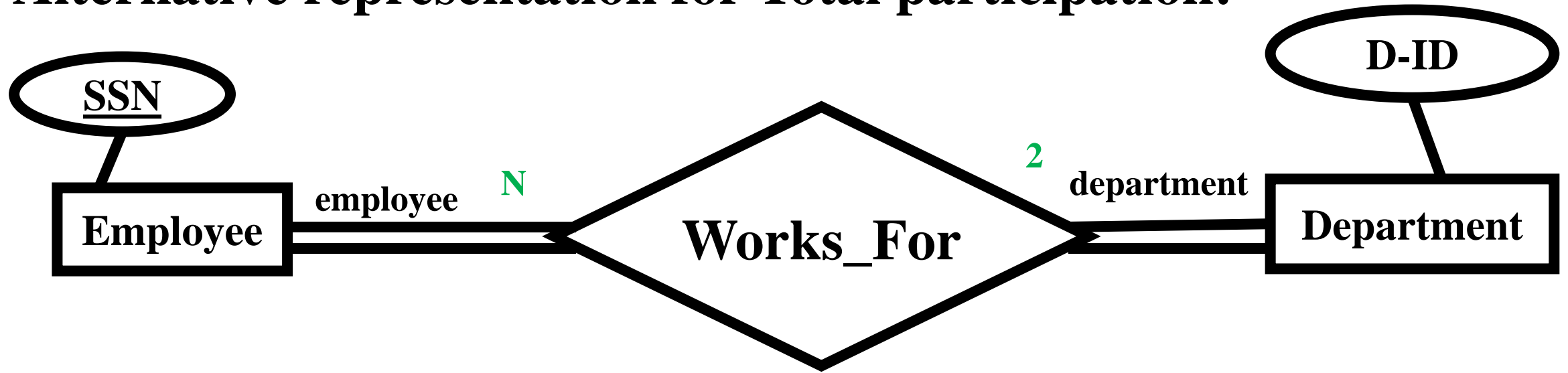


SSN
101
102
103
104

SSN	D-ID
101	F-601
102	M-602
103	S-603
104	S-603
104	RD-604

D-ID
F-601
M-602
S-603
RD-604

- Alternative representation for Total participation:

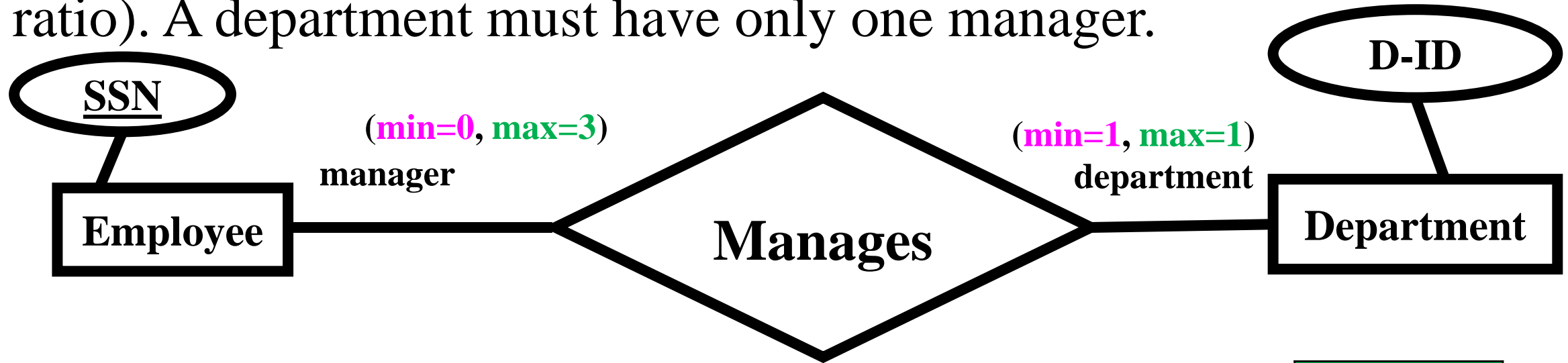


SSN
101
102
103
104

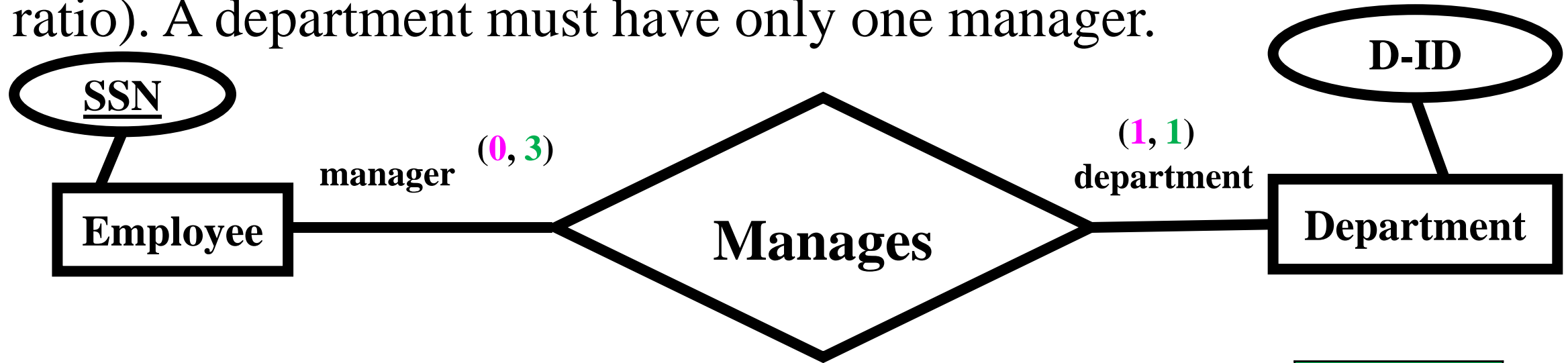
SSN	D-ID
101	F-601
102	M-602
103	S-603
104	S-603
104	RD-604

D-ID
F-601
M-602
S-603
RD-604

- **Partial participation:**
- Constraint: Some employees may not manage any department and some employee may manage at most 3 departments (cardinality ratio). A department must have only one manager.

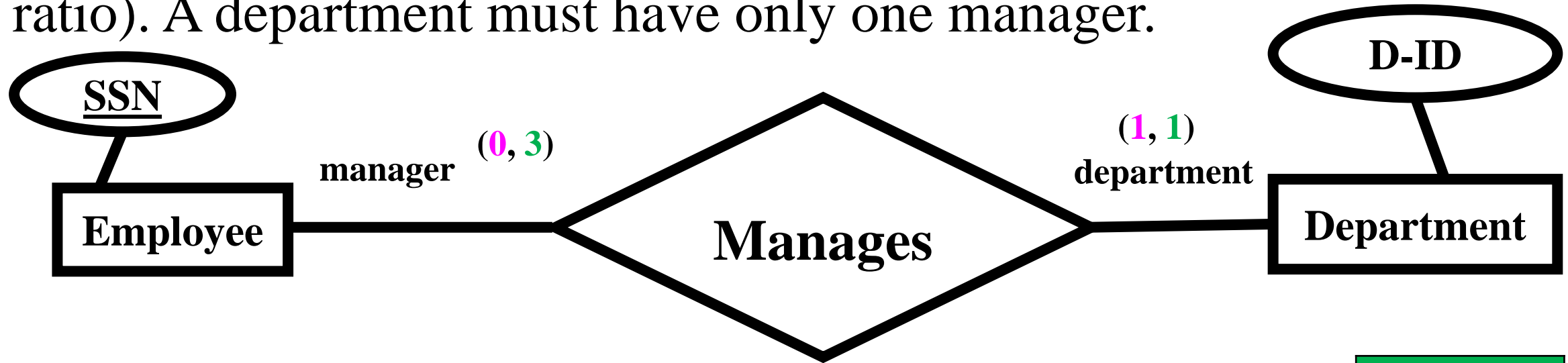


- **Partial participation:**
- Constraint: Some employees may not manage any department and some employee may manage at most 3 departments (cardinality ratio). A department must have only one manager.





- **Partial participation:**
- Constraint: Some employees may not manage any department and some employee may manage at most 3 departments (cardinality ratio). A department must have only one manager.

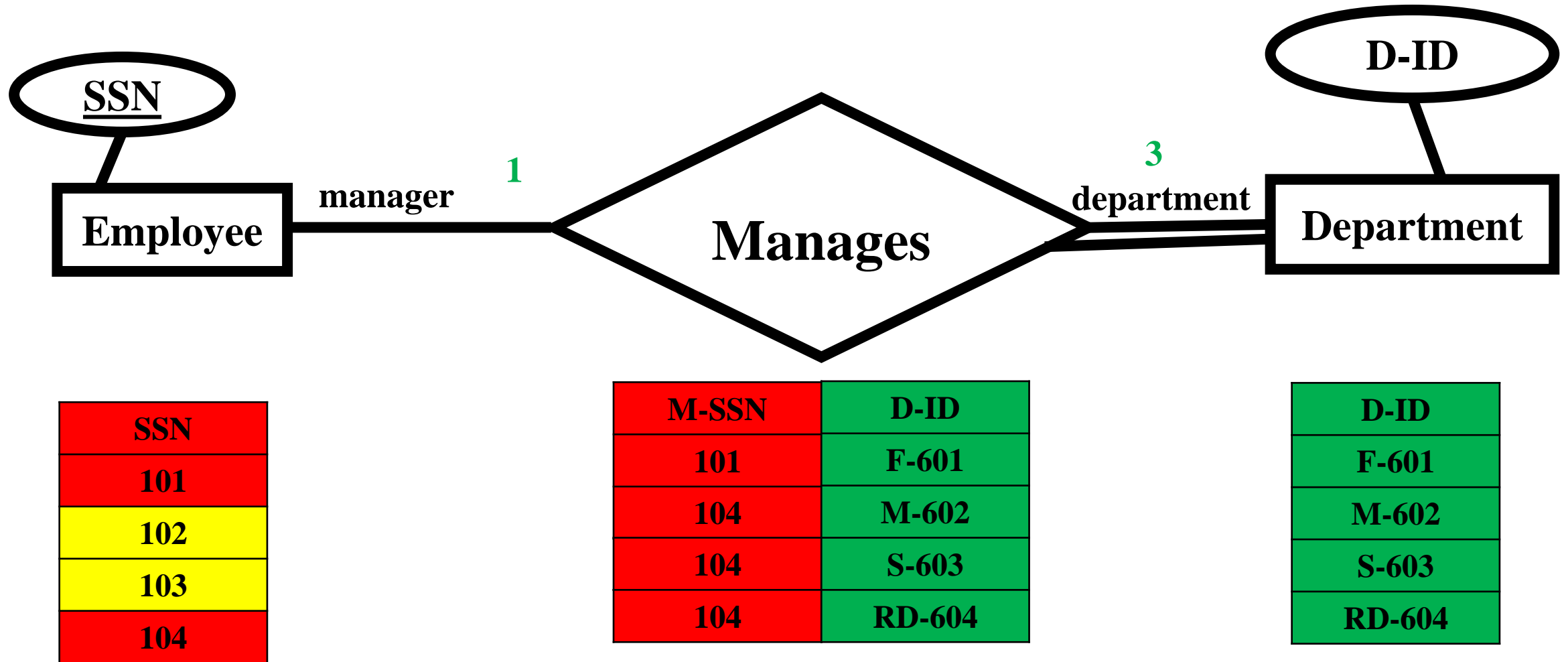


SSN
101
102
103
104

M-SSN	D-ID
101	F-601
104	M-602
104	S-603
104	RD-604

D-ID
F-601
M-602
S-603
RD-604

- Alternative representation for Partial participation:



- **Weak Entity Type:**

- Entity types that do not have key attributes of their own are called weak entities.

- **Example:**

- Dependents of employees

<b>Dependent_Name</b>	<b>Dependent_Type</b>
<b>Jude</b>	<b>Children</b>
<b>Jessica</b>	<b>Wife</b>
<b>Mary</b>	<b>wife</b>
<b>John</b>	<b>children</b>
<b>Jessica</b>	<b>children</b>

**Dependent**

- **Weak Entity Type:**

- Entity types that do not have key attributes of their own are called weak entities.

<u>SSN</u>	Name
<b>E101</b>	<b>James</b>
<b>E102</b>	<b>Jack</b>

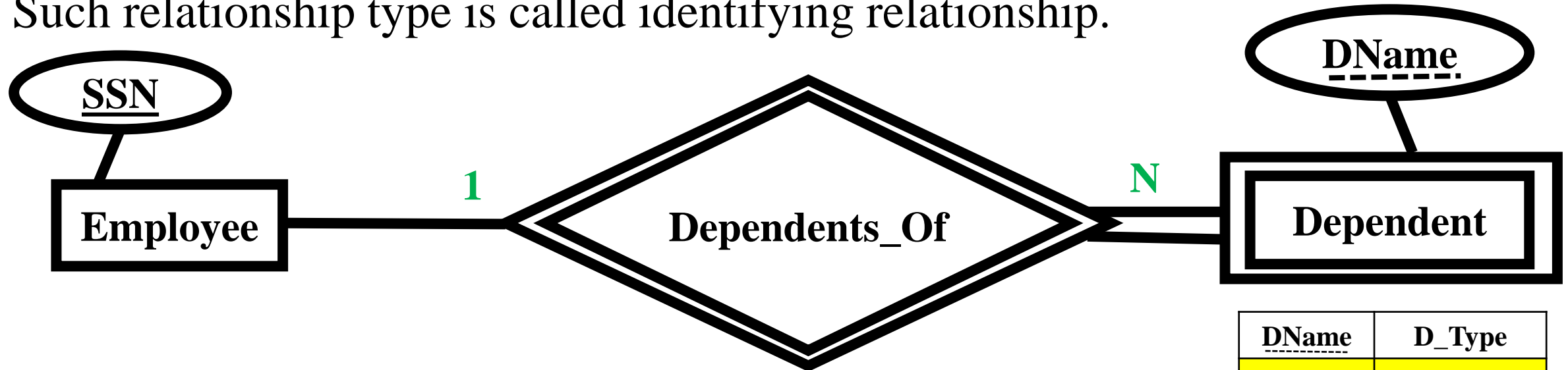
**Employee**

Dependent_Name	Dependent_Type
<b>Jude</b>	<b>Children</b>
<b>Jessica</b>	<b>Wife</b>
<b>Mary</b>	<b>wife</b>
<b>John</b>	<b>children</b>
<b>Jessica</b>	<b>children</b>

**Dependent**

- **Weak Entity Type:**

- Entity types should be identified with a strong entity type via a relationship type.
- Such relationship type is called identifying relationship.



<u>SSN</u>	Name
E101	James
E102	Jack

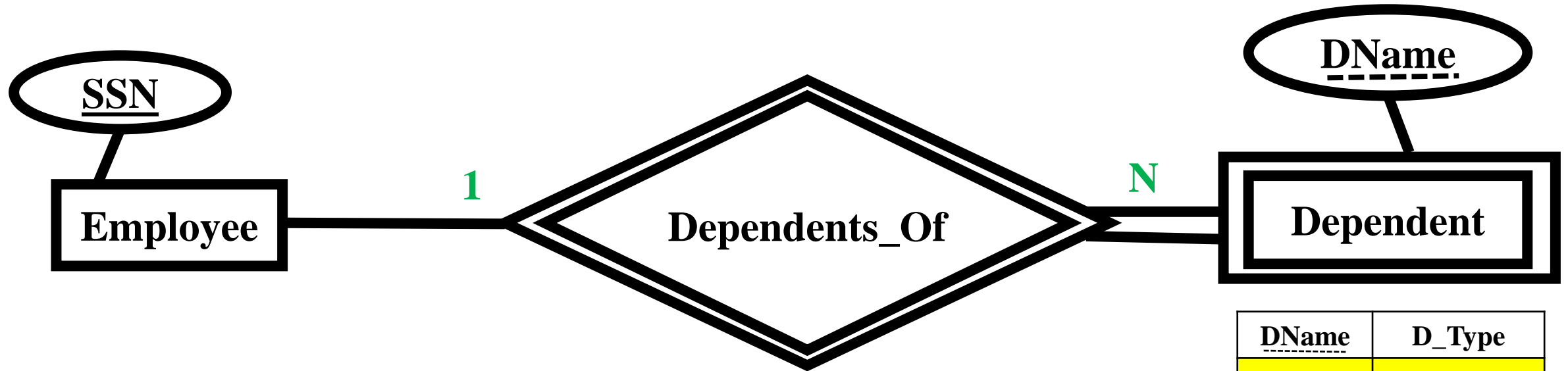
**Employee**

<u>SSN</u>	DName	D_Type
E101	Jude	Children
E101	Jessica	Wife
E102	Mary	wife
E102	John	children
E102	Jessica	children

<u>DName</u>	D_Type
Jude	Children
Jessica	wife
Mary	wife
John	children
Jessica	children

**Dependent**

- **Key of Weak Entity Type:**
- It is not unique
- It is called **partial key** and is underlined with a dotted line.



<u>SSN</u>	Name
E101	James
E102	Jack

**Employee**

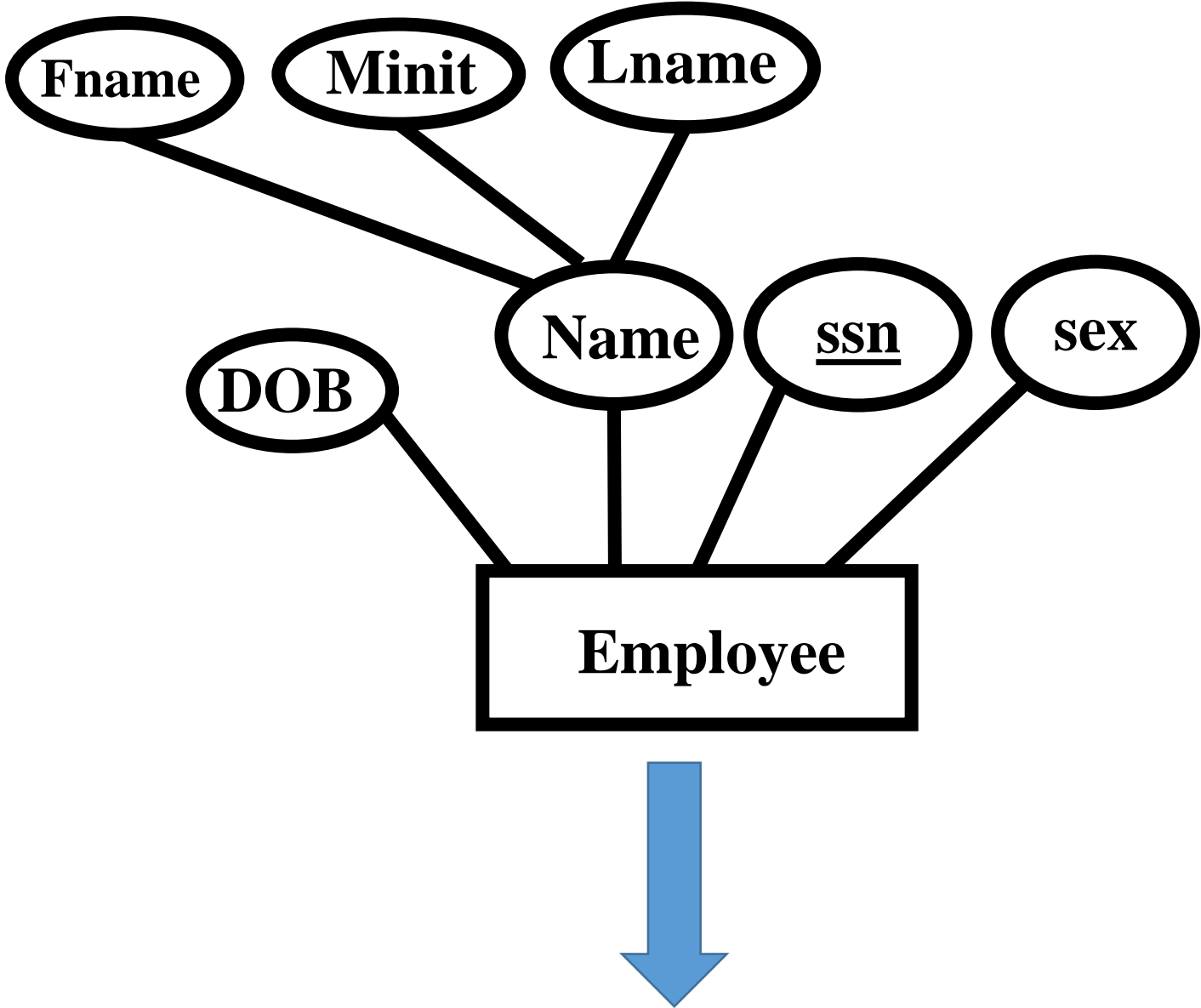
<u>SSN</u>	DName	D_Type
E101	Jude	Children
E101	Jessica	Wife
E102	Mary	wife
E102	John	children
E102	Jessica	children

<u>.....DName</u>	D_Type
Jude	Children
Jessica	wife
Mary	wife
John	children
Jessica	children

**Dependent**

- **Mapping a Conceptual Design into a Logical Design:**
- **Step-1: Mapping of strong/regular entity types.**
- For each strong entity type 'E', create a relation 'R' and identify a super keys, candidate keys and primary key.
- Relation 'R' must contain **only simple attributes**.
- For **composite attributes** of entity type, only their constituent simple attributes must be included in its relation 'R'.
- For **multivalued attributes** of entity type,
  1. the multivalued attributes should be treated as a separate entity type
  2. Create a relation for the multivalued attributes entity type.
  3. Link the relation for the multivalued attributes entity type with its parent relation using foreign keys.

**Example:**



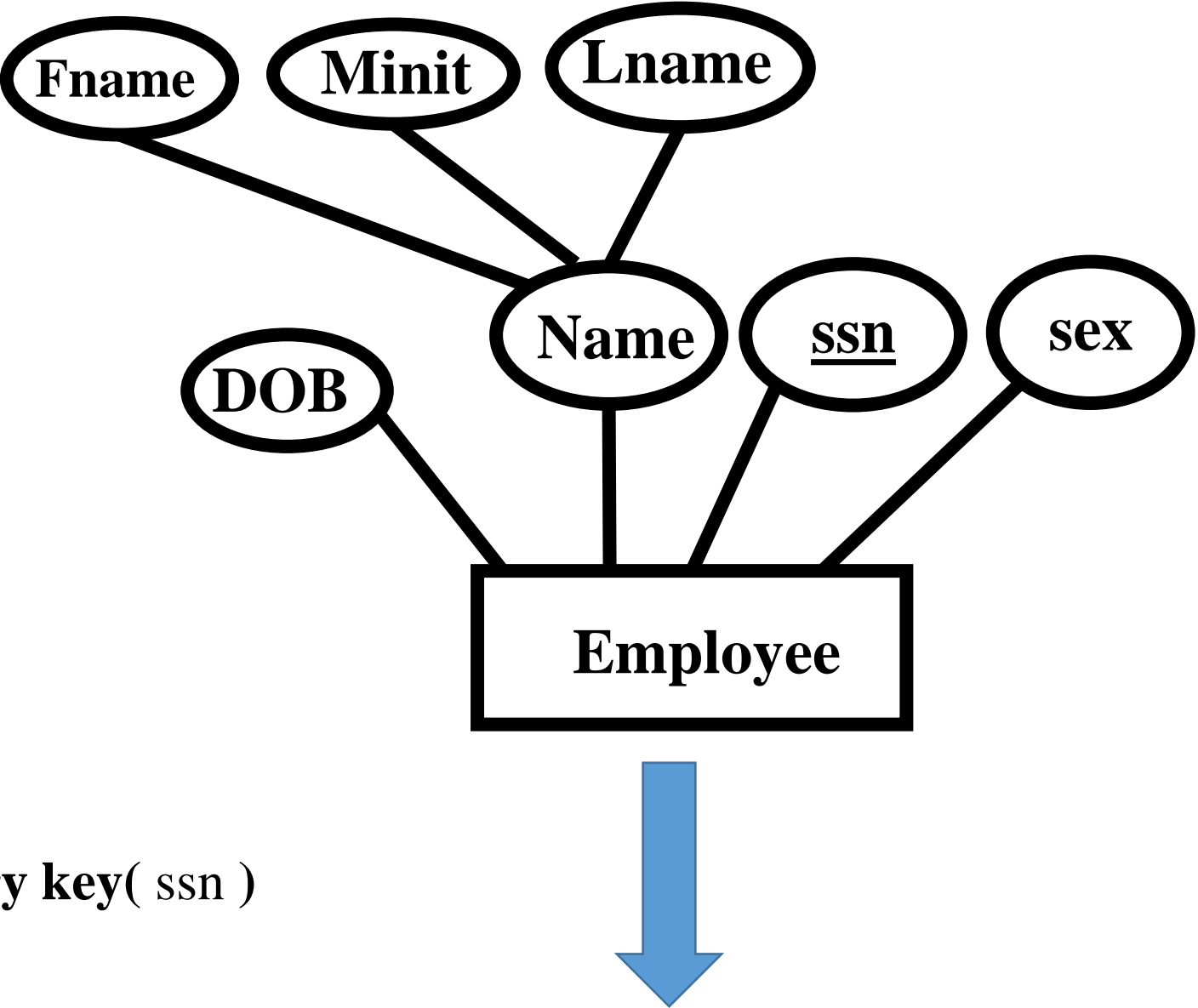
**Employee**

<u>SSN</u>	Fname	Minit	Lname	DOB	Sex
------------	-------	-------	-------	-----	-----



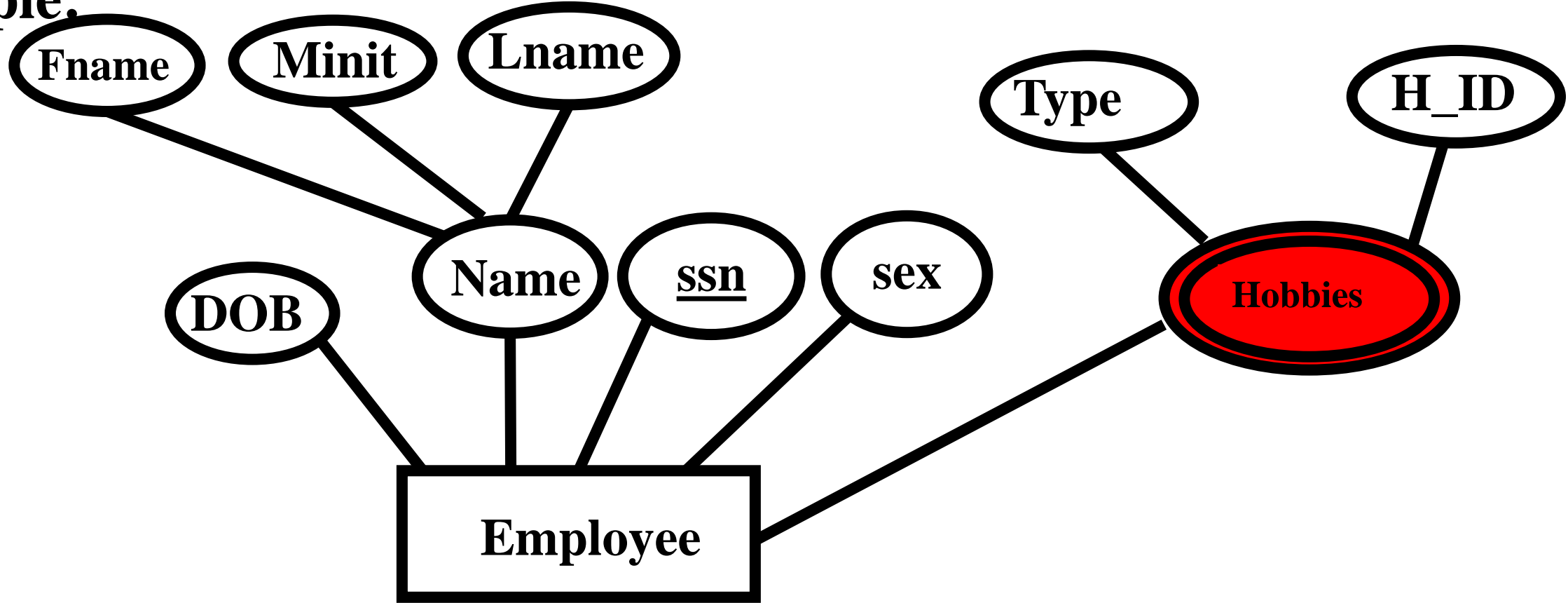
Example:

```
create table employee
(  
  ssn varchar2(10),  
  Fname varchar2(30) not null,  
  Minit varchar2(4),  
  Lname varchar2(30),  
  DOB date,  
  sex varchar2(1),  
  constraint pk_employee primary key( ssn )  
)
```



Employee	<u>SSN</u>	Fname	Minit	Lname	DOB	Sex
----------	------------	-------	-------	-------	-----	-----

Example:



Employee



<u>SSN</u>	Fname	Minit	Lname	DOB	Sex	H-ID	H_Name	H-Type
------------	-------	-------	-------	-----	-----	------	--------	--------

- **Example:**

## Employee

<u>SSN</u>	Fname	Minit	Lname	DOB	Sex	H-ID	H_Name	H-Type
101	john	n	taylor	02-02-1996	M	H01	tracking	outdoor
101	john	n	taylor	02-02-1996	M	H02	gardening	outdoor
102	breet	P	lee	04-07-1980	M	H01	tracking	outdoor

## Employee

<u>SSN</u>	Fname	Minit	Lname	DOB	Sex
101	john	n	taylor	02-02-1996	M
102	breet	P	lee	04-07-1980	M

## Hobby

<u>H-ID</u>	H_Name	H-Type	<u>SSN</u>
H01	tracking	outdoor	101
H02	gardening	outdoor	101
H01	tracking	outdoor	102

# Employee

<u>SSN</u>	Fname	Minit	Lname	DOB	Sex
101	john	n	taylor	02-02-1996	M
102	breet	P	lee	04-07-1980	M

# Hobby

<u>H-ID</u>	H_Name	H-Type	<u>SSN</u>
H01	tracking	outdoor	101
H02	gardening	outdoor	101
H01	tracking	outdoor	102

**create table employee**

```
(  
    ssn varchar2(10),  
    Fname varchar2(30) not null,  
    Minit varchar2(4),  
    Lname varchar2(30),  
    DOB date,  
    sex varchar2(1),  
    constraint pk_employee primary key( ssn )  
)
```

**create table hobby**

```
(  
    H-ID varchar2(10),  
    H_name varchar2(30) not null,  
    H-Type varchar2(4),  
    ssn varchar2(10),  
    constraint pk_hobby primary key( H-ID, ssn ),  
    constraint fk_hobby foreign key(ssn) references  
        employee(ssn) on delete cascade  
)
```

- **Mapping a Conceptual Design into a Logical Design:**
- **Step-2: Mapping of Weak entity types.**
- For a weak entity type 'W' of strong entity type 'E', create a relation 'R'.
- Include the simple attributes of weak entity type 'W' in relation 'R'.
- Also, include the primary keys of the strong entity type 'E' as foreign keys in 'R'.
- The primary key of 'R' is formed by combining the primary key of the strong entity type and weak entity type.

• **Example:**

SSN

**Employee**

<u>SSN</u>	Fname	Minit	Lname	DOB	Sex
------------	-------	-------	-------	-----	-----

**Employee**

DName

**Dependent**

**Dependent**

<u>DName</u>	D_Type
--------------	--------

**Dependent**

<u>SSN</u>	<u>DName</u>	D_Type
------------	--------------	--------

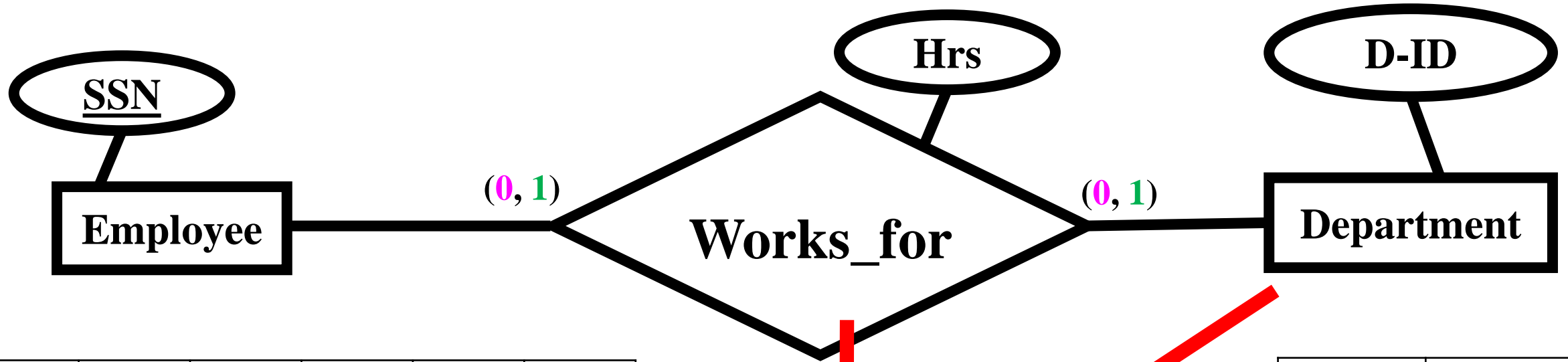
**create table dependent**

```
(  
  dname varchar2(10) not null,  
  D_type varchar2(30),  
  ssn varchar2(10) not null,  
  constraint fk_dependant foreign key(ssn) references  
    employee(ssn) on delete cascade  
)
```

- **Mapping a Conceptual Design into a Logical Design:**
- **Step-3: Mapping of Binary 1:1 Relationship Types.**
- **Case-1 (when the participations are only partial):**
- **Create a separate relation 'R' for a relationship type**, by combining the primary keys of the partially participating entity types 'E' and 'D' along with the attributes of the relationship type.
- The primary keys of the partially participating entity types 'E' and 'D' are foreign keys in 'R'.
- Primary key of either 'E' or 'D' can be made as the primary key of 'R'.
- **Draw back:**
- Needs extra relation
- Needs extra join operation

- **Step-3: Mapping of Binary 1:1 Relationship Types.**

- **Case-1 (when the participations are only partial):**



<u>SSN</u>	Fname	Minit	Lname	DOB	Sex
101	John	N	Taylor	02/03/88	M
102	Mary	L	Smith	02/03/92	F
103	Mark	N	Lee	02/03/82	M

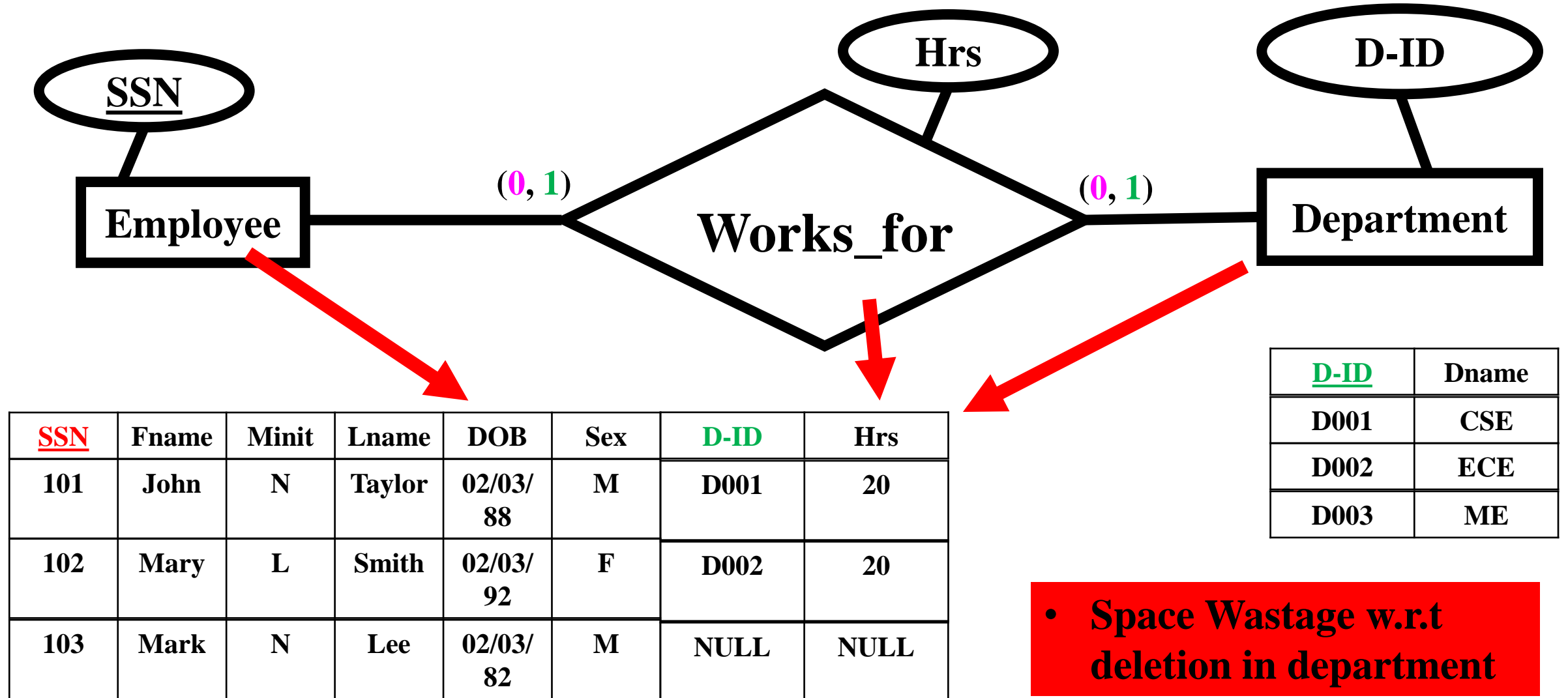
<u>SSN</u>	<u>D-ID</u>	Hrs
101	D001	20
102	D002	20
1003	NULL	NULL
NULL	D003	NULL

<u>D-ID</u>	Dname
D001	CSE
D002	ECE
D003	ME

- **Primary Key Problem**
- **Space Wastage**

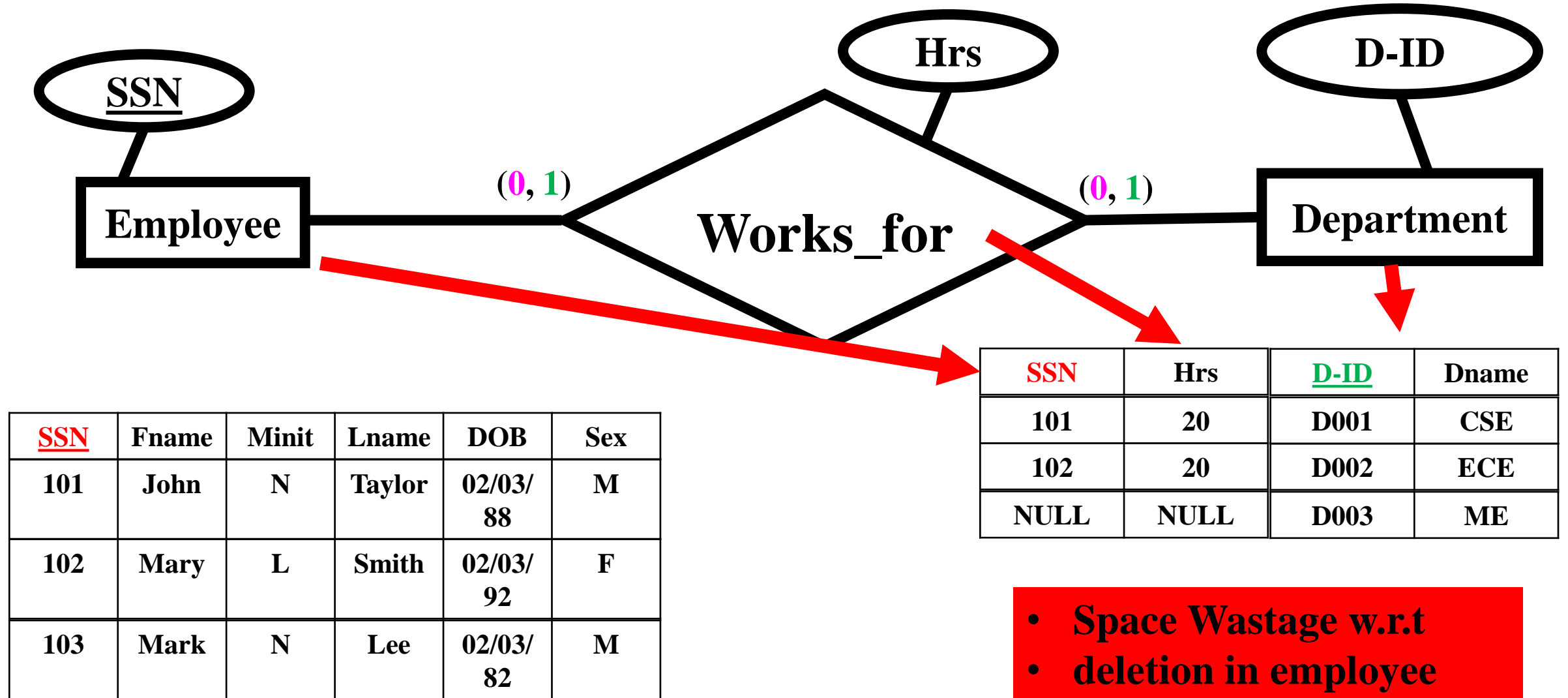


- **Step-3: Mapping of Binary 1:1 Relationship Types.**
- **Case-1 (when the participations are only partial):**



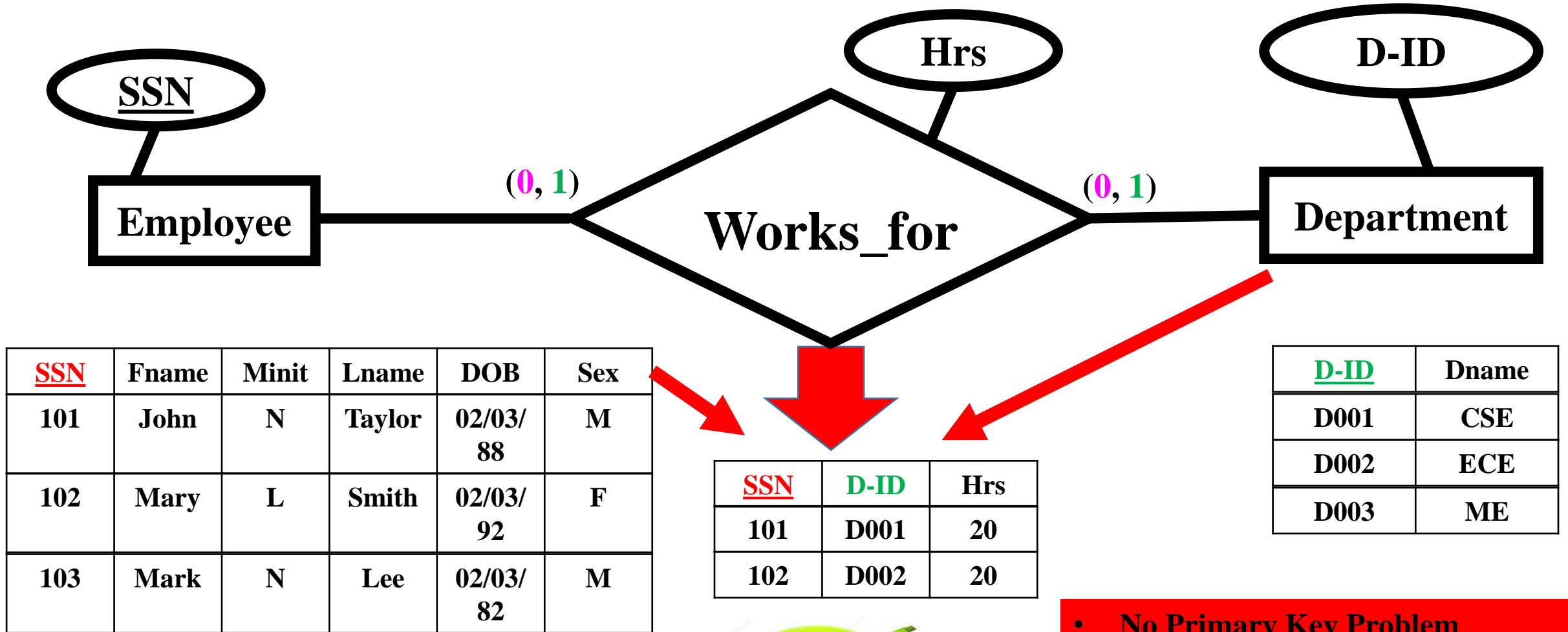
- **Step-3: Mapping of Binary 1:1 Relationship Types.**

- **Case-1 (when the participations are only partial):**



## • Step-3: Mapping of Binary 1:1 Relationship Types.

### • Case-1 (when the participations are only partial):



- No Primary Key Problem
- No Space Wastage w.r.t deletion in either employee or department

- **Step-3: Mapping of Binary 1:1 Relationship Types.**

- **Case-1 (when the participations are only partial):**

### Employee

<u>SSN</u>	Fname	Minit	Lname	DOB	Sex
------------	-------	-------	-------	-----	-----

### Works\_for

<u>SSN</u>	D-ID	Hrs
------------	------	-----

### Department

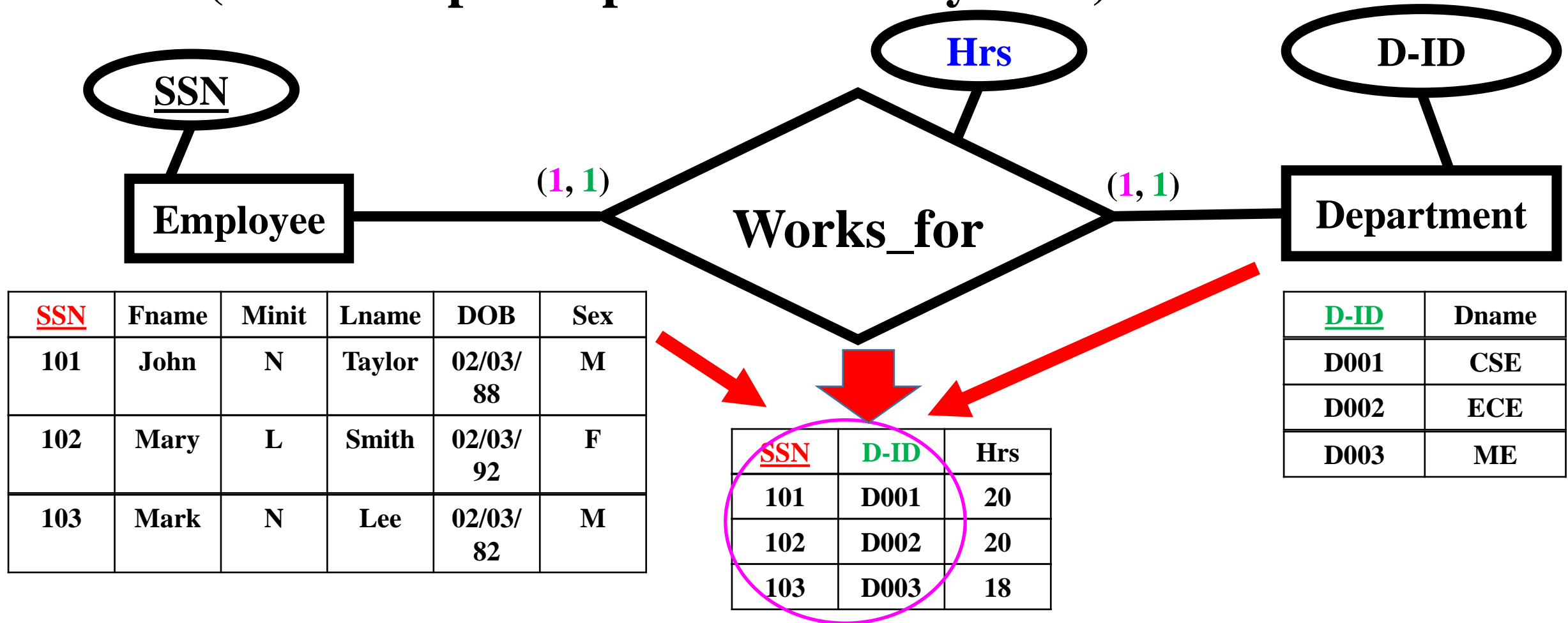
<u>D-ID</u>	Dname
-------------	-------

```
create table works_for
(
    ssn varchar2(10),
    D-ID varchar2(10) not null,
    hrs varchar2(4),
    constraint pk_ssn primary key( ssn ),
    constraint fk_works_1 foreign key(ssn) references employee(ssn) on delete cascade
    constraint fk_works_2 foreign key(D-ID) references department(D-ID) on delete
                                     cascade
)
```

- **Step-3: Mapping of Binary 1:1 Relationship Types.**
- **Case-2(when the participations are only total):**
  - Create a single new relation 'R' by merging the attributes of the two totally participating entity types 'E' and 'D'.
  - Select the either of the primary keys of the two totally participating entity types as the primary key of the new relation 'R'.

- **Step-3: Mapping of Binary 1:1 Relationship Types.**

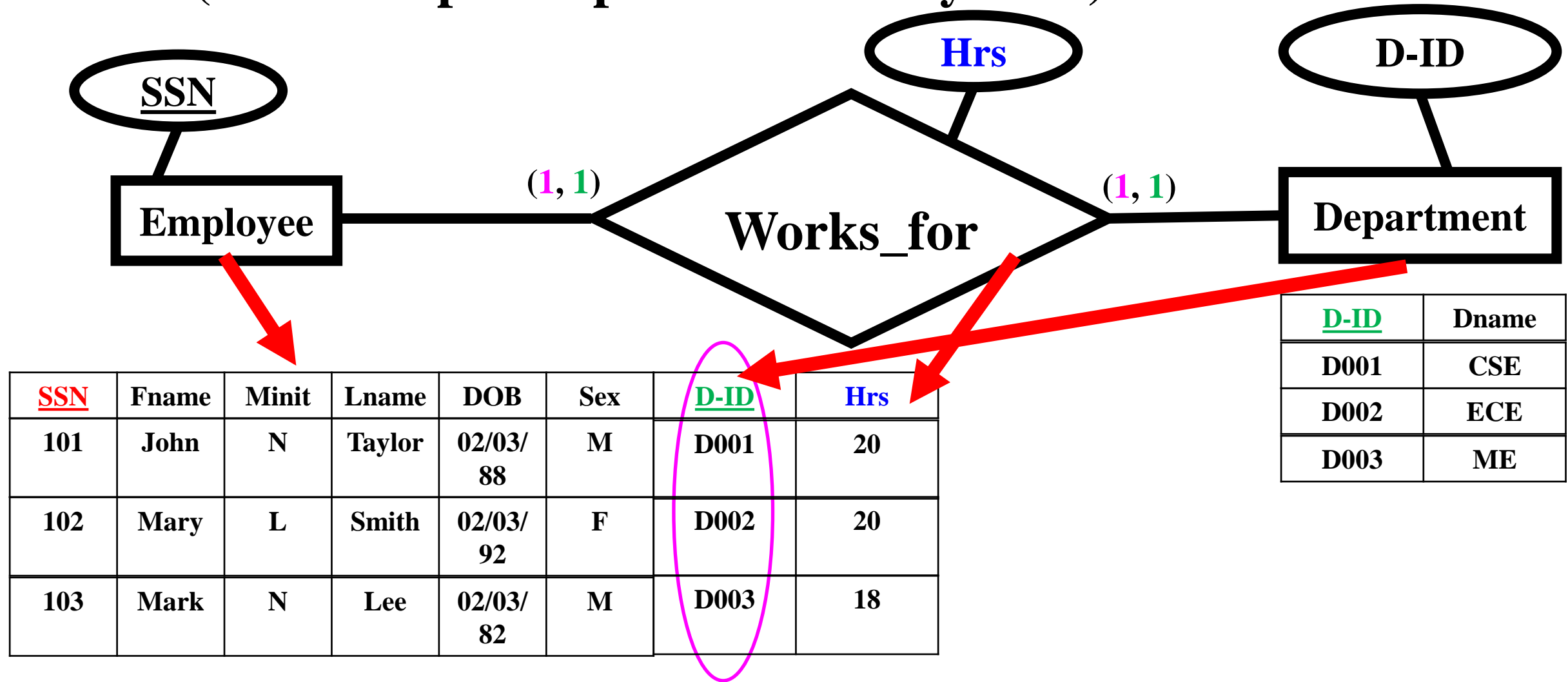
- **Case-2 (when the participations are only total):**



- **Space Wastage**

- **Step-3: Mapping of Binary 1:1 Relationship Types.**

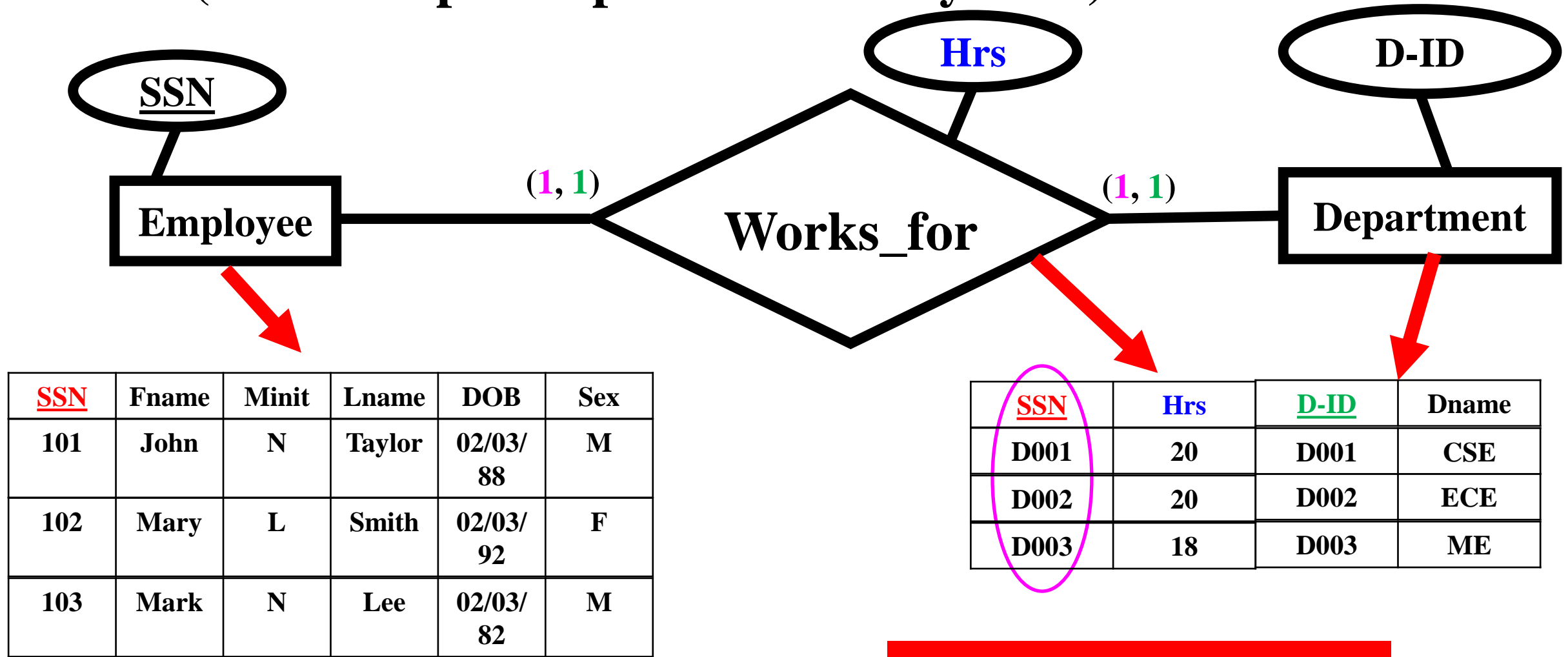
- **Case-2 (when the participations are only total):**



- **Space Wastage**

- **Step-3: Mapping of Binary 1:1 Relationship Types.**

- **Case-2 (when the participations are only total):**

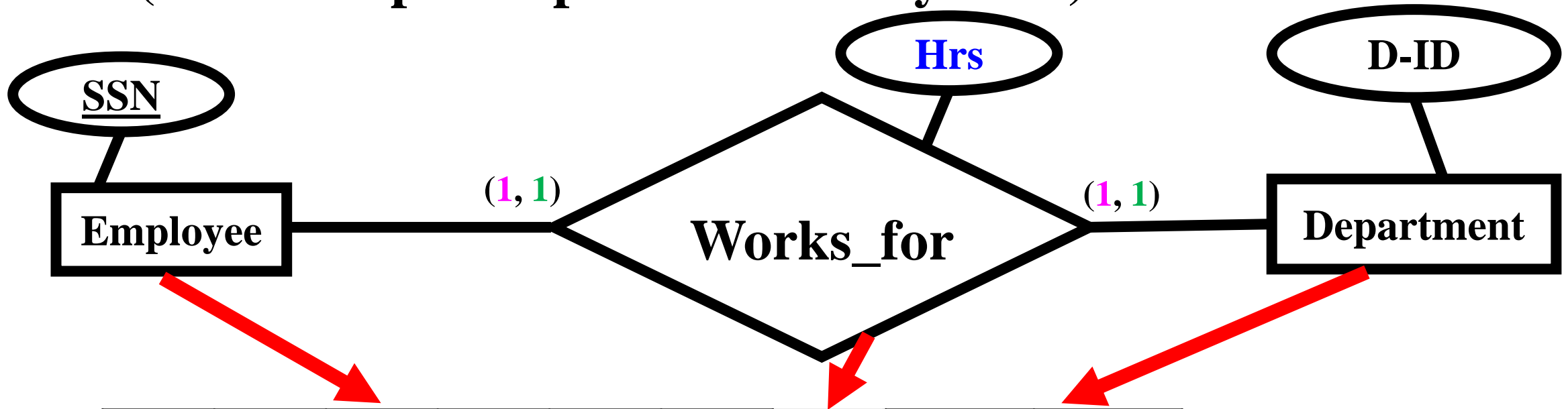


- **Space Wastage**



- **Step-3: Mapping of Binary 1:1 Relationship Types.**

- **Case-2 (when the participations are only total):**

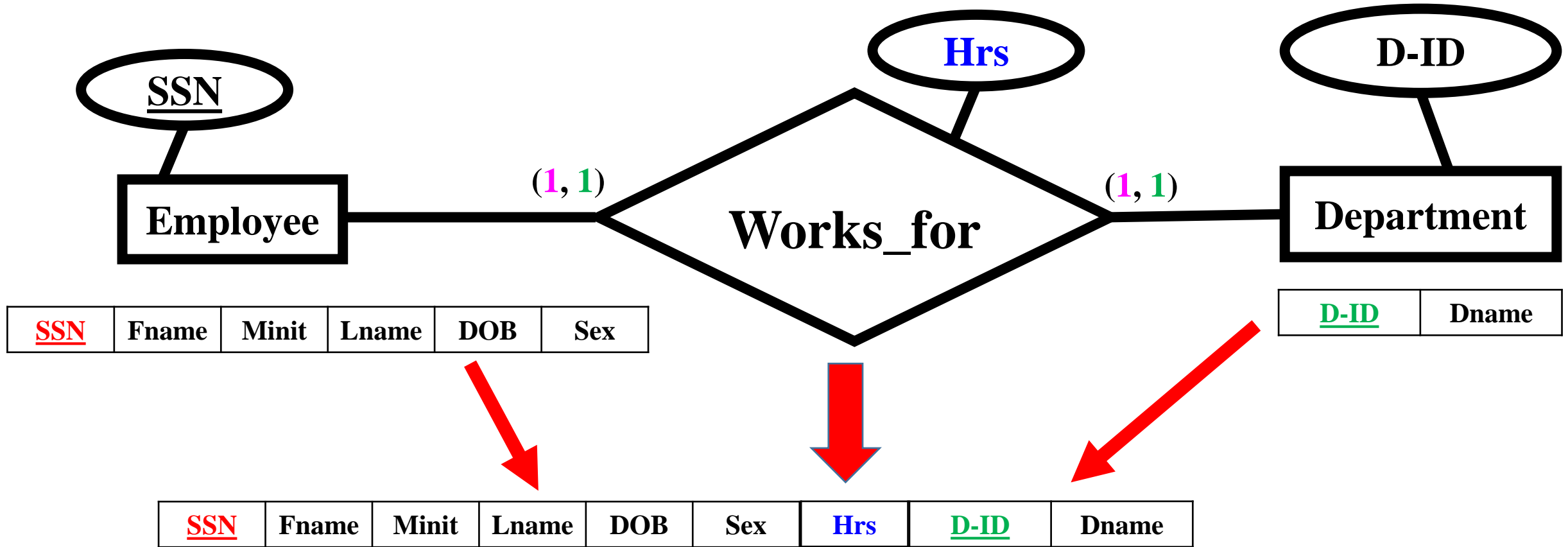


<u>SSN</u>	Fname	Minit	Lname	DOB	Sex	Hrs	D-ID	Dname
101	John	N	Taylor	02/03/88	M	20	D001	CSE
102	Mary	L	Smith	02/03/92	F	20	D002	ECE
103	Mark	N	Lee	02/03/82	M	18	D003	ME



- **No Space Wastage**

- **Step-3: Mapping of Binary 1:1 Relationship Types.**
- **Case-2 (when the participations are only total):**



- **Step-3: Mapping of Binary 1:1 Relationship Types.**
- **Case-2**(when the participations are only total):

**Employee      Works\_for      Department**

<u>SSN</u>	Fname	Minit	Lname	DOB	Sex	Hrs	<u>D-ID</u>	Dname
------------	-------	-------	-------	-----	-----	-----	-------------	-------

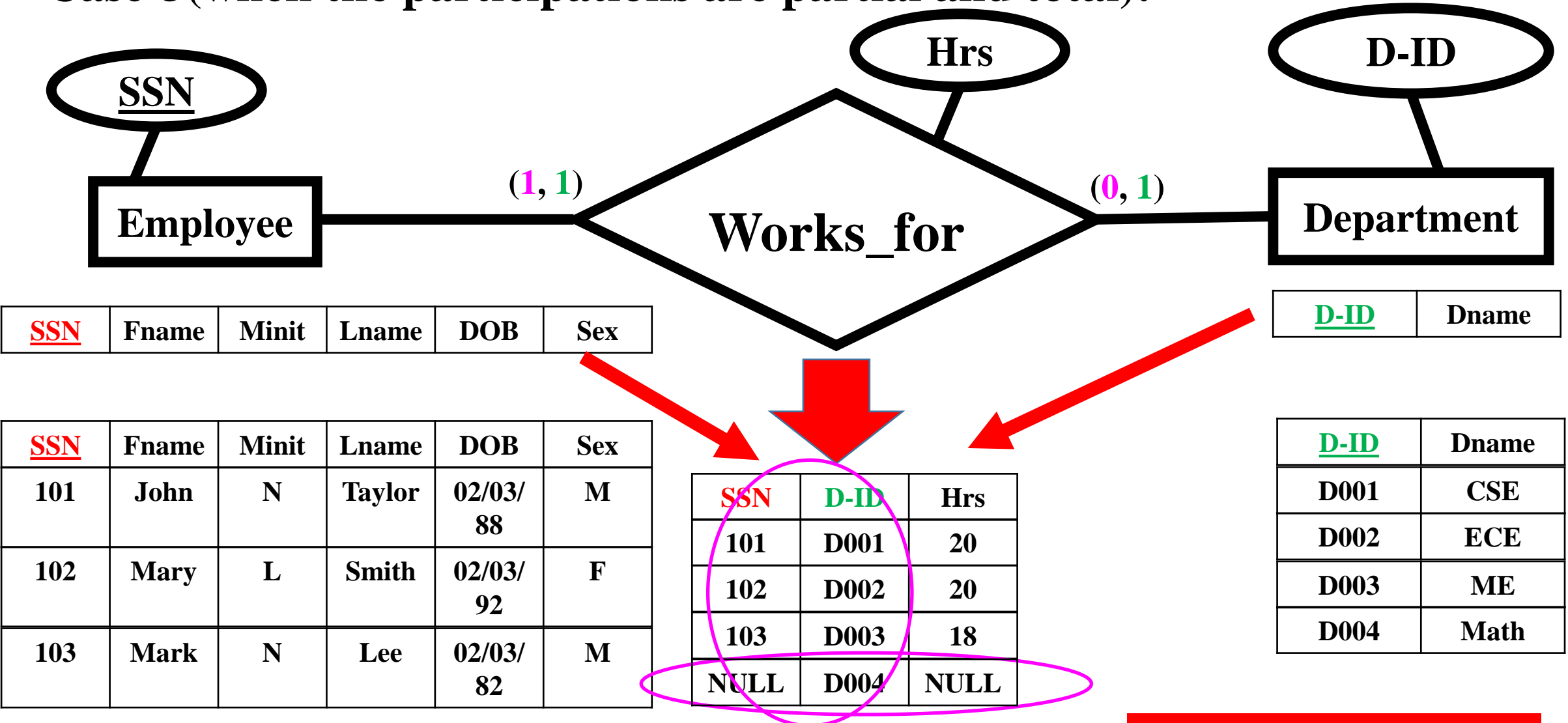
```
create table Employee_works_for_Department
(
    ssn varchar2(10),
    Fname varchar2(30) not null,
    Minit varchar2(4),
    Lname varchar2(30),
    DOB date,
    sex varchar2(1),
    Hrs number(5),
    D-ID varchar2(10),
    Dname varchar2(20),
    constraint pk_employee_work_department primary key( ssn )
)
```

- **Step-3: Mapping of Binary 1:1 Relationship Types.**

- **Case-3(when the participations are partial and total):**

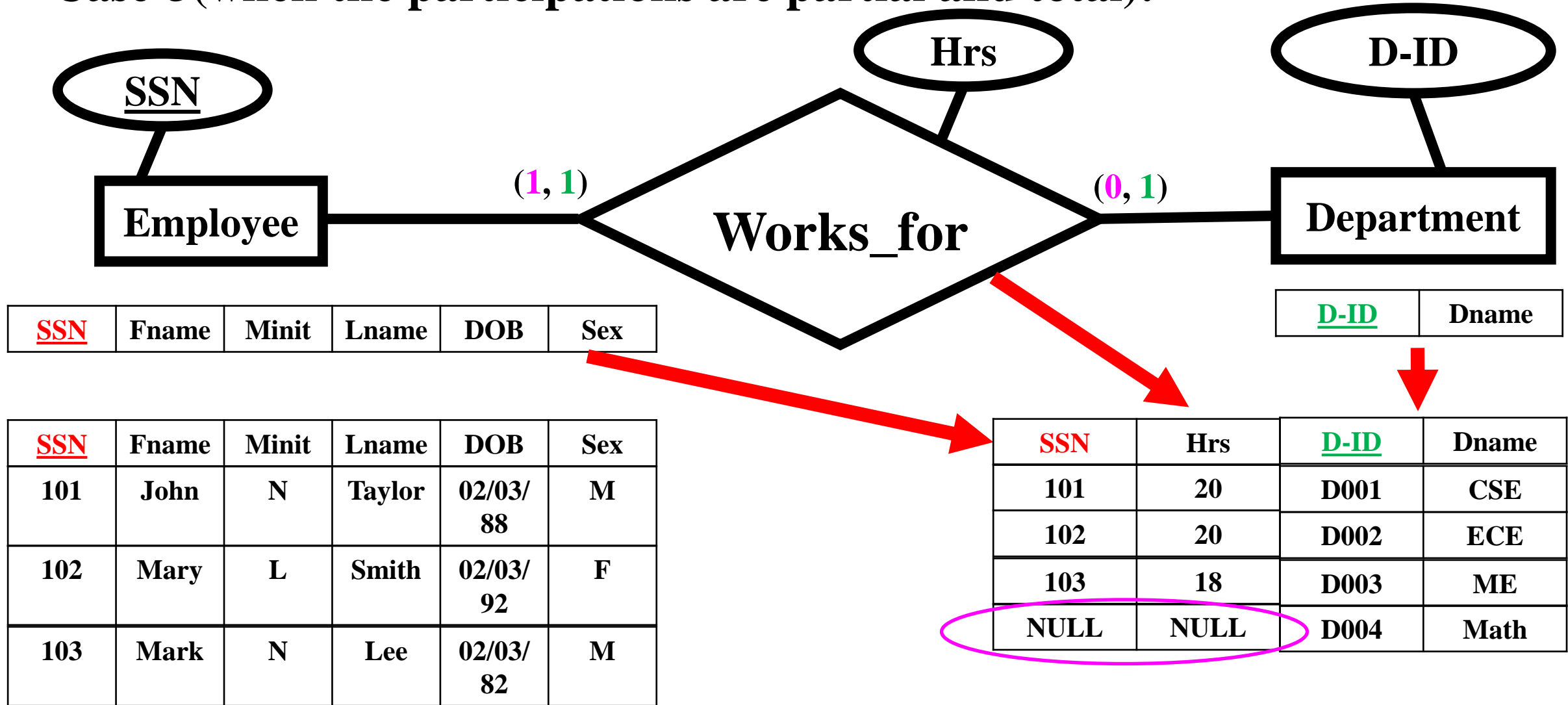
- Identify the entity type 'E' that is totally participating in the relationship type 'R'.
- Also, identify the entity type 'D' that is partially participating in the relationship type 'R'.
- Include the primary key of the partially participating entity type 'D' in 'E' as its foreign key.
- Also, include the attribute of 'R' in 'E'.

- **Step-3: Mapping of Binary 1:1 Relationship Types.**
- **Case-3**(when the participations are partial and total):



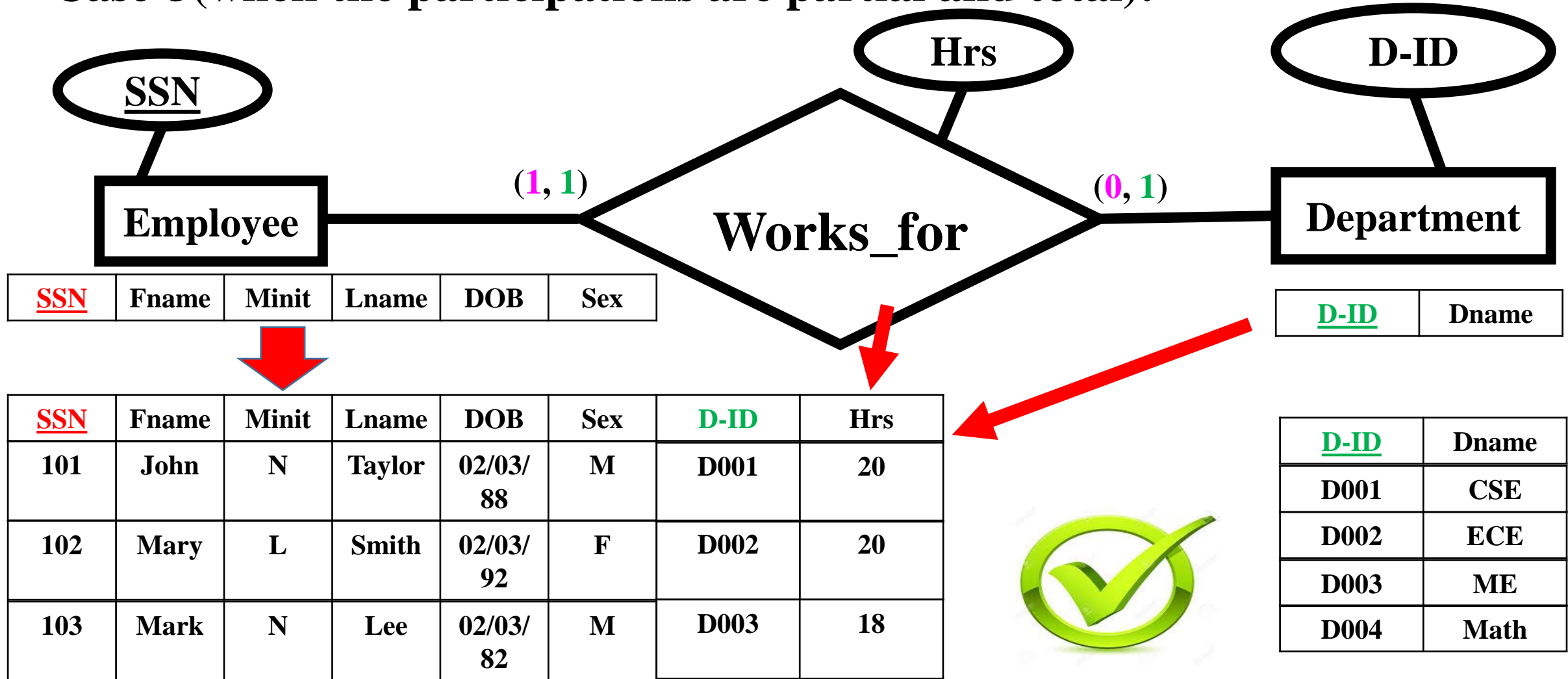
• **Space Wastage**

- **Step-3: Mapping of Binary 1:1 Relationship Types.**
- **Case-3 (when the participations are partial and total):**



- **Space Wastage w.r.t deletion in Employee**

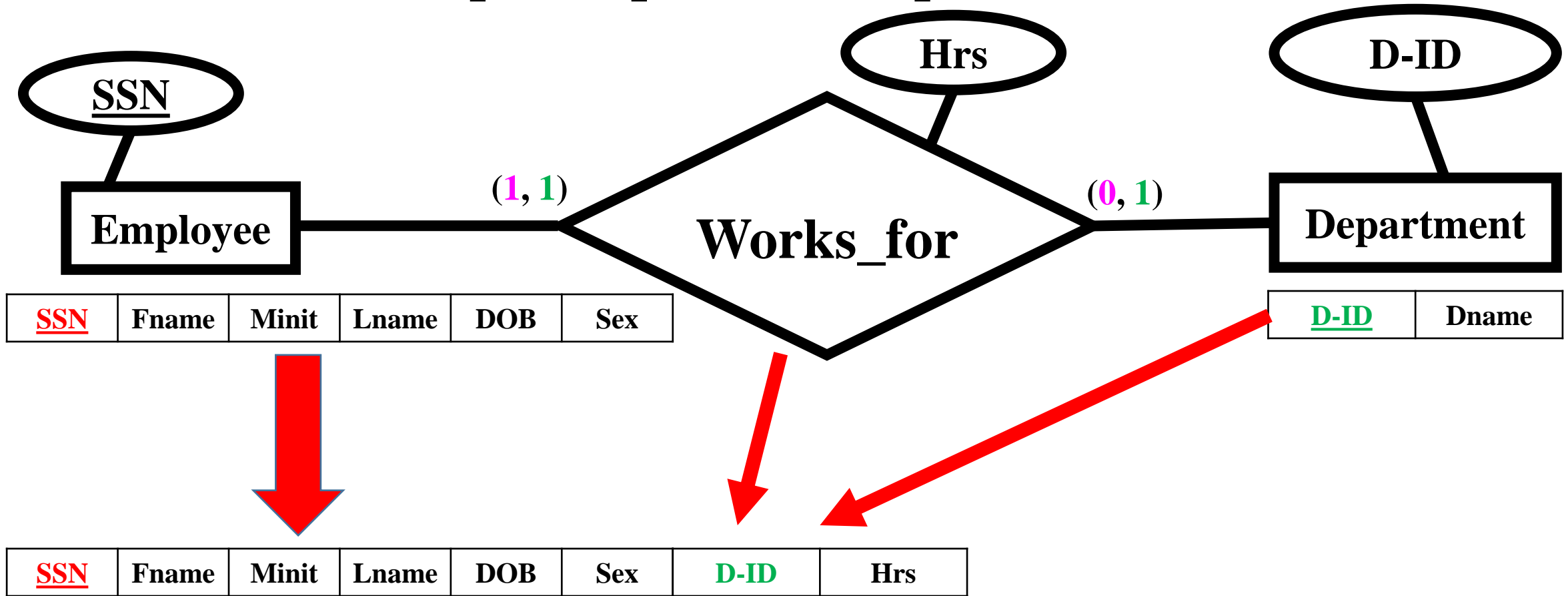
- **Step-3: Mapping of Binary 1:1 Relationship Types.**
- **Case-3 (when the participations are partial and total):**



- **No Space Wastage w.r.t deletion in Employee and Department**

- **Step-3: Mapping of Binary 1:1 Relationship Types.**

- **Case-3 (when the participations are partial and total):**





- **Step-3: Mapping of Binary 1:1 Relationship Types.**
- **Case-3 (when the participations are partial and total):**

**Employee Works\_for**

<u>SSN</u>	Fname	Minit	Lname	DOB	Sex	D-ID	Hrs
------------	-------	-------	-------	-----	-----	------	-----

**Department**

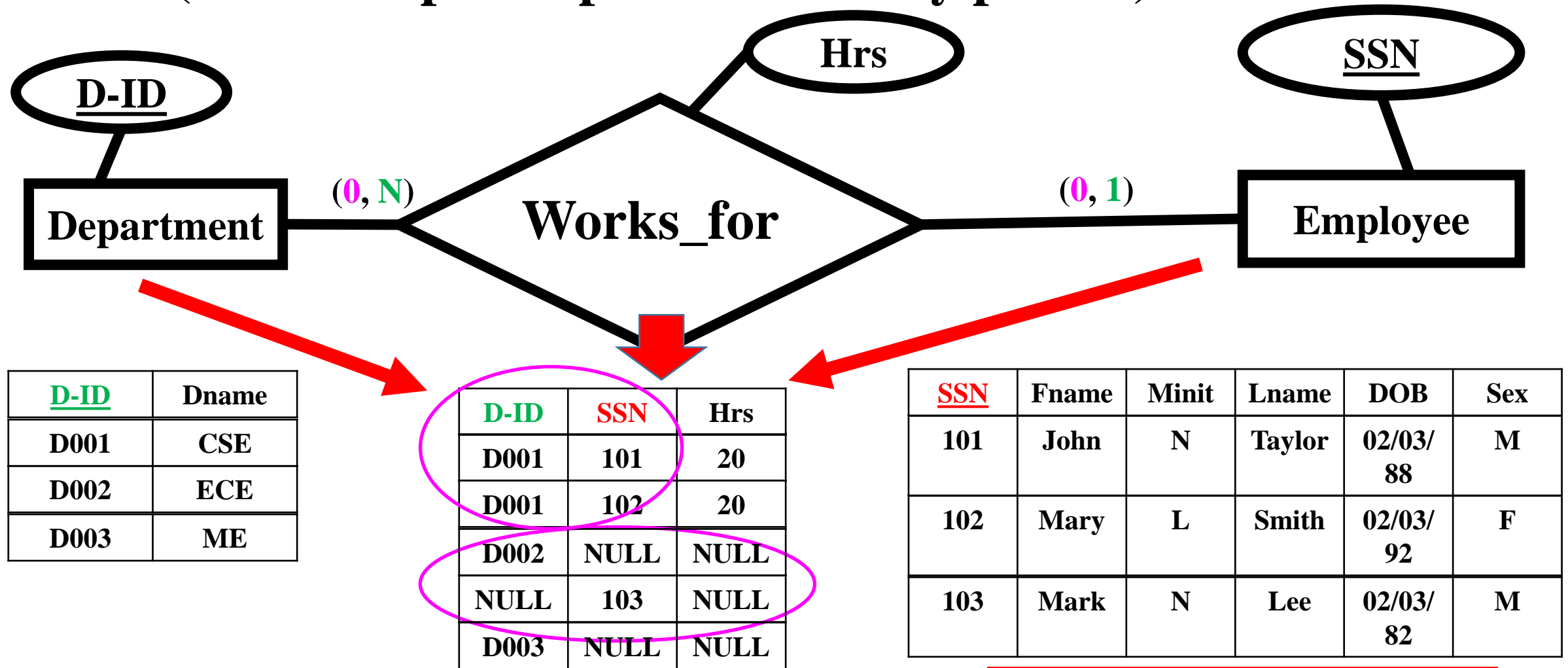
<u>D-ID</u>	Dname
-------------	-------

```
create table Employee_Works
(
    ssn varchar2(10),
    Fname varchar2(30) not null,
    Minit varchar2(4),
    Lname varchar2(30),
    DOB date,
    sex varchar2(1),
    D-ID varchar2(10) not null unique,
    Hrs number(5),
    constraint pk_employee primary key( ssn ),
    constraint fk_employee foreign key( D-ID ) references department(D-ID) on delete cascade
)
```

- **Step-4: Mapping of Binary 1:N Relationship Types.**

- **Step-4: Mapping of Binary 1:N Relationship Types.**

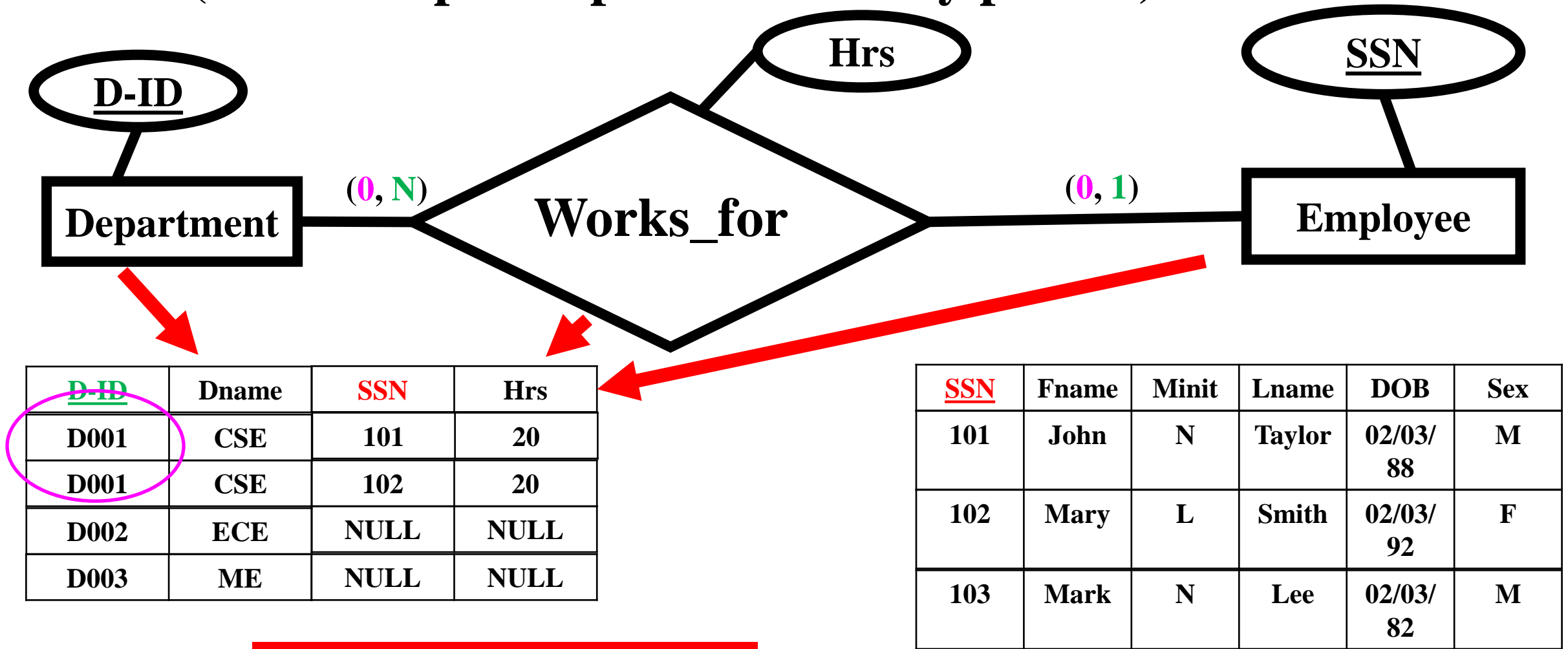
- **Case-1 (when the participations are only partial):**



- **Primary Key Problem**
- **Space Wastage**

- **Step-4: Mapping of Binary 1:N Relationship Types.**

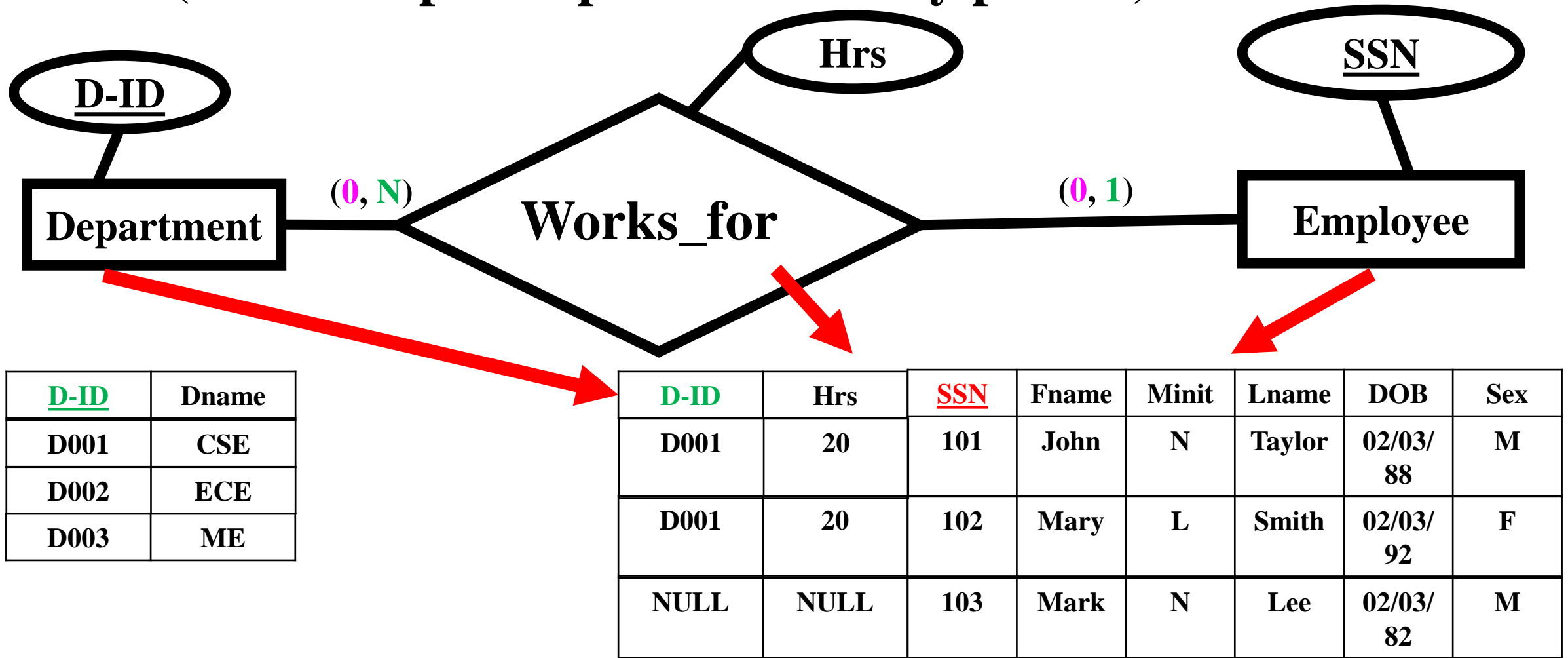
- **Case-1 (when the participations are only partial):**



- **Primary Key Problem**

- **Step-4: Mapping of Binary 1:N Relationship Types.**

- **Case-1 (when the participations are only partial):**

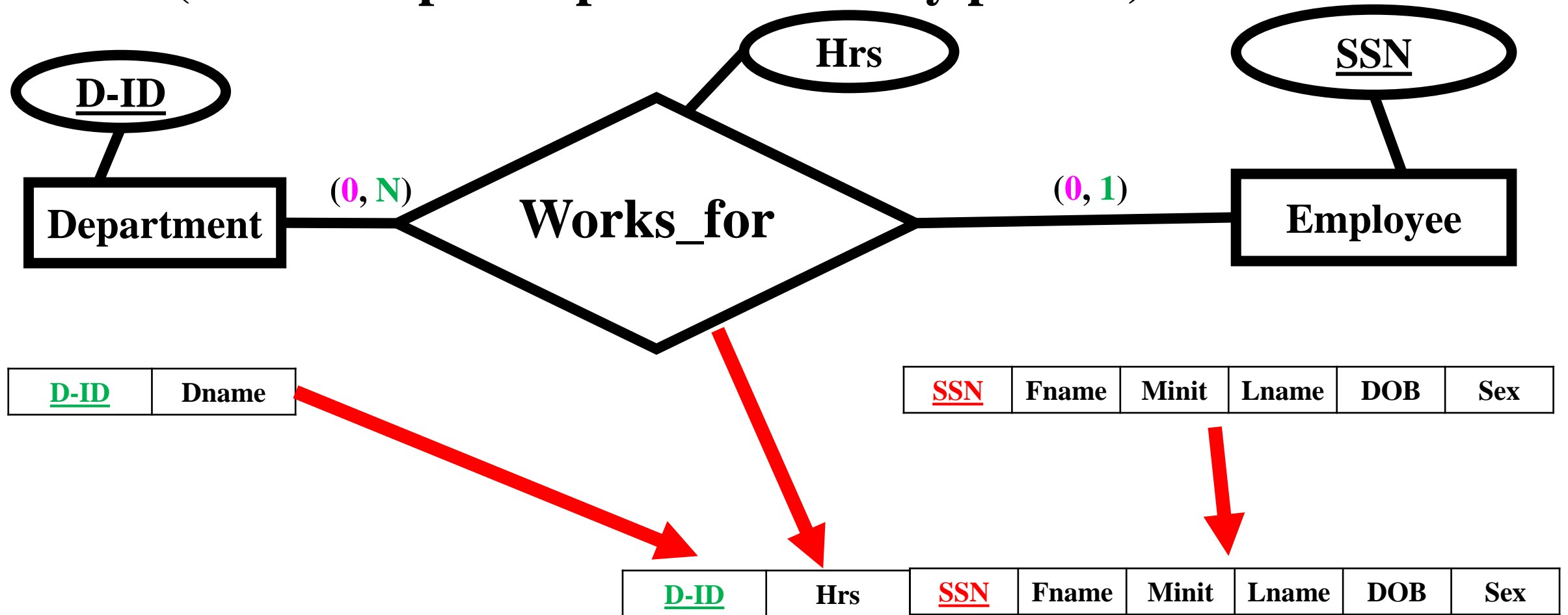


- **Less Space Wastage when there is ample participation**



- **Step-4: Mapping of Binary 1:N Relationship Types.**

- **Case-1 (when the participations are only partial):**



- **Step-4: Mapping of Binary 1:N Relationship Types.**

- **Case-1 (when the participations are only partial):**

### Department

<u>D-ID</u>	Dname
-------------	-------

### Employee Works\_for

<u>D-ID</u>	Hrs	<u>SSN</u>	Fname	Minit	Lname	DOB	Sex
-------------	-----	------------	-------	-------	-------	-----	-----

```
create table Employee_Works
```

```
(
```

```
  D-ID varchar2(10),
```

```
  ssn varchar2(10),
```

```
  Fname varchar2(30) not null,
```

```
  Minit varchar2(4),
```

```
  Lname varchar2(30),
```

```
  DOB date,
```

```
  sex varchar2(1),
```

```
  Hrs number(5),
```

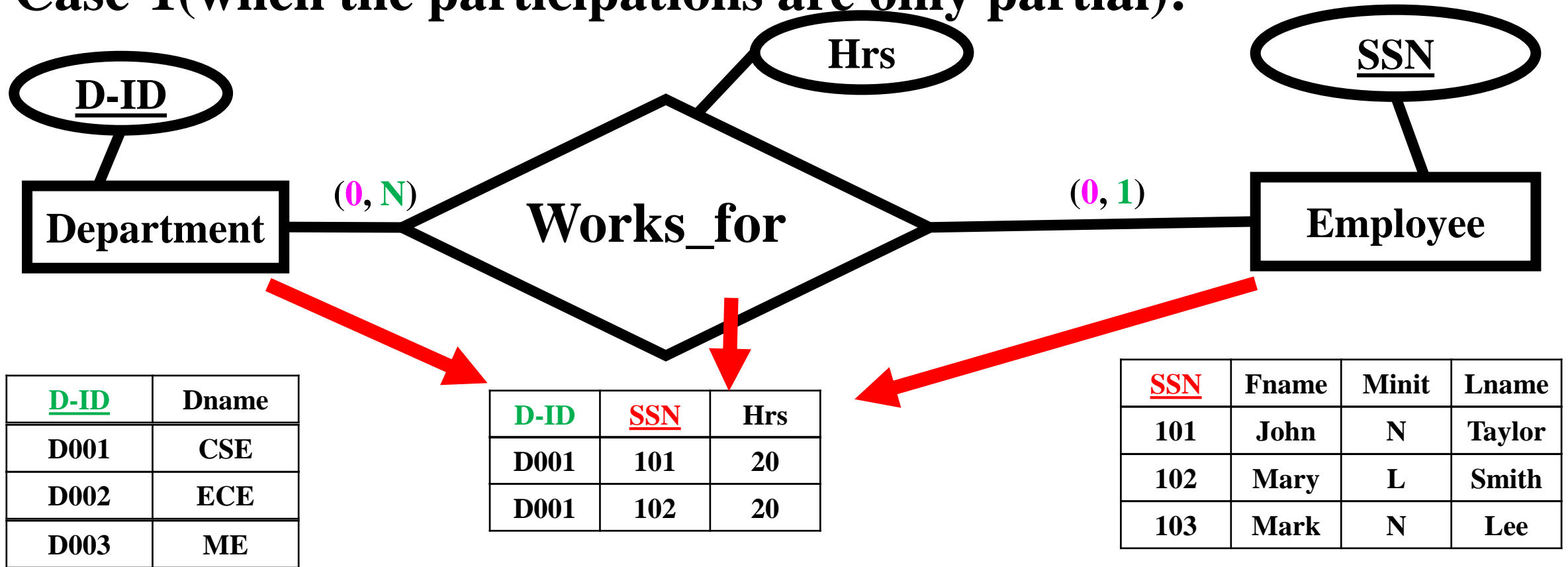
```
  constraint pk_employee primary key( ssn ),
```

```
  constraint fk_employee foreign key( D-ID ) references department(D-ID) on delete set NULL
```

```
)
```

- **Step-4: Mapping of Binary 1:N Relationship Types.**

- **Case-1 (when the participations are only partial):**



- **Less Space Wastage w.r.t deletion in Department**





- **Step-4: Mapping of Binary 1:N Relationship Types.**

- **Case-1 (when the participations are only partial):**

### Department

<u>D-ID</u>	Dname
-------------	-------

### Works\_for

<u>D-ID</u>	Hrs	<u>SSN</u>
-------------	-----	------------

### Employee

<u>SSN</u>	Fname	Minit	Lname	DOB	Sex
------------	-------	-------	-------	-----	-----

```
create table Works_for
(
    D-ID varchar2(10),
    ssn varchar2(10),
    Hrs number(5),
    constraint pk_works_for primary key( ssn ),
    constraint fk_works_for foreign key( D-ID ) references department(D-ID) on delete set cascade,
    constraint fk_works_for foreign key( ssn ) references employee(ssn) on delete set cascade
)
```

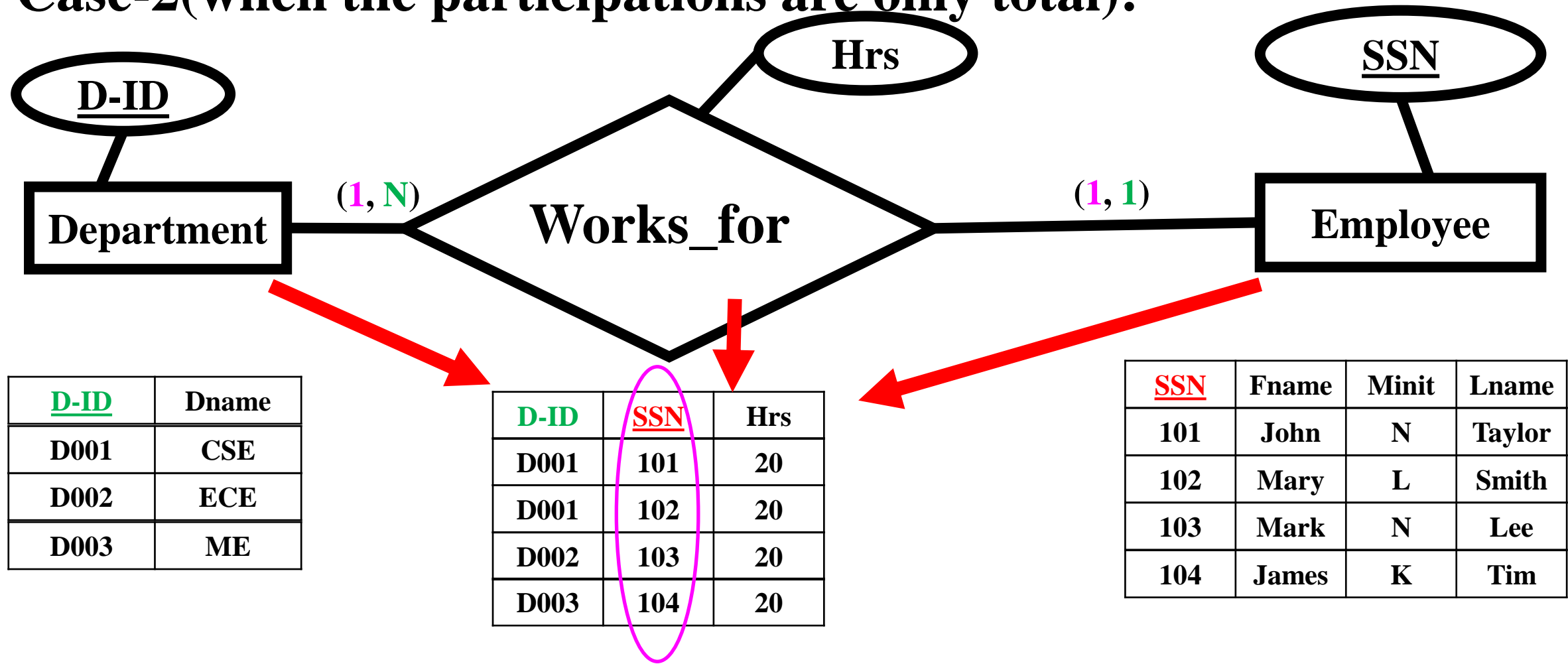
- **Step-4: Mapping of Binary 1:N Relationship Types.**

- **Case-2(when the participations are only total):**

- Identify the entity type on the N-side of the relationship as 'E'.
- Also, identify the entity type on the 1-side of the relationship as 'D'.
- Include the primary key of the of 'D' in 'E' as its foreign key.
- Also, include the attributes of 'R' in 'E'.

- **Step-4: Mapping of Binary 1:N Relationship Types.**

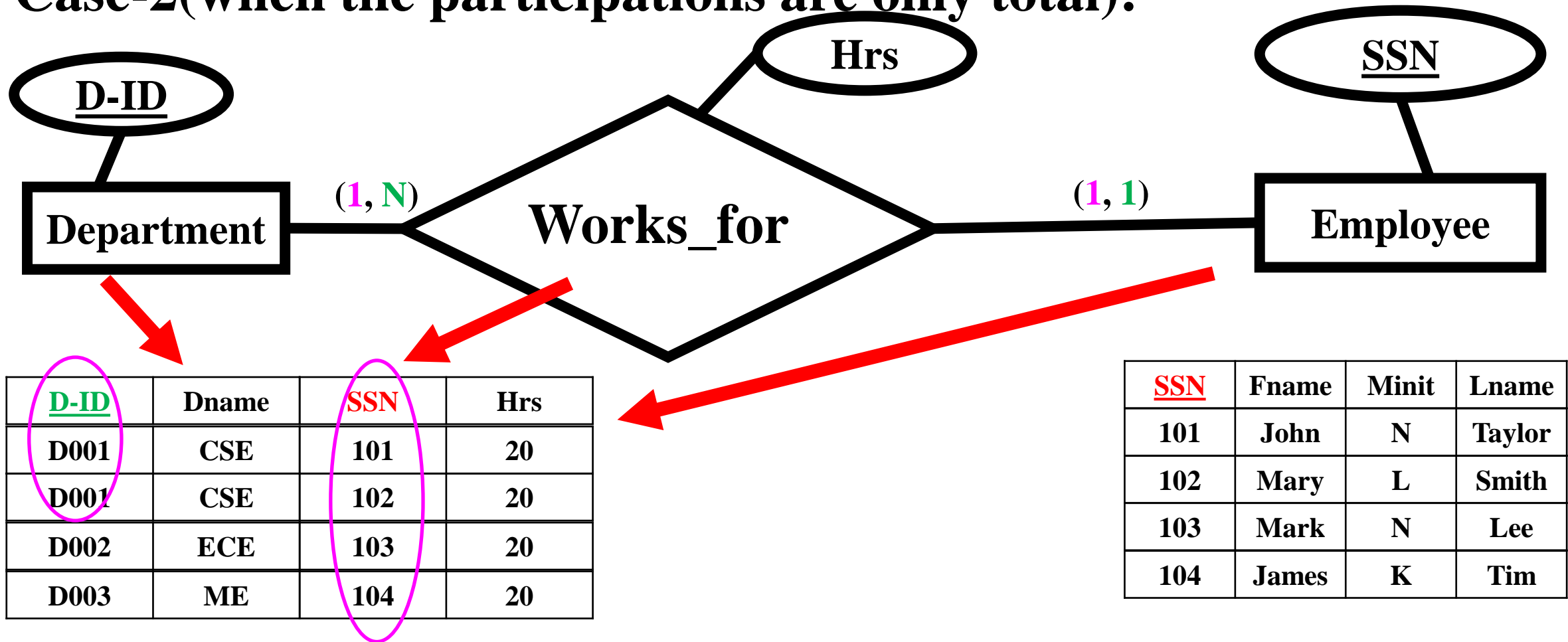
- **Case-2 (when the participations are only total):**



- **Space Wastage**

- **Step-4: Mapping of Binary 1:N Relationship Types.**

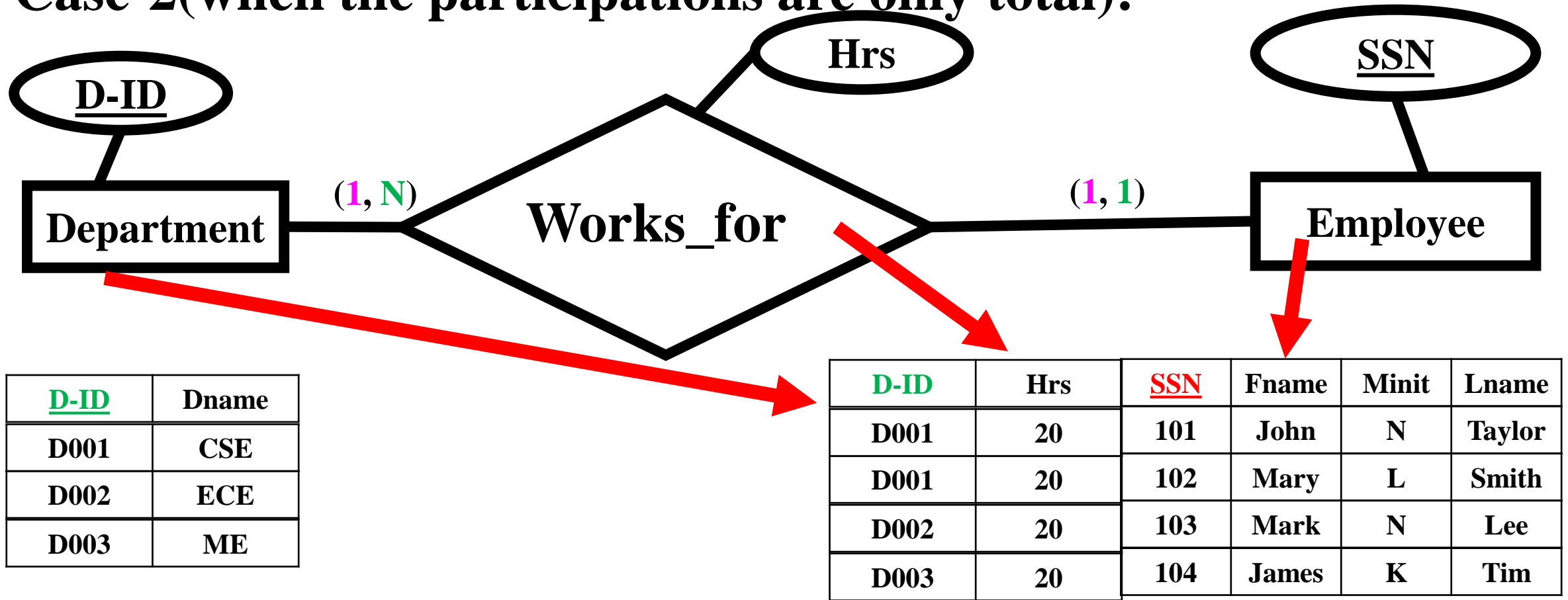
- **Case-2 (when the participations are only total):**



- **Primary Key problem**
- **Space wastage**

- **Step-4: Mapping of Binary 1:N Relationship Types.**

- **Case-2 (when the participations are only total):**

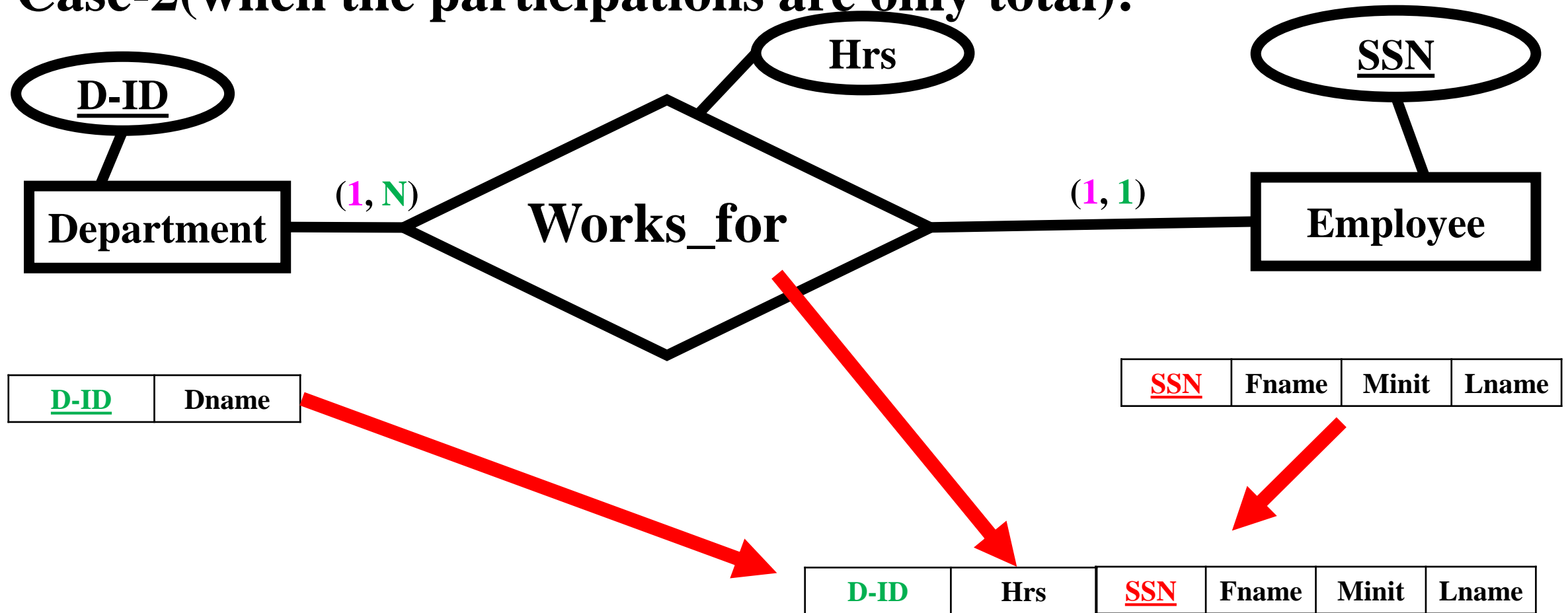


- **No Space Wastage**



- **Step-4: Mapping of Binary 1:N Relationship Types.**

- **Case-2 (when the participations are only total):**



- **Step-4: Mapping of Binary 1:N Relationship Types.**

- **Case-2(when the participations are only total):**

### Department

<u>D-ID</u>	Dname
-------------	-------

### Employee

### Works\_for

<u>D-ID</u>	Hrs	<u>SSN</u>	Fname	Minit	Lname	DOB	Sex
-------------	-----	------------	-------	-------	-------	-----	-----

```
create table Employee_Work
```

```
(
```

```
    D-ID varchar2(10),
```

```
    ssn varchar2(10),
```

```
    Fname varchar2(30) not null,
```

```
    Minit varchar2(4),
```

```
    Lname varchar2(30),
```

```
    DOB date,
```

```
    sex varchar2(1),
```

```
    Hrs number(5),
```

```
    constraint pk_employee primary key( ssn ),
```

```
    constraint fk_employee foreign key( D-ID ) references department(D-ID) on delete cascade
```

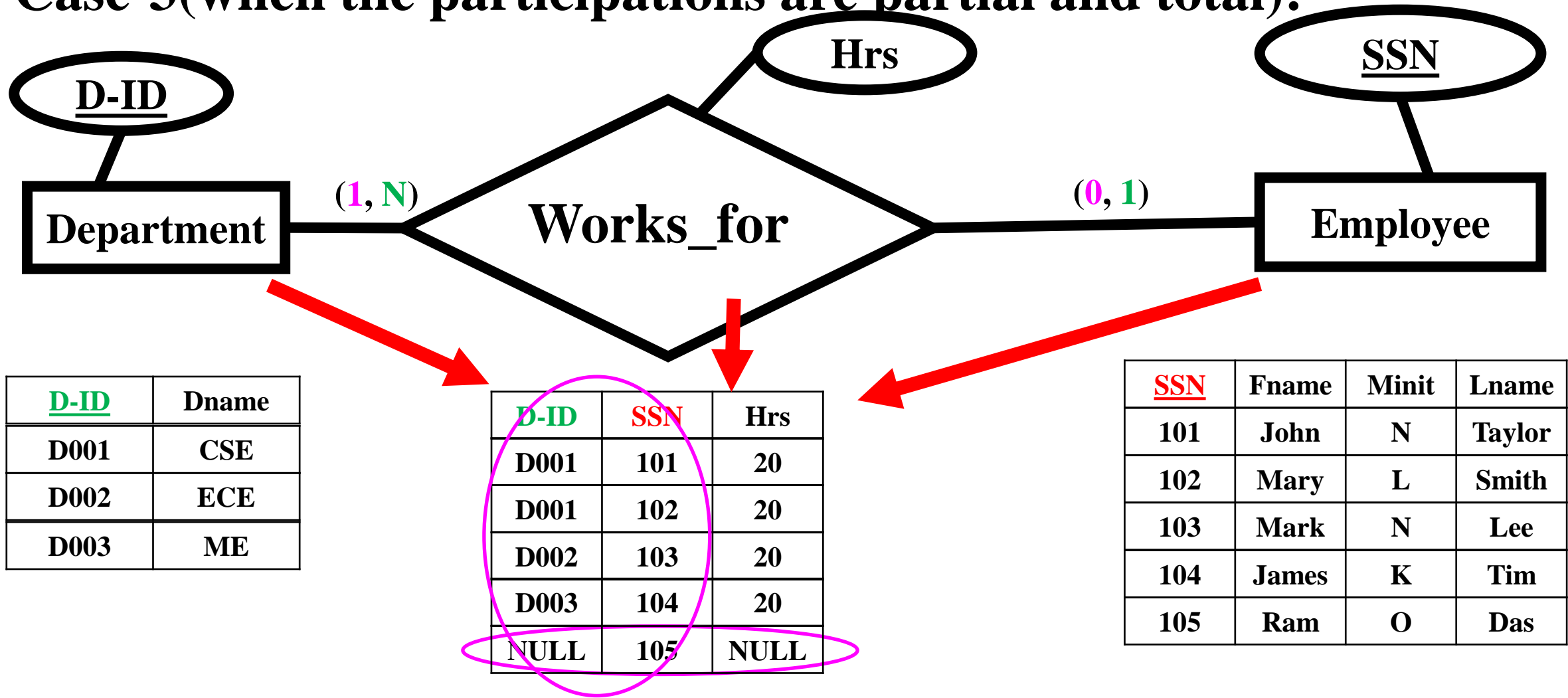
```
)
```

- **Step-4: Mapping of Binary 1:N Relationship Types.**
- **Case-3(when the participations are partial and total):**
  - Identify the entity type on the N-side of the relationship as 'E'.
  - Also, identify the entity type on the 1-side of the relationship as 'D'.
  - Include the primary key of the of 'D' in 'E' as its foreign key.
  - Also, include the attributes of 'R' in 'E'.



- **Step-4: Mapping of Binary 1:N Relationship Types.**

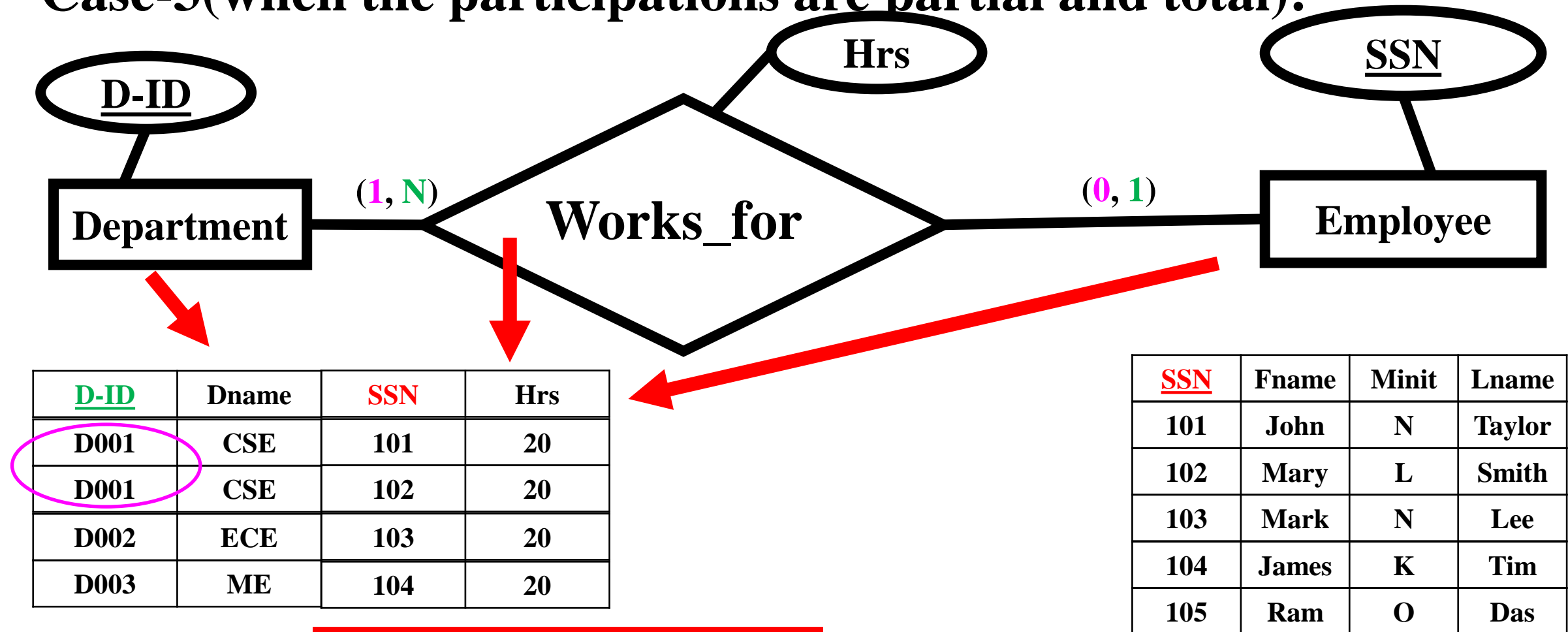
- **Case-3 (when the participations are partial and total):**



- **Space Wastage**

- **Step-4: Mapping of Binary 1:N Relationship Types.**

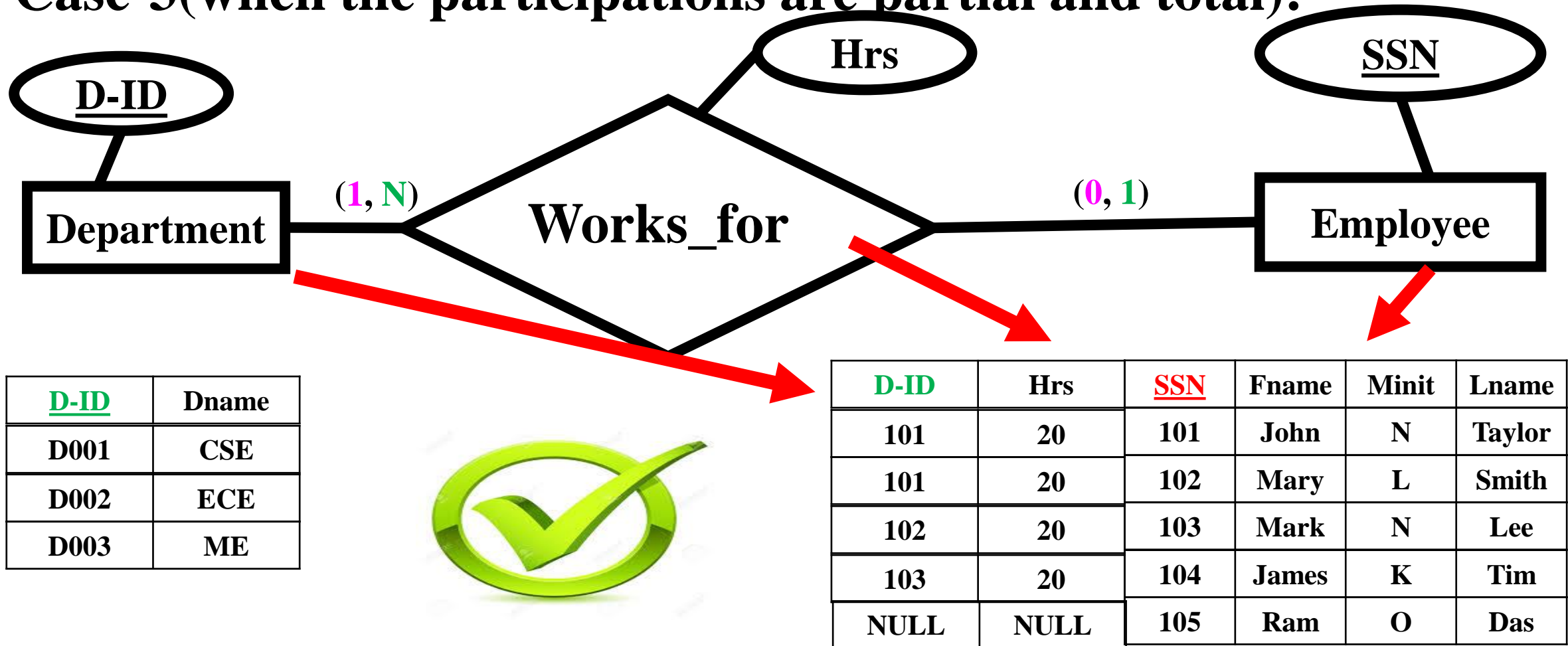
- **Case-3 (when the participations are partial and total):**



- **Primary Key Problem**

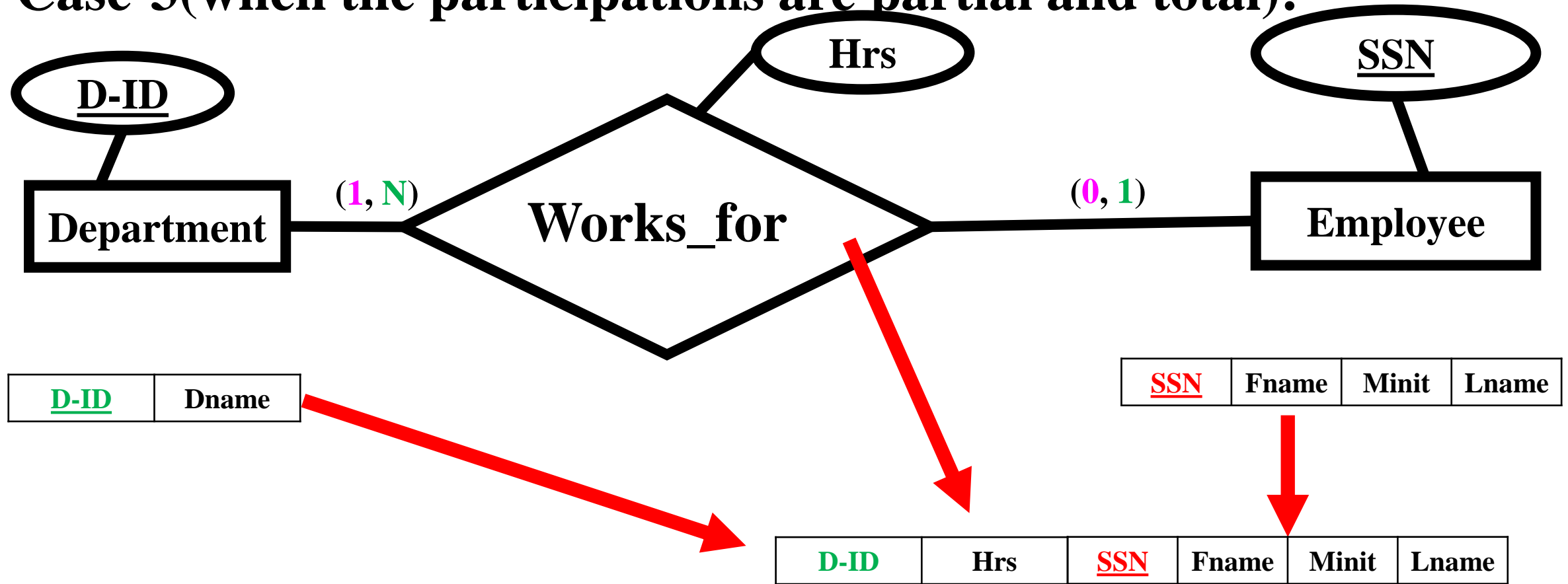
- **Step-4: Mapping of Binary 1:N Relationship Types.**

- **Case-3 (when the participations are partial and total):**



- **Less Space Wastage**

- **Step-4: Mapping of Binary 1:N Relationship Types.**
- **Case-3 (when the participations are partial and total):**



- **Step-4: Mapping of Binary 1:N Relationship Types.**
- **Case-3(when the participations are partial and total):**

### Department

<u>D-ID</u>	Dname
-------------	-------

### Employee Works\_for

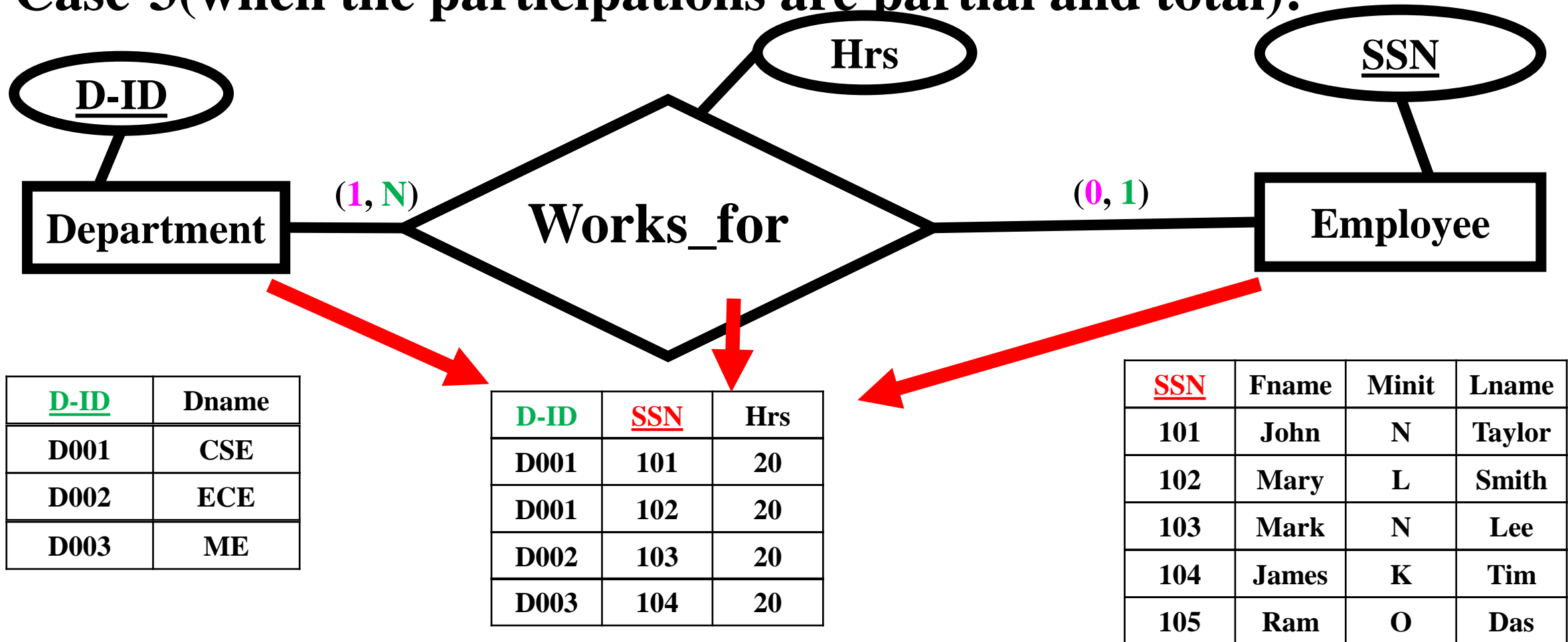
<u>D-ID</u>	Hrs	<u>SSN</u>	Fname	Minit	Lname	DOB	Sex
-------------	-----	------------	-------	-------	-------	-----	-----

**create table Employee\_Work**

```
(
  D-ID varchar2(10),
  ssn varchar2(10),
  Fname varchar2(30) not null,
  Minit varchar2(4),
  Lname varchar2(30),
  DOB date,
  sex varchar2(1),
  Hrs number(5),
  constraint pk_employee primary key( ssn ),
  constraint fk_employee foreign key( D-ID ) references department(D-ID) on delete set NULL
)
```

- **Step-4: Mapping of Binary 1:N Relationship Types.**

- **Case-3 (when the participations are partial and total):**



- **Less Space Wastage w.r.t deletion in Department**



- **Step-4: Mapping of Binary 1:N Relationship Types.**
- **Case-3(when the participations are partial and total):**

### Department

<u>D-ID</u>	Dname
-------------	-------

### Works\_for

D-ID	Hrs	<u>SSN</u>
------	-----	------------

### Employee

<u>SSN</u>	Fname	Minit	Lname	DOB	Sex
------------	-------	-------	-------	-----	-----

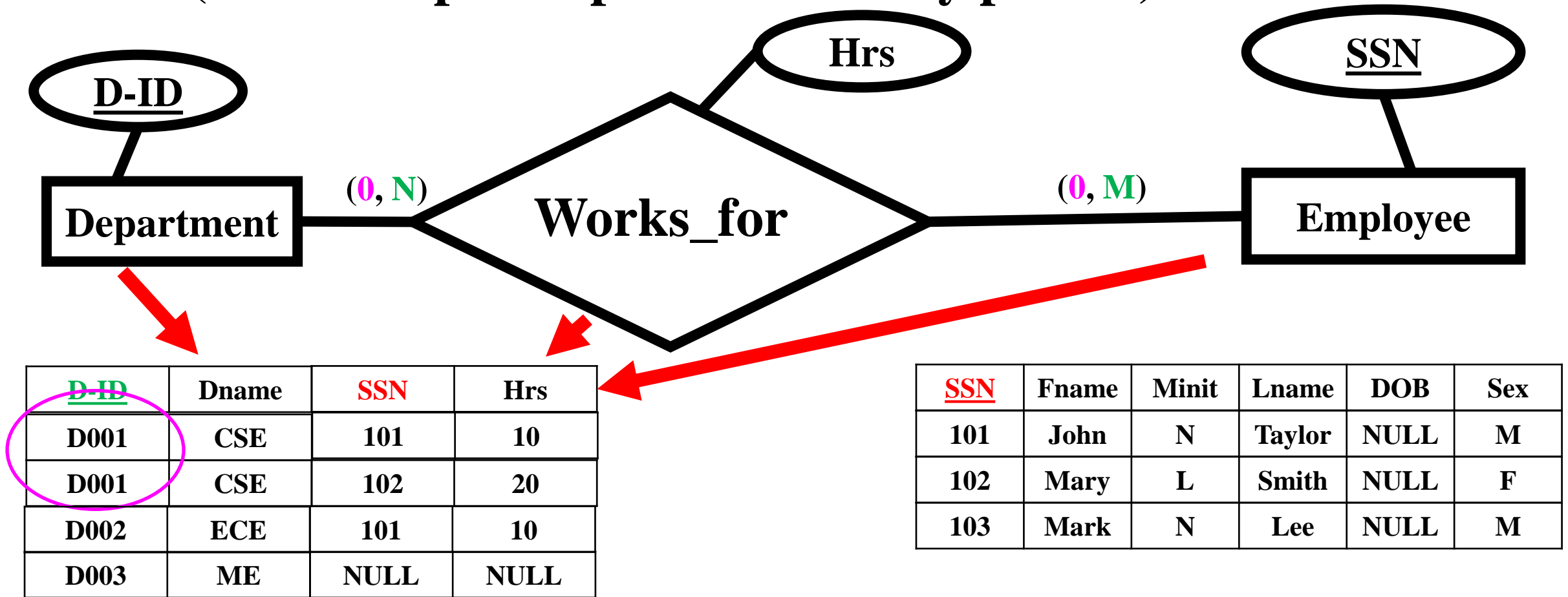
```
create table Works_for
(
    D-ID varchar2(10) not null,
    ssn varchar2(10),
    Hrs number(5),
    constraint pk_works_for primary key( ssn ),
    constraint fk_works_for foreign key( D-ID ) references department(D-ID) on delete set cascade,
    constraint fk_works_for foreign key( ssn ) references employee(ssn) on delete set cascade
)
```

- **Step-5: Mapping of Binary M:N Relationship Types.**



- **Step-5: Mapping of Binary M:N Relationship Types.**

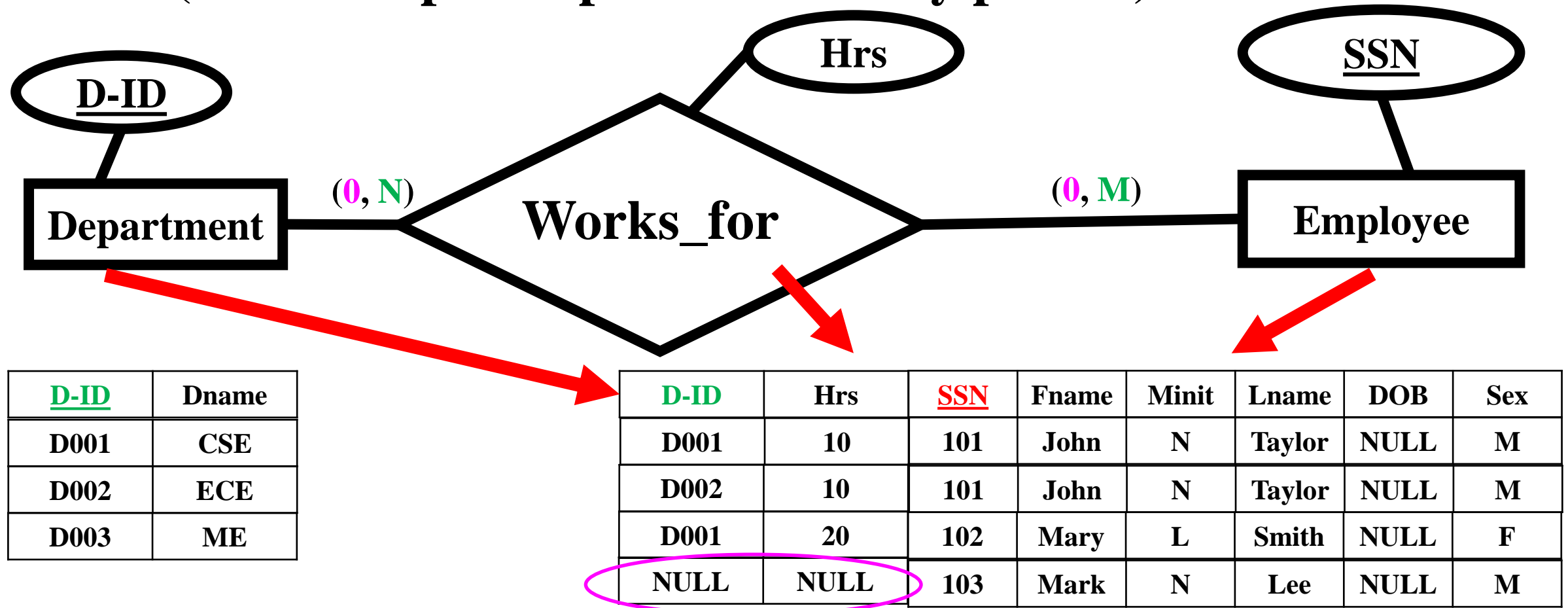
- **Case-1 (when the participations are only partial):**



- **Primary Key Problem**

- **Step-5: Mapping of Binary M:N Relationship Types.**

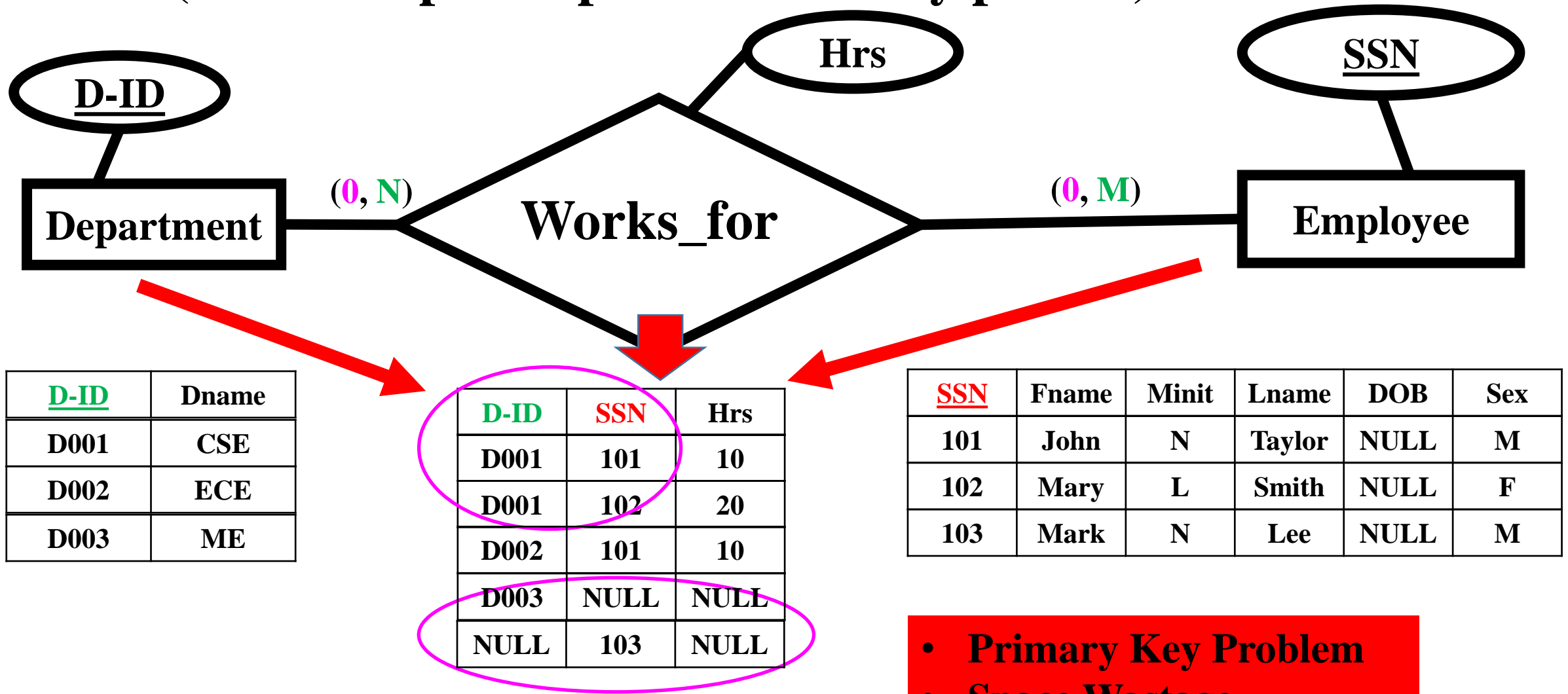
- **Case-1 (when the participations are only partial):**



- **Space Wastage when there is less participation of employees**

- **Step-5: Mapping of Binary M:N Relationship Types.**

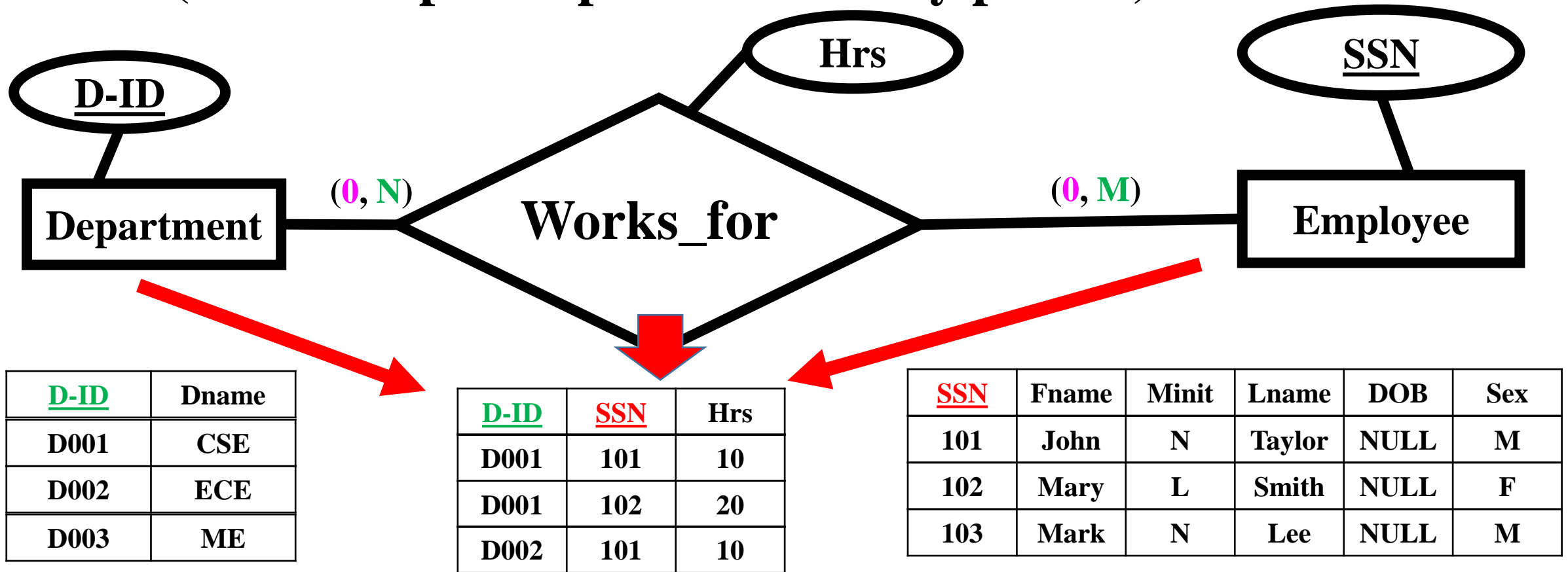
- **Case-1 (when the participations are only partial):**



- **Primary Key Problem**
- **Space Wastage**

- **Step-5: Mapping of Binary M:N Relationship Types.**

- **Case-1 (when the participations are only partial):**

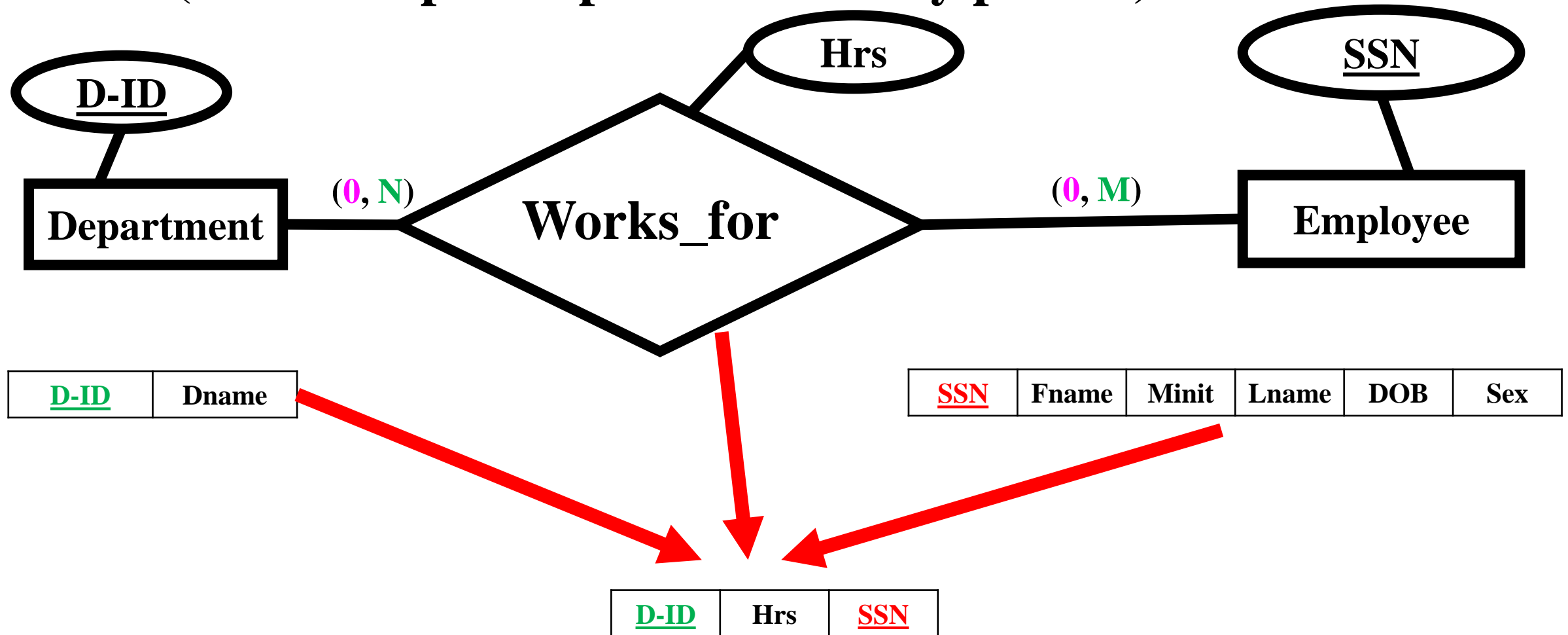


- **Less Space Wastage**



- **Step-5: Mapping of Binary M:N Relationship Types.**

- **Case-1 (when the participations are only partial):**



- **Step-5: Mapping of Binary M:N Relationship Types.**

- **Case-1 (when the participations are only partial):**

### Department

<u>D-ID</u>	Dname
-------------	-------

### Works\_for

<u>D-ID</u>	<u>SSN</u>	Hrs
-------------	------------	-----

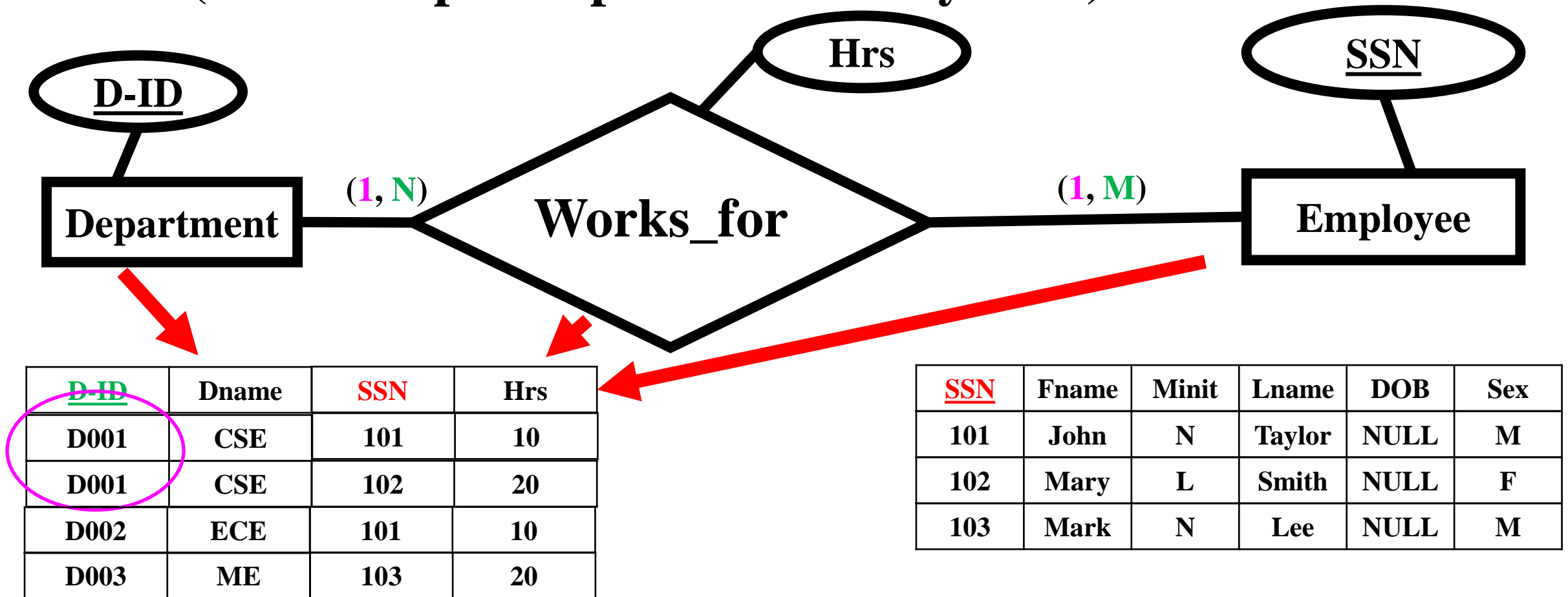
### Employee

<u>SSN</u>	Fname	Minit	Lname	DOB	Sex
------------	-------	-------	-------	-----	-----

```
create table Works_for
(
  D-ID varchar2(10),
  ssn varchar2(10),
  Hrs number(5),
  constraint pk_works_for primary key( D-ID, ssn ),
  constraint fk_works_for1 foreign key( ssn ) references employee(ssn) on delete set cascade,
  constraint fk_works_for2 foreign key( D-ID ) references department(D-ID) on delete set cascade
)
```

- **Step-5: Mapping of Binary M:N Relationship Types.**

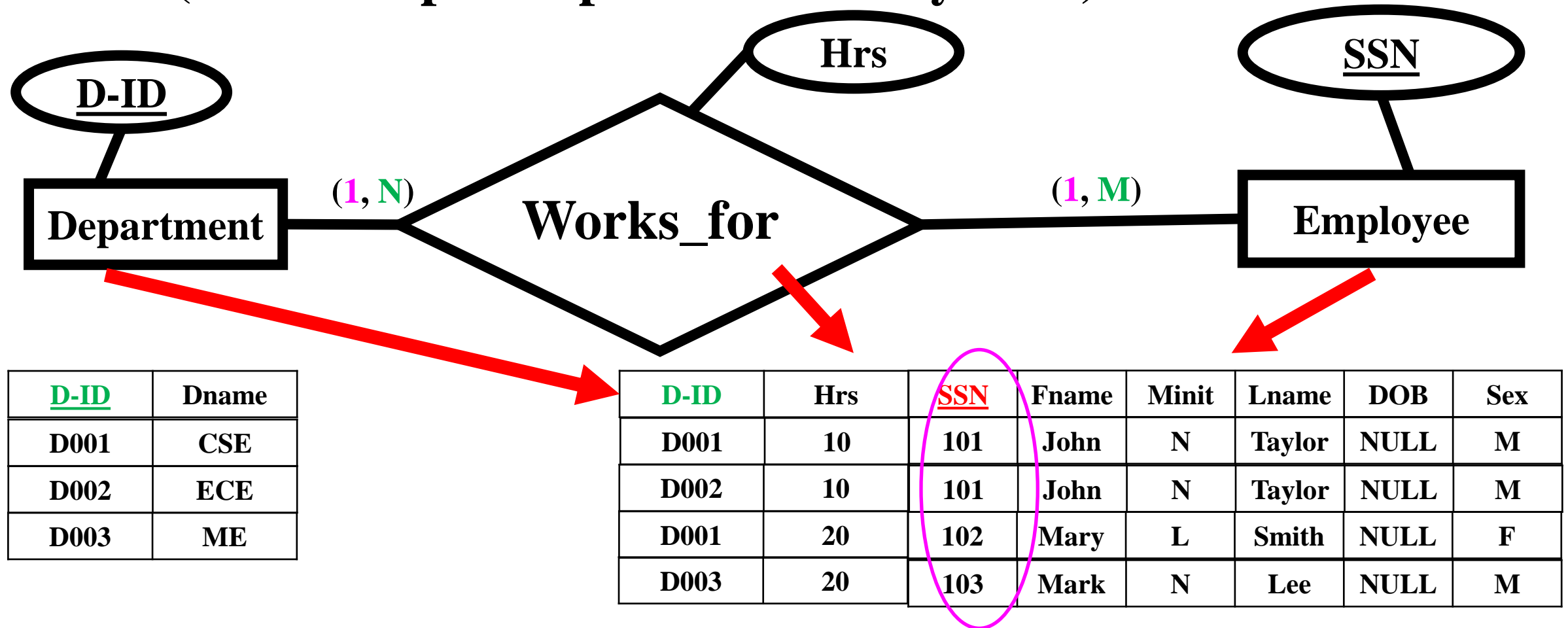
- **Case-2 (when the participations are only total):**



- **Primary Key Problem**

- **Step-5: Mapping of Binary M:N Relationship Types.**

- **Case-2 (when the participations are only total):**

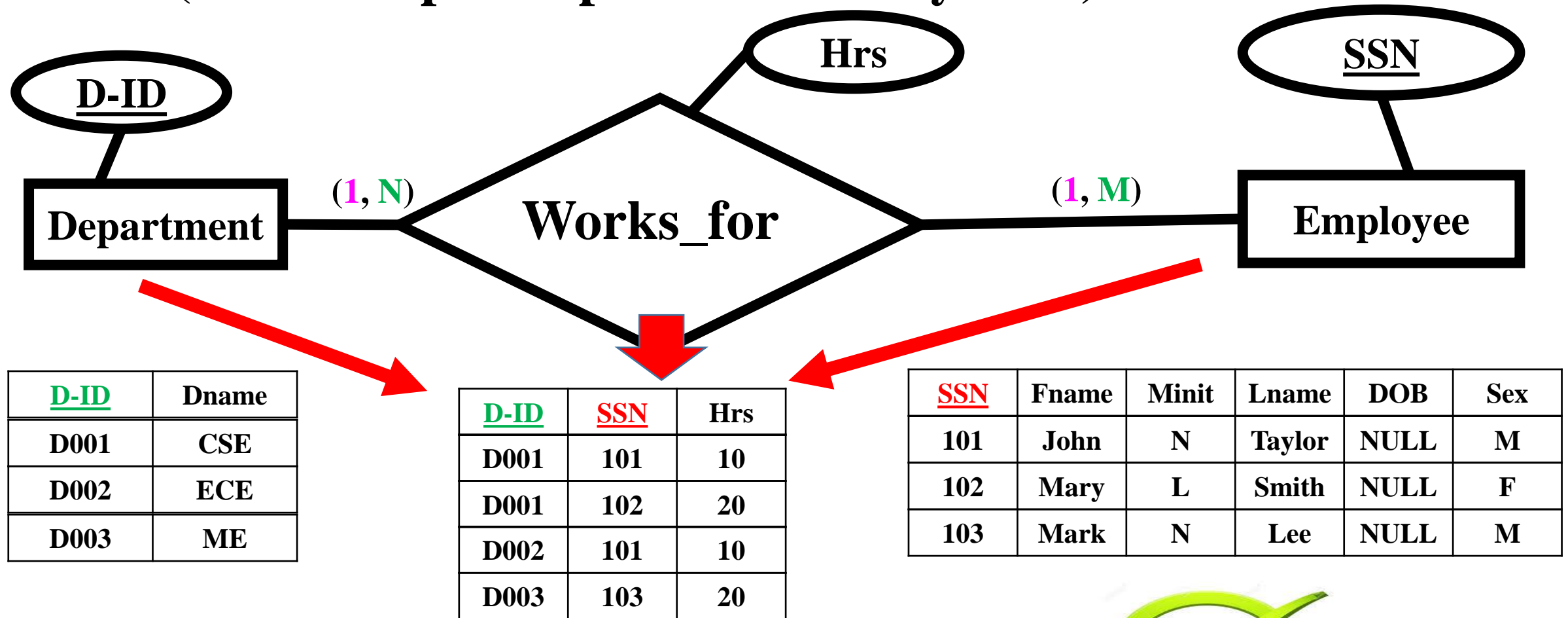


- **Primary Key Problem**



- **Step-5: Mapping of Binary M:N Relationship Types.**

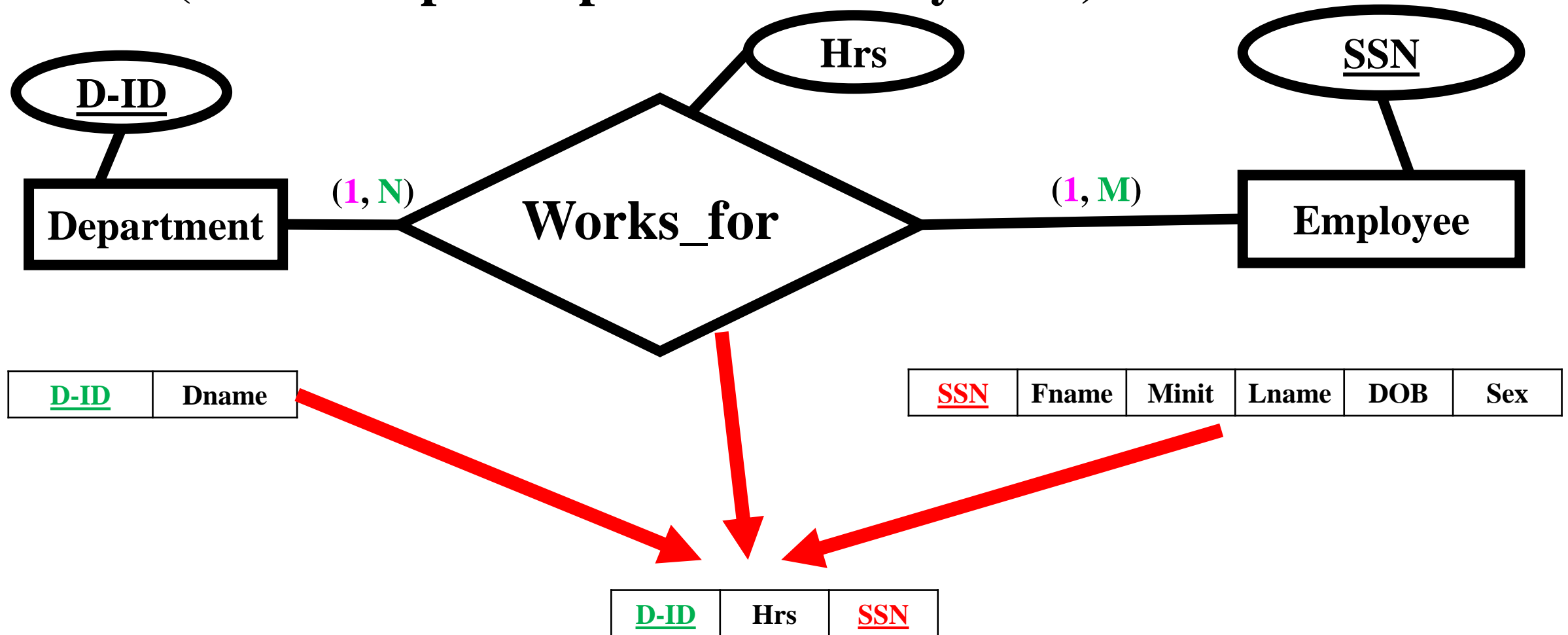
- **Case-2 (when the participations are only total):**



- **No Primary Key Problem**



- **Step-5: Mapping of Binary M:N Relationship Types.**
- **Case-2 (when the participations are only total):**



- **Step-5: Mapping of Binary M:N Relationship Types.**

- **Case-2(when the participations are only total):**

### Department

<u>D-ID</u>	Dname
-------------	-------

### Works\_for

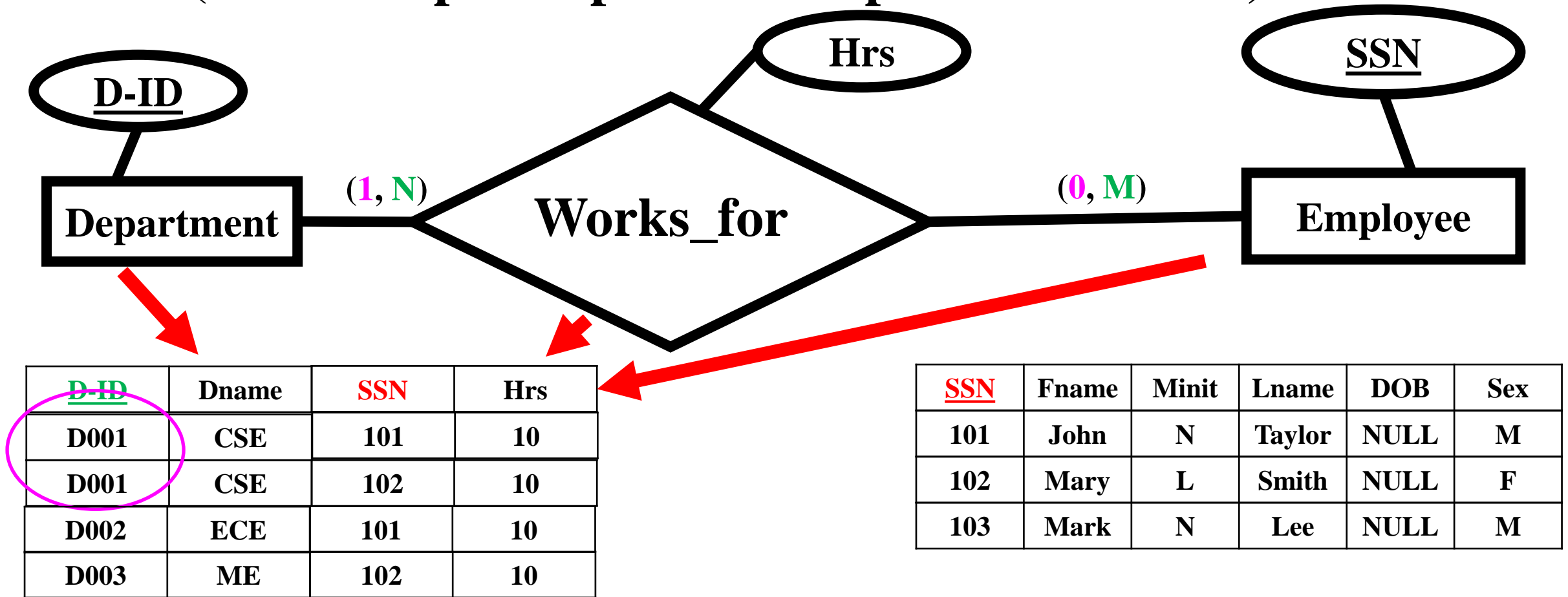
<u>D-ID</u>	<u>SSN</u>	Hrs
-------------	------------	-----

### Employee

<u>SSN</u>	Fname	Minit	Lname	DOB	Sex
------------	-------	-------	-------	-----	-----

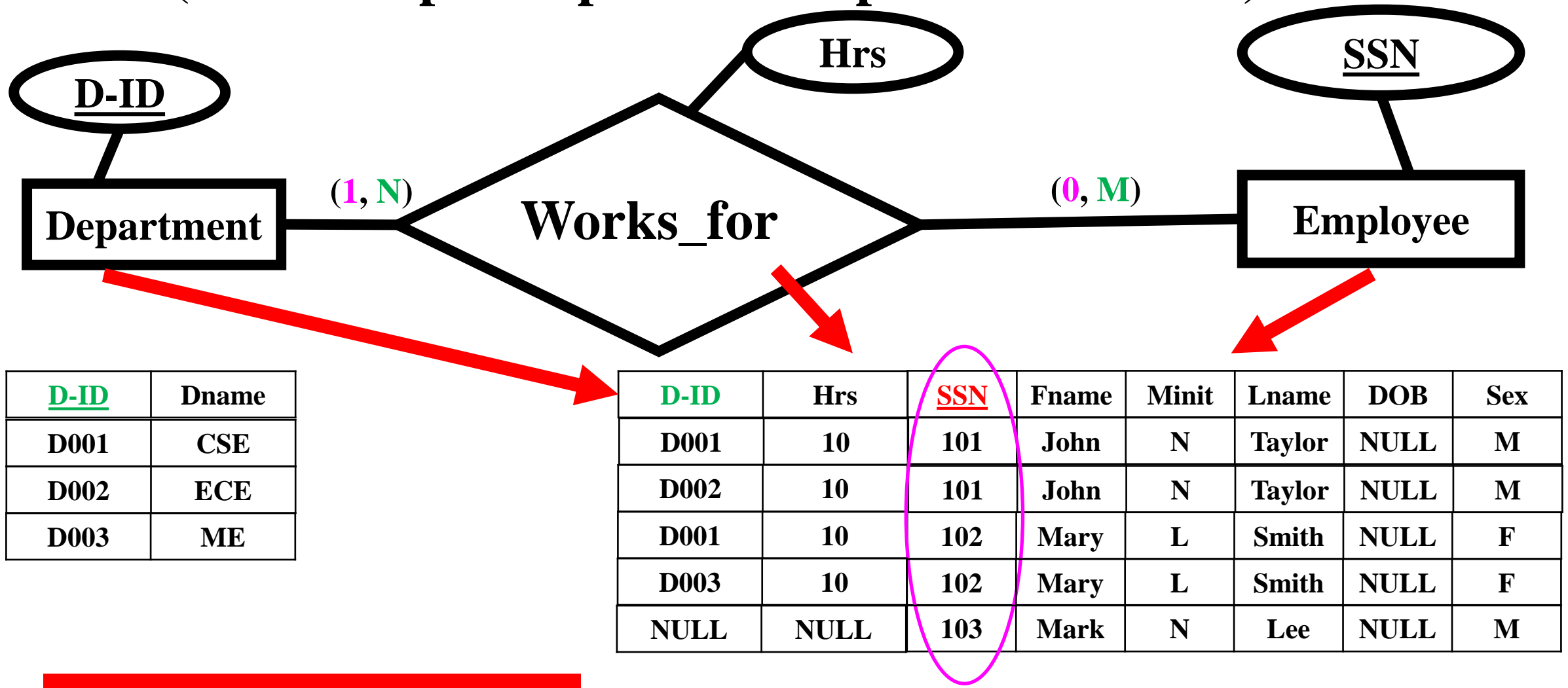
```
create table Works_for
(
  D-ID varchar2(10),
  ssn varchar2(10),
  Hrs number(5),
  constraint pk_works_for primary key( D-ID, ssn ),
  constraint fk_works_for1 foreign key( ssn ) references employee(ssn) on delete set cascade,
  constraint fk_works_for2 foreign key( D-ID ) references department(D-ID) on delete set cascade
)
```

- **Step-5: Mapping of Binary M:N Relationship Types.**
- **Case-3 (when the participations are partial and total):**



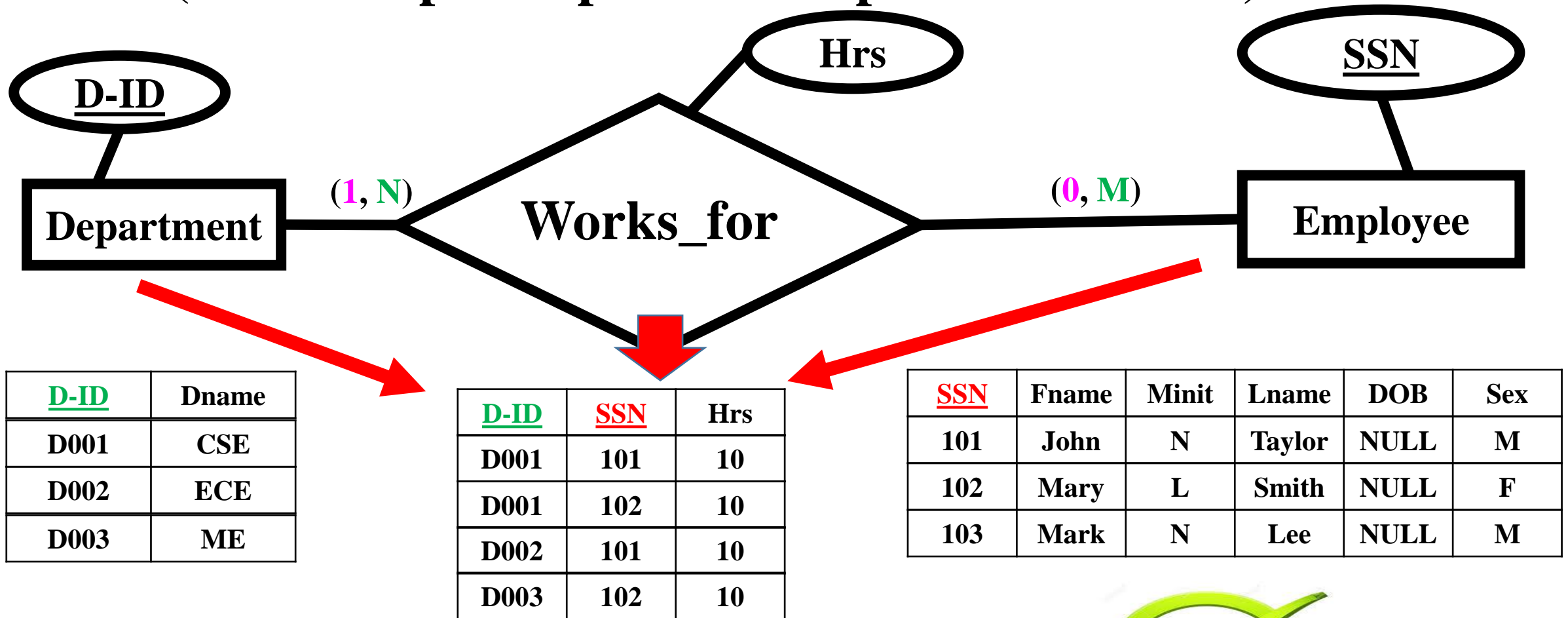
- **Primary Key Problem**

- **Step-5: Mapping of Binary M:N Relationship Types.**
- **Case-3 (when the participations are partial and total):**



- **Primary Key Problem**

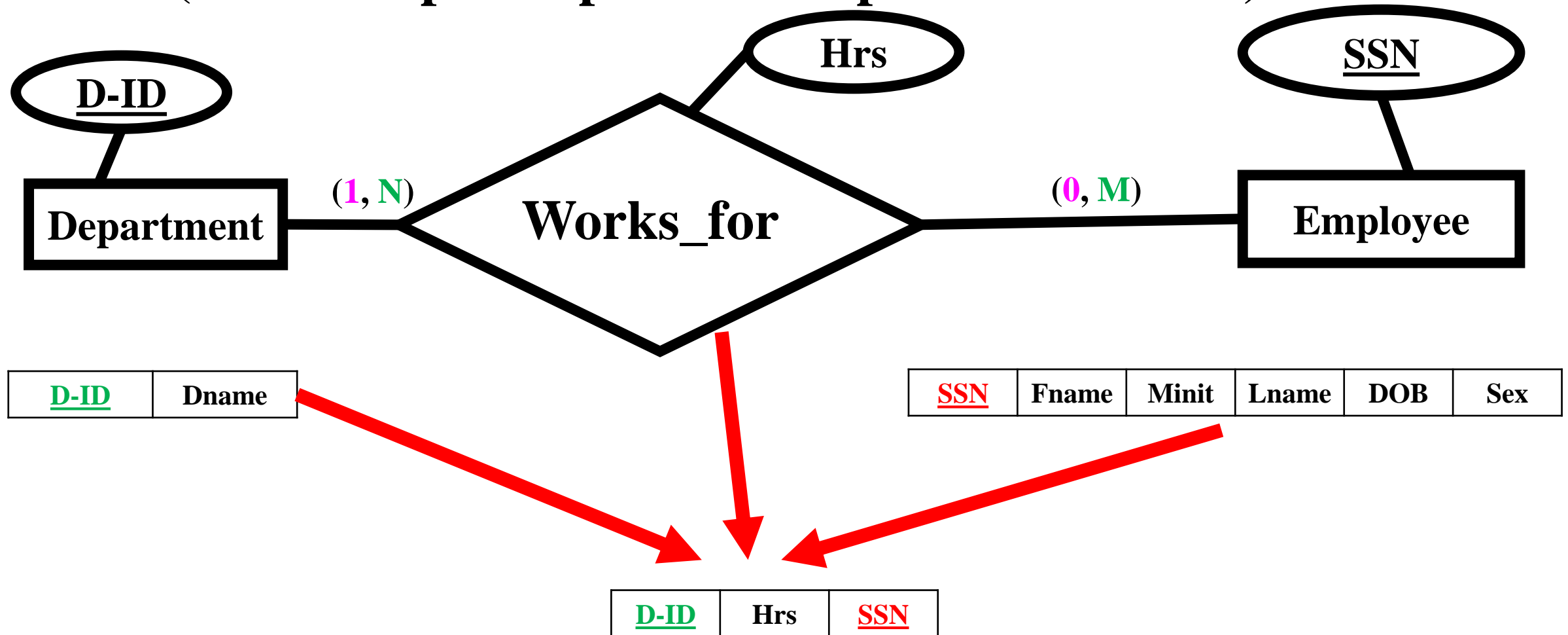
- **Step-5: Mapping of Binary M:N Relationship Types.**
- **Case-3 (when the participations are partial and total):**



- **No Primary Key Problem**



- **Step-5: Mapping of Binary M:N Relationship Types.**
- **Case-3 (when the participations are partial and total):**



- **Step-5: Mapping of Binary M:N Relationship Types.**
- **Case-3(when the participations are partial and total):**

### Department

<u>D-ID</u>	Dname
-------------	-------

### Works\_for

<u>D-ID</u>	<u>SSN</u>	Hrs
-------------	------------	-----

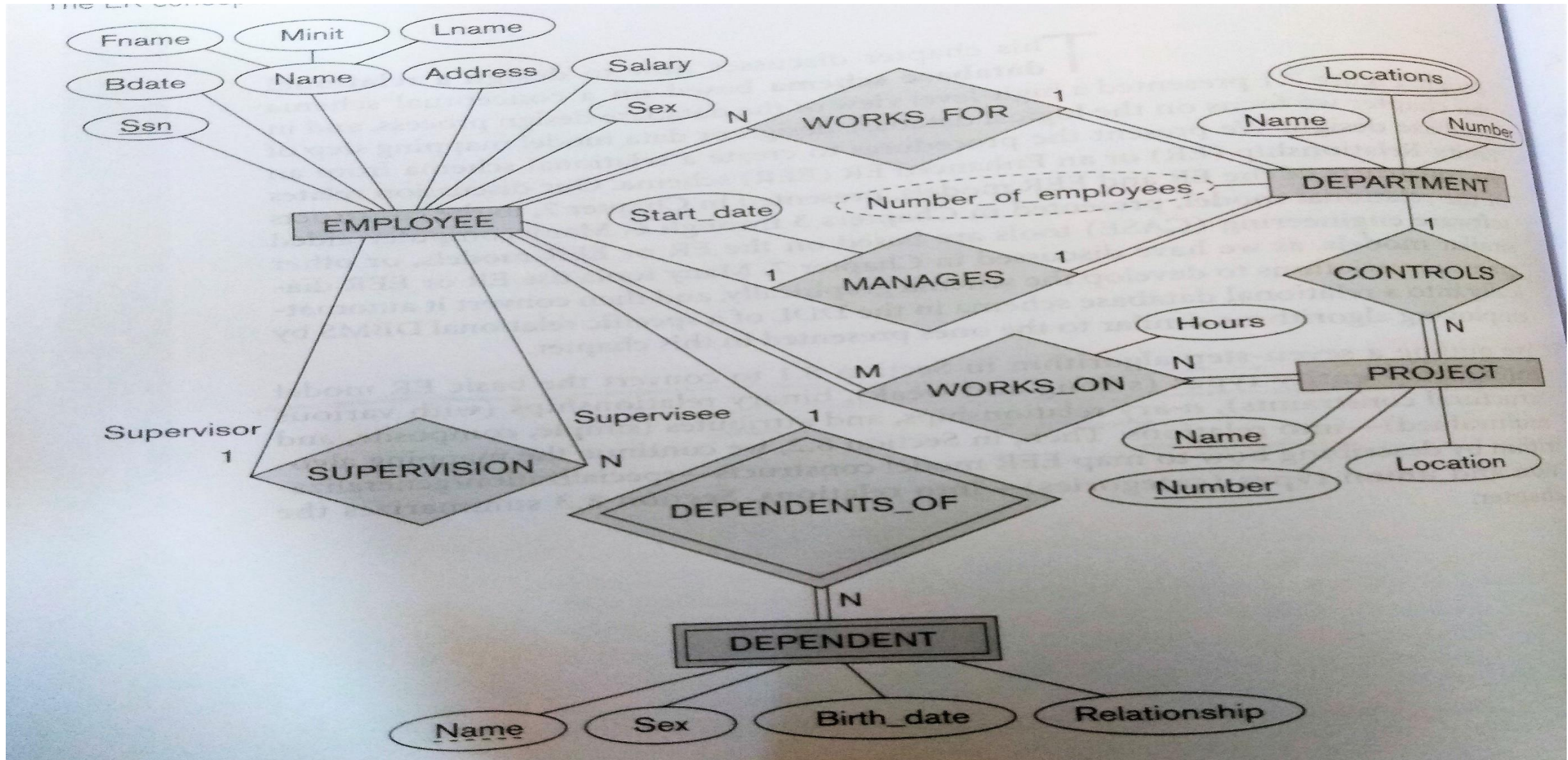
### Employee

<u>SSN</u>	Fname	Minit	Lname	DOB	Sex
------------	-------	-------	-------	-----	-----

```
create table Works_for
(
    D-ID varchar2(10),
    ssn varchar2(10),
    Hrs number(5),
    constraint pk_works_for primary key( D-ID, ssn ),
    constraint fk_works_for1 foreign key( ssn ) references employee(ssn) on delete set cascade,
    constraint fk_works_for2 foreign key( D-ID ) references department(D-ID) on delete set cascade
)
```

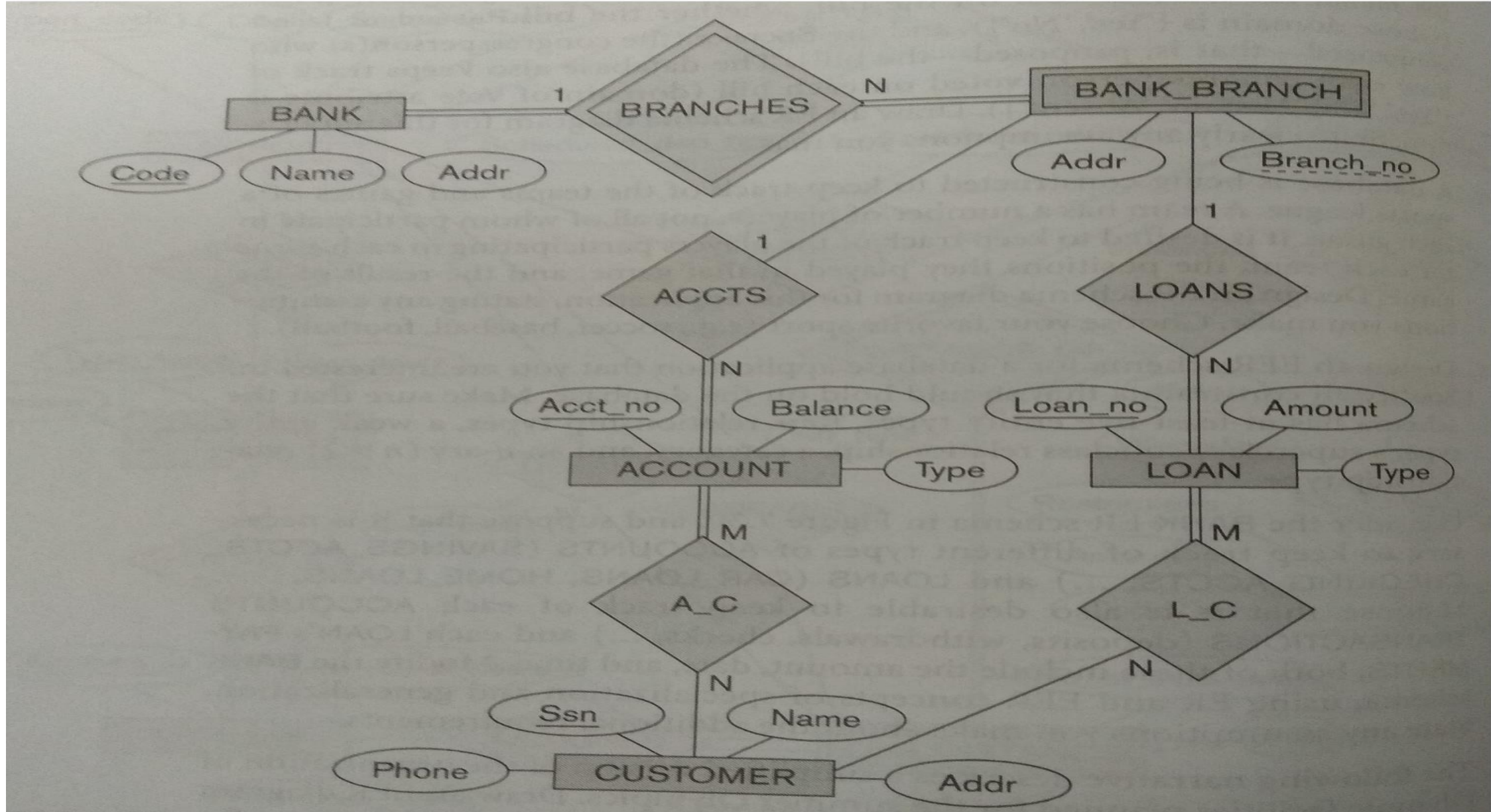


- Construct the relational schema for the given ER Diagram.





- Construct the relational schema for the given ER Diagram.



# Normalization

## **Database Design Guidelines:**

1. Semantics of the attributes in relation schema should be unambiguous and should represent the real world problem domain.
2. Redundant data in the tuples of relations should be minimal.
3. NULL values in the tuples of relations should be minimal.
4. Likelihood of generating spurious tuples should be entirely eliminated.

## **Problems due to Redundant Data:**

1. Insertion Anomaly
2. Deletion Anomaly
3. Update Anomaly

## Insertion Anomaly:

# Employee

[illegible]

## Insertion Anomaly:

- Suppose only the Employee Data is available and Department data is not available for a new employee.

# Employee

[illegible]

# Insertion Anomaly:

- Suppose both Employee Data for a new employee who works for CSE is available.

## Employee

<u>SSN</u>	E_name	H.NO	District	State	Dept-ID	D_name	Building.NO	District	State
S101	John	h45	SD	OD	D501	CSE	B98	SD	OD
S102	Mary	h98	SB	OD	D502	ECE	B96	SD	OD
S103	Mark	h76	GH	WB	NULL	NULL	NULL	NULL	NULL
s104	James	h55	MK	AP	D501	CSE	B99	SD	OD

- There is always a possibility that the data about a particular department may be different in different tuples of the same relation.



# Insertion Anomaly:

- Suppose only the data about a new Department is available.

## Employee

<u>SSN</u>	E_name	H.NO	District	State	Dept-ID	D_name	Building.NO	District	State
S101	John	h45	SD	OD	D501	CSE	B98	SD	OD
S102	Mary	h98	SB	OD	D502	ECE	B96	SD	OD
S103	Mark	h76	GH	WB	NULL	NULL	NULL	NULL	NULL
S104	James	h55	MK	AP	D501	CSE	B99	SD	OD
					D601	AI	B74	SD	OD

- But the data of the new department can not be stored in the above relation, since there is no employee in that department that will satisfy the primary key constraint of the above relation.

# Deletion Anomaly:

## Employee

<u>SSN</u>	E_name	H.NO	District	State	Dept-ID	D_name	Building.NO	District	State
S101	John	h45	SD	OD	D501	CSE	B98	SD	OD
S102	Mary	h98	SB	OD	D502	ECE	B96	SD	OD
S103	Mark	h76	GH	WB	NULL	NULL	NULL	NULL	NULL
S104	James	h55	MK	AP	D501	CSE	B99	SD	OD
S105	Martha	h51	LM	JK	D601	AI	B74	SD	OD

- Deletion of tuple with SSN=S102 will cause the data about ECE department to be completely lost from the database, if the above relation was the only relation that stored the department data in the database.
- Deletion of tuple with SSN=S105 will cause the data about AI department to be completely lost from the database, if the above relation was the only relation that stored the department data in the database.

# Update Anomaly:

## Employee

<u>SSN</u>	E_name	H.NO	District	State	Dept-ID	D_name	Building.NO	District	State
S101	John	h45	SD	OD	D501	CSE	B98	SD	OD
S102	Mary	h98	SB	OD	D502	ECE	B96	SD	OD
S103	Mark	h76	GH	WB	NULL	NULL	NULL	NULL	NULL
S104	James	h55	MK	AP	D501	CSE	B98	SD	OD
S105	Martha	h51	LM	JK	D601	AI	B74	SD	OD

- If the attributes Dept-ID and D\_name of CSE department is updated, then multiple tuples in the above relation have to be updated.
- This may cause the data of CSE department to be erroneous.

# Functional Dependency

# Functional Dependency

- It is a semantic constraint between two sets of attributes (X, Y) of a relation schema R of a database.
- It is defined according to the semantic of the attributes of R.
- It is represented as  $X \rightarrow Y$ .
- X is called **determinant** and Y is called **dependent**.
- It is read as X **functionally determines** Y.
- It is read as Y **is functionally determined by** X.
- The constraint imposed by a functional dependency,  $X \rightarrow Y$ , is that for any two tuples t1 and t2 of a **relational instance r** of **relation schema R**:  
$$\text{If } (t1.X = t2.X) \text{ then } (t1.Y = t2.Y)$$

# Functional Dependency

## Employee

<u>SSN</u>	E_name	H.NO	District	S_code	State
S101	John	h45	SD	OD	Odisha
S102	Mary	h45	SD	OD	Odisha
S103	Mark	h76	GH	WB	West Bengal
S104	John	h22	SG	Jk	Jharkand
S105	Martha	h45	MK	JK	Jharkand

# Functional Dependency

## Employee

<u>SSN</u>	E_name	H.NO	District	S_code	State
S101	John	h45	SD	OD	Odisha
S102	Mary	h45	SD	OD	Odisha
S103	Mark	h76	GH	WB	West Bengal
S104	John	h22	SG	Jk	Jharkand
S105	Martha	h45	MK	JK	Jharkand

**S\_code → State**

# Functional Dependency

## Employee

<u>SSN</u>	E_name	H.NO	District	S_code	State
S101	John	h45	SD	OD	Odisha
S102	Mary	h45	SD	OD	Odisha
S103	Mark	h76	GH	WB	West Bengal
S104	John	h22	SG	Jk	Jharkand
S105	Martha	h45	MK	JK	Jharkand

**SSN  $\rightarrow$  E\_name**



# Functional Dependency

## Employee

<u>SSN</u>	E_name	H.NO	District	S_code	State
S101	John	h45	SD	OD	Odisha
S102	Mary	h45	SD	OD	Odisha
S103	Mark	h76	GH	WB	West Bengal
S104	John	h22	SG	Jk	Jharkand
S105	Martha	h45	MK	JK	Jharkand

**SSN  $\rightarrow$  H.NO**

# Functional Dependency

## Employee

<u>SSN</u>	E_name	H.NO	District	S_code	State
S101	John	h45	SD	OD	Odisha
S102	Mary	h45	SD	OD	Odisha
S103	Mark	h76	GH	WB	West Bengal
S104	John	h22	SG	Jk	Jharkand
S105	Martha	h45	MK	JK	Jharkand

**SSN → District**

# Functional Dependency

## Employee

<u>SSN</u>	E_name	H.NO	District	S_code	State
S101	John	h45	SD	OD	Odisha
S102	Mary	h45	SD	OD	Odisha
S103	Mark	h76	GH	WB	West Bengal
S104	John	h22	SG	Jk	Jharkand
S105	Martha	h45	MK	JK	Jharkand

**SSN  $\rightarrow$  S\_code**

# Functional Dependency

## Employee

<u>SSN</u>	E_name	H.NO	District	S_code	State
S101	John	h45	SD	OD	Odisha
S102	Mary	h45	SD	OD	Odisha
S103	Mark	h76	GH	WB	West Bengal
S104	John	h22	SG	Jk	Jharkand
S105	Martha	h45	MK	JK	Jharkand

**SSN  $\rightarrow$  State**

# Functional Dependency

## Employee

<u>SSN</u>	E_name	H.NO	District	S_code	State
S101	John	h45	SD	OD	Odisha
S102	Mary	h45	SD	OD	Odisha
S103	Mark	h76	GH	WB	West Bengal
S104	John	h22	SG	Jk	Jharkand
S105	Martha	h45	MK	JK	Jharkand

**SSN  $\rightarrow$  E\_name, H.NO, District, S\_code, State**

# Functional Dependency

## Employee

<u>SSN</u>	E_name	H.NO	District	S_code	State
S101	John	h45	SD	OD	Odisha
S102	Mary	h45	SD	OD	Odisha
S103	Mark	h76	GH	WB	West Bengal
S104	John	h22	SG	Jk	Jharkand
S105	Martha	h45	MK	JK	Jharkand

**E\_name**  $\not\rightarrow$  **H.NO**

# Functional Dependency

## Employee

<u>SSN</u>	E_name	H.NO	District	S_code	State
S101	John	h45	SD	OD	Odisha
S102	Mary	h45	SD	OD	Odisha
S103	Mark	h76	GH	WB	West Bengal
S104	John	h22	SG	Jk	Jharkand
S105	Martha	h45	MK	JK	Jharkand

**E\_name**  $\not\rightarrow$  **H.NO, District, S\_code, State**

# Functional Dependency

## Employee

<u>SSN</u>	E_name	H.NO	District	S_code	State
S101	John	h45	SD	OD	Odisha
S102	Mary	h45	SD	OD	Odisha
S103	Mark	h76	GH	WB	West Bengal
S104	John	h22	SG	Jk	Jharkand
S105	Martha	h45	MK	JK	Jharkand

H.NO  $\not\rightarrow$  District



# Functional Dependency

## Employee

<u>SSN</u>	E_name	H.NO	District	S_code	State
S101	John	h45	SD	OD	Odisha
S102	Mary	h45	SD	OD	Odisha
S103	Mark	h76	GH	WB	West Bengal
S104	John	h22	SG	Jk	Jharkand
S105	Martha	h45	MK	JK	Jharkand

H.NO  $\not\rightarrow$  District, S\_code, State

# Functional Dependency

<u>SSN</u>	E_name	Mobile	Bank_Account_No	S_code	State
S101	John	9876555412	B_A11298	OD	Odisha
S102	Mary	8876555467	B_A41298	OD	Odisha
S103	Mark	8476555168	B_A11399	WB	West Bengal
S104	John	9976545411	B_A41290	Jk	Jharkand
S105	Martha	9876555410	B_A51253	JK	Jharkand

**Super Key:**  $X = \{SSN, E\_name, Mobile, Bank\_Account\_No, S\_code, State\}$

$$X \longrightarrow X$$

- This is a trivial Functional Dependency

# Functional Dependency

<u>SSN</u>	E_name	Mobile	Bank_Account_No	S_code	State
S101	John	9876555412	B_A11298	OD	Odisha
S102	Mary	8876555467	B_A41298	OD	Odisha
S103	Mark	8476555168	B_A11399	WB	West Bengal
S104	John	9976545411	B_A41290	Jk	Jharkand
S105	Martha	9876555410	B_A51253	JK	Jharkand

**Super Key:**  $X = \text{SSN, Mobile, Bank\_Account\_No}$

$X \longrightarrow \text{E\_name, S\_code, State}$

# Functional Dependency

<u>SSN</u>	E_name	Mobile	Bank_Account_No	S_code	State
S101	John	9876555412	B_A11298	OD	Odisha
S102	Mary	8876555467	B_A41298	OD	Odisha
S103	Mark	8476555168	B_A11399	WB	West Bengal
S104	John	9976545411	B_A41290	Jk	Jharkand
S105	Martha	9876555410	B_A51253	JK	Jharkand

**Super Key: X = SSN, Mobile**

**$X \longrightarrow E\_name, Bank\_Account\_No, S\_code, State$**

# Functional Dependency

<u>SSN</u>	E_name	Mobile	Bank_Account_No	S_code	State
S101	John	9876555412	B_A11298	OD	Odisha
S102	Mary	8876555467	B_A41298	OD	Odisha
S103	Mark	8476555168	B_A11399	WB	West Bengal
S104	John	9976545411	B_A41290	Jk	Jharkand
S105	Martha	9876555410	B_A51253	JK	Jharkand

**Super Key: X = SSN, Bank\_Account\_No**

**$X \longrightarrow E\_name, Mobile, S\_code, State$**

# Functional Dependency

<u>SSN</u>	E_name	Mobile	Bank_Account_No	S_code	State
S101	John	9876555412	B_A11298	OD	Odisha
S102	Mary	8876555467	B_A41298	OD	Odisha
S103	Mark	8476555168	B_A11399	WB	West Bengal
S104	John	9976545411	B_A41290	Jk	Jharkand
S105	Martha	9876555410	B_A51253	JK	Jharkand

**Super Key: X = Mobile, Bank\_Account\_No**

**$X \longrightarrow \text{SSN, E\_name, S\_code, State}$**

# Functional Dependency

<u>SSN</u>	E_name	Mobile	Bank_Account_No	S_code	State
S101	John	9876555412	B_A11298	OD	Odisha
S102	Mary	8876555467	B_A41298	OD	Odisha
S103	Mark	8476555168	B_A11399	WB	West Bengal
S104	John	9976545411	B_A41290	Jk	Jharkand
S105	Martha	9876555410	B_A51253	JK	Jharkand

**Candidate Key: X = Mobile**

**X  $\longrightarrow$  SSN, E\_name, Bank\_Account\_No, S\_code, State**

# Functional Dependency

<u>SSN</u>	E_name	Mobile	Bank_Account_No	S_code	State
S101	John	9876555412	B_A11298	OD	Odisha
S102	Mary	8876555467	B_A41298	OD	Odisha
S103	Mark	8476555168	B_A11399	WB	West Bengal
S104	John	9976545411	B_A41290	Jk	Jharkand
S105	Martha	9876555410	B_A51253	JK	Jharkand

**Candidate Key: X = Bank\_Account\_No**

**X  $\longrightarrow$  SSN, E\_name, Mobile, S\_code, State**



# Functional Dependency

<u>SSN</u>	E_name	Mobile	Bank_Account_No	S_code	State
S101	John	9876555412	B_A11298	OD	Odisha
S102	Mary	8876555467	B_A41298	OD	Odisha
S103	Mark	8476555168	B_A11399	WB	West Bengal
S104	John	9976545411	B_A41290	Jk	Jharkand
S105	Martha	9876555410	B_A51253	JK	Jharkand

**Primary Key:** is One of the candidate keys.

# Functional Dependency

<u>SSN</u>	E_name	Mobile	Bank_Account_No	S_code	State
S101	John	9876555412	B_A11298	OD	Odisha
S102	Mary	8876555467	B_A41298	OD	Odisha
S103	Mark	8476555168	B_A11399	WB	West Bengal
S104	John	9976545411	B_A41290	Jk	Jharkand
S105	Martha	9876555410	B_A51253	JK	Jharkand

**Primary Key:** is One of the candidate keys.

# Functional Dependency

- FDs can increase **data redundancy** and its associated problems.

<u>Roll</u>	Name	Sub_Code	Subject	Sub_Teacher
S101	John	CS102	DBMS	RAM
S102	Mary	CS102	DBMS	RAM
S103	Mark	CS102	DBMS	RAM
S104	John	CS104	DAA	Gautam
S105	Martha	CS104	DAA	Gautam

Sub\_Code  $\longrightarrow$  Subject, Sub\_Teacher

Roll  $\longrightarrow$  Name, Sub\_Code, Subject, Sub\_Teacher


Subject  $\longrightarrow$  Sub\_Teacher

# Functional Dependency

- FDs can increase **data redundancy** and its associated problems.

<u>Roll</u>	Name	Sub_Code	Subject	Sub_Teacher
S101	John	CS102	DBMS	RAM
S102	Mary	CS102	DBMS	RAM
S103	Mark	CS102	DBMS	RAM
S104	John	CS104	DAA	Gautam
S105	Martha	CS104	DAA	Gautam

Sub\_Code  $\longrightarrow$  Subject, Sub\_Teacher



<u>Roll</u>	Name	Sub_Code
S101	John	CS102
S102	Mary	CS102
S103	Mark	CS102
S104	John	CS104
S105	Martha	CS104

<u>Sub_Code</u>	Subject	Sub_Teacher
CS102	DBMS	RAM
CS104	DAA	Gautam

**Redundancy is Minimized**

# Normal Forms:

1. First Normal Form
2. Second Normal Form
3. Third Normal Form
4. Boyce-Codd Normal Form
5. Fourth Normal Form
6. Fifth Normal Form

# First Normal Forms(1NF):

1. First Normal Form