

# Entity-Relationship (ER) Model for Conceptual Data Modeling

CS 301

# “Goal of the chapter”

“Learn about the modeling concepts of the entity-relationship (ER) model, a popular high level conceptual data model.”

1. Discuss basic data-structuring concepts (entity, attributes, relationships) and constraints of the ER model.
2. Discuss the use of ER model in the design of conceptual schema for DB applications.
3. Discuss diagrammatic notations associated with the ER model called the ER diagrams.


# Using high level conceptual data models for DB design

- A major part of the DB application requires design, implementation and testing of the application programs (*often considered part of software engineering*).
- In many software design tools, the DB design and software design methodologies are intertwined as they are related.



- DB design process involves:

1. Requirement collection and analysis – The DB designers interview DB users to understand the data requirements.
2. Collection of functional requirements of the application such as user defined operations (transactions) like updation, retrievals, etc. are also specified.
3. Creation of conceptual schema for the DB using a high-level conceptual data model – through the conceptual design.
  - Conceptual schema is a concise description of the data requirements of the users, entity types, relationships, constraints, etc. - they are easy to understand because they do not have implementation details.
  - The approach allows DB designers to specify the properties of the data without being bothered about storage and implementation details.

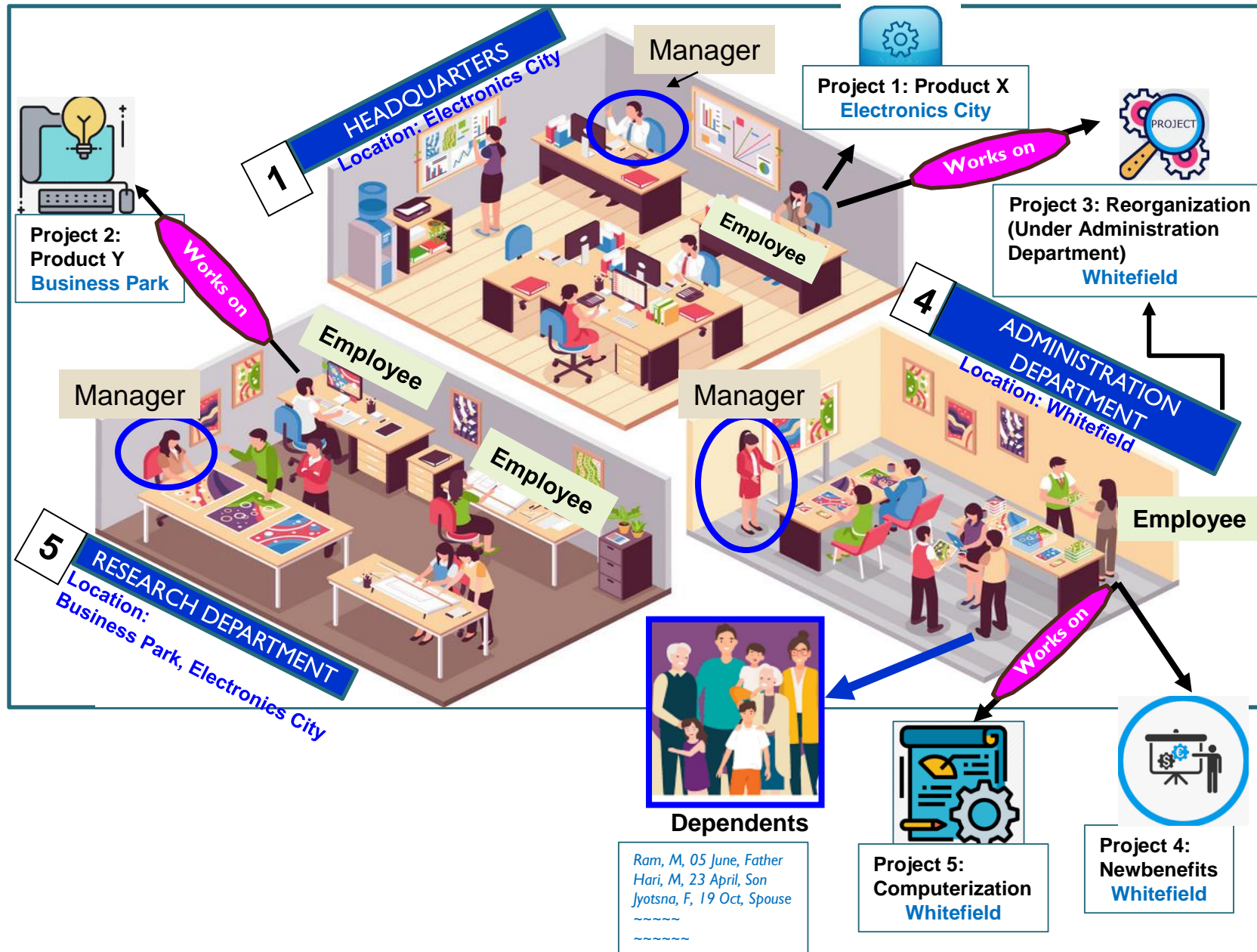
- 
4. Actual implementation of the DB through SQL – so the conceptual schema is transformed from high level data model to implementation data model – known as logic design or data model mapping.
  5. Physical design phase – the internal storage structures, file organisations, indexes, access path, and physical design parameters for the DB files are specified.
  6. Application programs are developed on top of all these models.

# A DB application: Company DB

- Let's understand the ER model and its role in the schema design through a “Company” DB.



# COMPANY - DEPARTMENTS





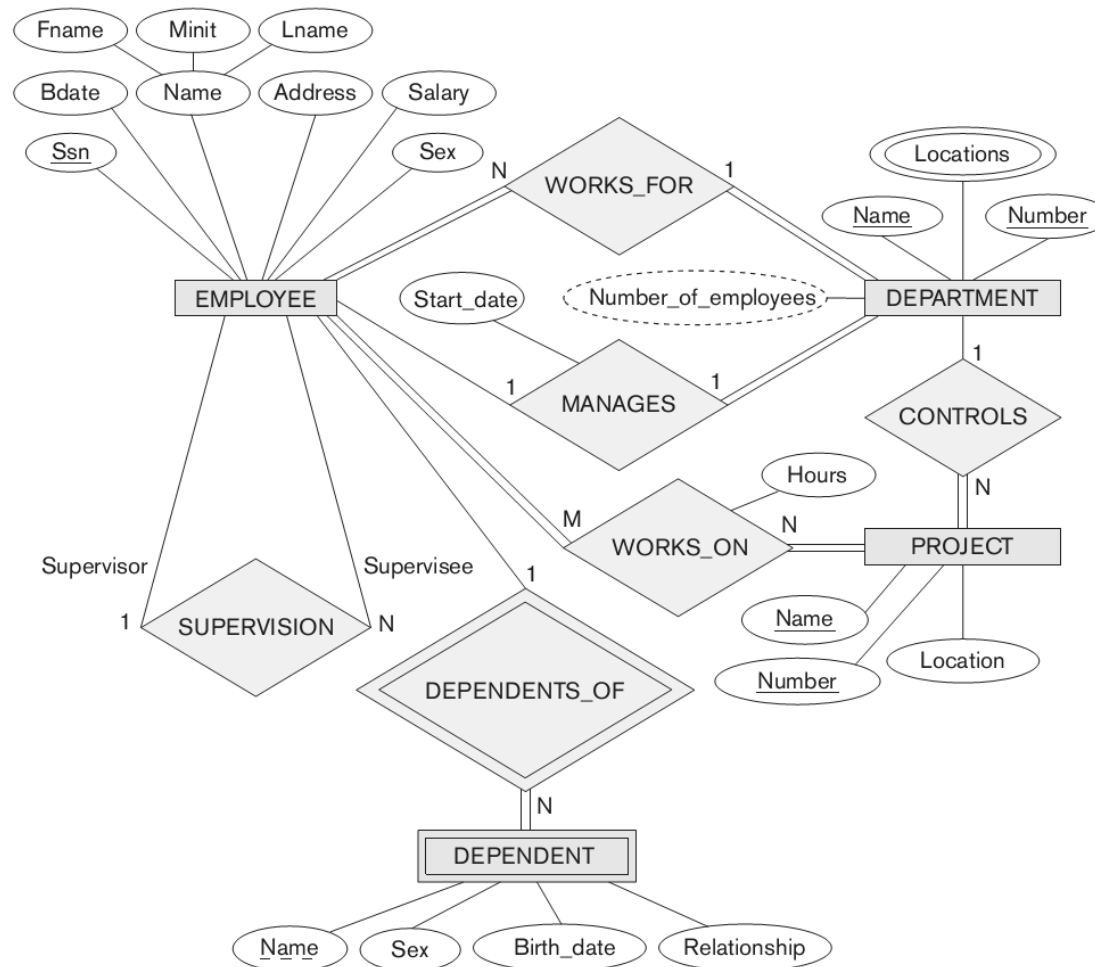
# A DB application: Company DB

- The Company DB has details of employees, departments and projects.

- Situation –

- The company has departments at various locations with a name, number, and an employee manager who manages the department. The manager has a start date for managing the department.
- The department controls a number of projects with name, number and a location.
- The DB should have employee's name, SSN, address, salary, gender and birth date.
- An employee is assigned to one department but may work on several projects, not necessarily controlled by the same department.

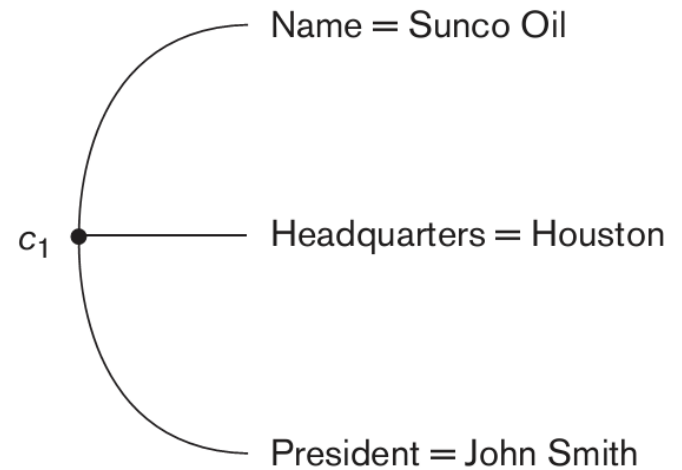
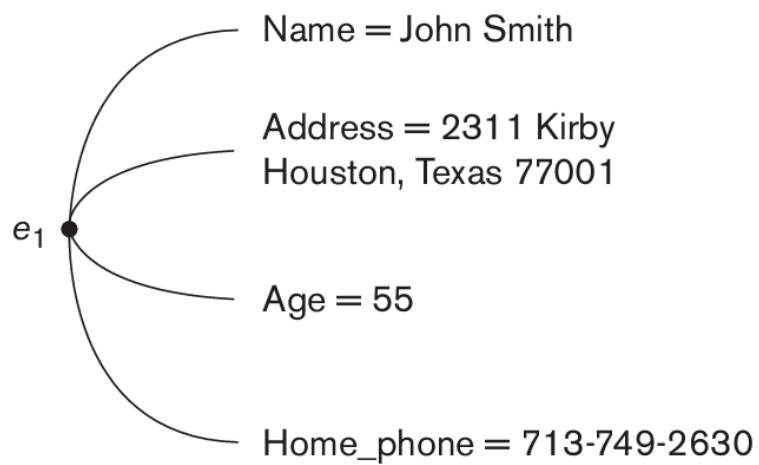
- An employee may work for a number of hours per week on each project as well as the supervisor of each employee.
- Each employee has dependents that have first name, gender, birth date and relationship to the employee.



ER schema diagram for the Company DB.

# Entity, attributes and keys

- ER model describes data as entities, relationships, and attributes.
- Entity – An entity is an object or thing with a physical and an independent existence in the real world. Example, car, house, employee, etc. Or it may have a conceptual existence (company, job, course, etc.)
- Attribute – Each entity has attributes i.e. properties that describe the entity. Example, employee's name, address, age, etc.
- A particular entity will have a value for each of its attributes which is stored in the DB.



Entities –  $e_1$  (employee) and  $c_1$  (company) and their attributes.

# Types of attributes

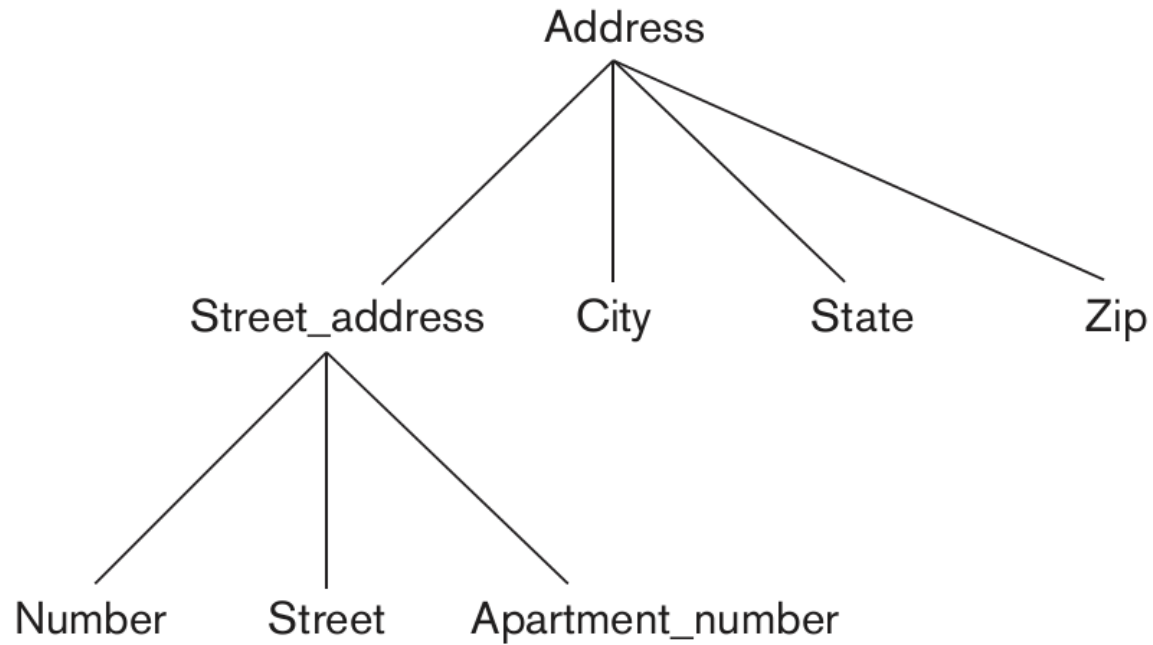
- Composite attributes
- Simple (atomic) attributes
- Single valued
- Multivalued
- Stored
- Derived
- NULL
- Complex attributes

# Types of attributes

- Composite attributes – Composite attributes can be divided into smaller subparts, which represent more basic attributes with independent meanings. Example, Address attribute can be subdivided into Street\_address, City, State and Zip.
  - They can form a hierarchy. Example, Street\_address can be further subdivided into three simple component attributes: Number, Street and Apartment\_number.
  - The value of a composite attribute is the concatenation of the values of its component simple attributes.
  - If there is no need to refer to the individual components – such as address, then the whole address can be designated as a simple attribute.
- Simple (atomic) attributes – Attributes that are not divisible . Example, gender, SSN.



# Example of a composite attribute



Composite attributes.

- Single-valued attributes – If the attributes have a single value for a particular entity. Example, age.
- Multivalued attributes – An entity may have multiple values for that attribute. Example, Cars of different *Colours*, Degree(s) of a person – no degree, high school degree, college degree – therefore, different people can have different number of values for the *college\_degree* attribute.
  - It may have lower and upper bounds to constraint the number of values that are allowed for each individual entity.
- *Composite* and *multivalued* attributes can be nested arbitrarily.

- Stored attributes – The original attribute value stored in the DB. Example, Birth\_date.
- Derived attributes – Some attributes can be derived from related attributes.
  - Example, Age and Birth\_date of a person. So, age can be determined from the current (today's) date and the person's Birth\_date.
  - Example, number\_of\_employees in a department entity can be derived by counting the number of employees related to that department.

- NULL values – Sometimes, an entity may not have an applicable value.
  - Example, apartment\_number of an address in a single family homes. Address of a single family home would be NULL for its apartment\_number attribute.
  - Example, college\_degrees are applied only to people who have degrees. Here a person with no college degree would be NULL. Here NULL means not applicable.
- NULL can also be used if we do not know the value of an attribute for a particular entity.
  - Example, if we do not know the home phone number. Here NULL means unknown.
- Unknown category of NULL can be further classified into two cases.
  - When it is known that the attribute value exist but is missing – height of a person.
  - When it is not known whether the attribute value exists – home\_phone attribute of a person.

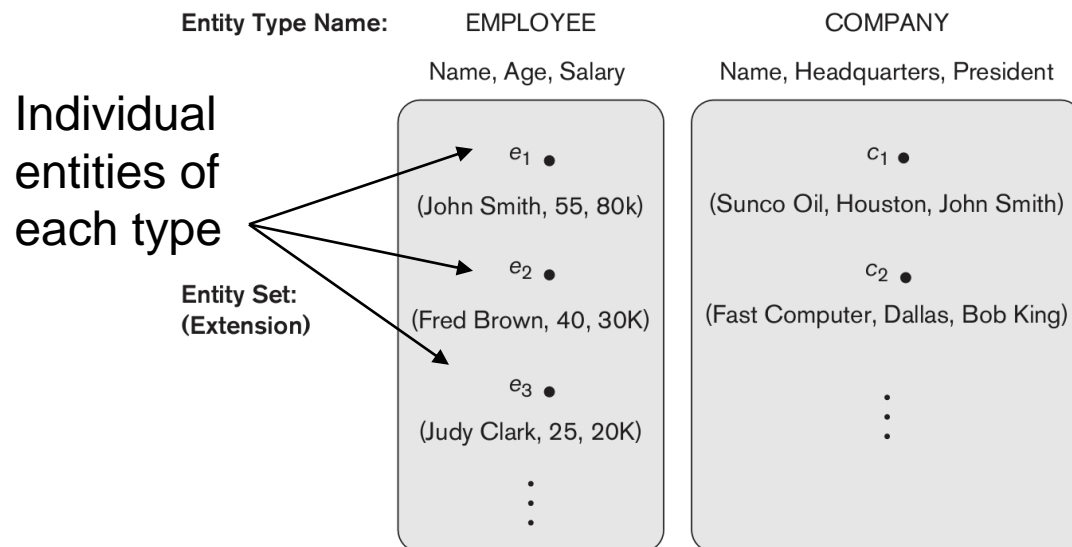
- Complex attributes – Given that *composite* and *multivalued* attributes can be nested arbitrarily, we can group components of a composite attribute between parentheses ( ) with the components separated by commas, and display multivalued attributes between braces { }. These are complex attributes.

- Example, if a person can have more than one residence and each residence can have a single address and multiple phones, an attribute *Address\_phone* for a person can be specified. Both phone and address are composite attributes.

{Address\_phone( {Phone(Area\_code,Phone\_number)},Address(Street\_address  
(Number,Street,Apartment\_number),City,State,Zip) )}

# Entity types, entity sets, keys and value sets

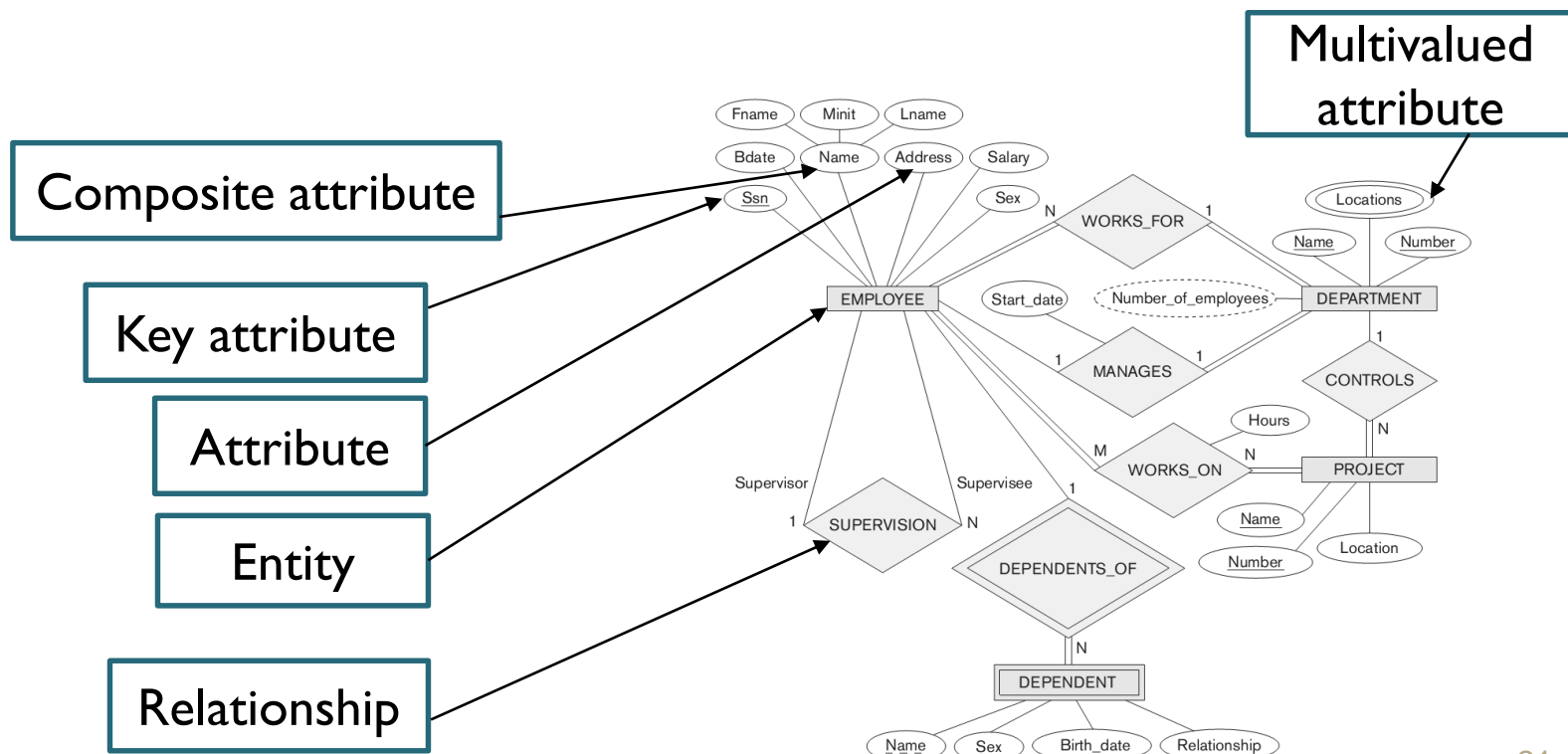
- A DB contains similar groups of entities, for example, employees in a company. The employee entities share the same attributes, but each entity has its own value(s) for each attribute.
- Entity type – It is a collection (or set) of entities that have same attributes. Each entity type in the DB is described by its name and attributes.



Two entity types, EMPLOYEE and COMPANY with some member entities of each.



- Entity set (collection) - The collection of all entities of a particular entity type in the DB at any point in time is called an entity set or entity collection.
- The entity set is usually referred to using the same name as the entity type.
  - Example, “employee” refers to both a *type of entity* as well as the current collection of *all employee entities* in the DB.

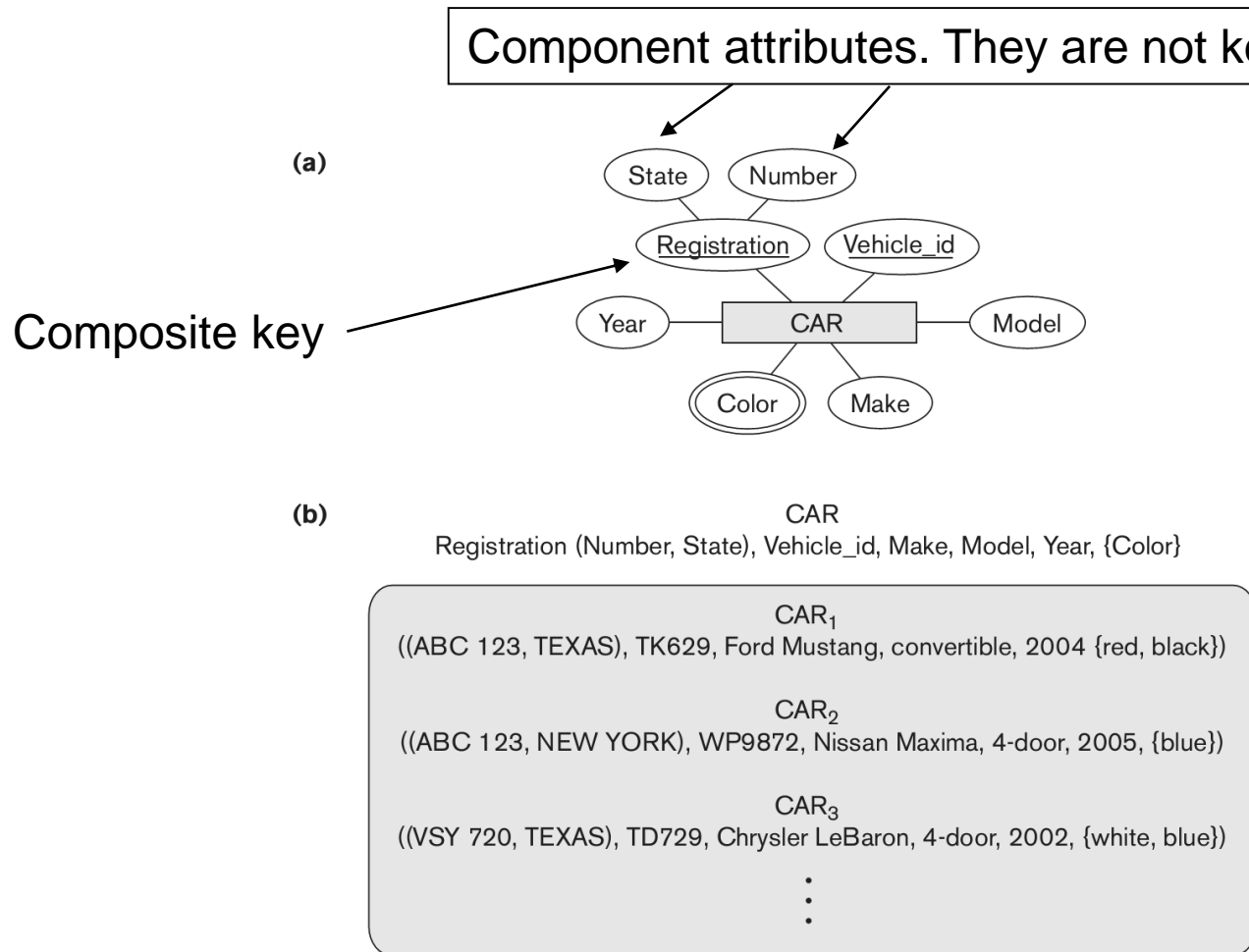


# Key attributes of an entity type

- An entity type usually has one or more attributes whose values are “distinct” for each individual entity in the entity set – called a “key attribute” and its values can be used to identify each entity uniquely.
- Example, the Name attribute is a key of the Company entity type because no two companies are allowed to have same name. For a person, key attribute is SSN/Aadhar.
- Sometimes, several attributes together (combination of attribute values) form a key. So, a composite attribute can be key attribute.

So, “key” is not a property of a particular entity set, rather it is a constraint on any entity set of the entity type that prohibits any two entities from having the same value for the key attribute at the same time.

- Some entity types have more than one key attribute. Example, *vehicle\_id* and *registration* attributes of the entity type CAR.
- If an entity type has *no key*, it is called a *weak entity type*.



- (a) CAR entity type with two key attributes – Registration and Vehicle\_id.
- (b) Entity set with three entities.

# Value sets (domains of attributes)

- Each simple attribute of an entity type is associated with a value set (or domain of values), which specifies the set of values that may be assigned to that attribute for each individual entity.
- Example, age of a person is a set of integer numbers.

- Mathematically, an attribute  $A$  of entity set  $E$  whose value set is  $V$  is a function from  $E$  to the power set  $* P(V)$  of  $V$ :

$$A : E \rightarrow P(V)$$

We refer to the value of attribute  $A$  for entity  $e$  as  $A(e)$ . The previous definition covers both single-valued and multivalued attributes, as well as NULLs. A NULL value is represented by the *empty set*. For single-valued attributes,  $A(e)$  is restricted to being a *singleton set* for each entity  $e$  in  $E$ , whereas there is no restriction on multivalued attributes.

\* A power set of a Set  $A$  denoted by “ $P(A)$ ” is defined as **the set (or group) of all subsets of the Set  $A$** , including the Set itself and the empty/NULL set, which is denoted by  $\{\}$  or  $\phi$ .  
Example: Power set of  $\{1, 2, 3\} = \{\phi, \{1\}, \{2\}, \{3\}, \{1, 2\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$ .

For a composite attribute  $A$ , the value set  $V$  is the power set of the Cartesian product\* of  $P(V_1), P(V_2), \dots, P(V_n)$ , where  $V_1, V_2, \dots, V_n$  are the value sets of the simple component attributes that form  $A$ :

$$V = P(P(V_1) \times P(V_2) \times \dots \times P(V_n))$$

- The value set provides all possible values. Only a small number of these values exist in the DB at a time.

\* The Cartesian product  $X \times Y$  between two sets  $X$  and  $Y$  is the set of all possible ordered pairs with first element from  $X$  and second element from  $Y$ :  $X \times Y = \{(x, y) : x \in X \text{ and } y \in Y\}$ .

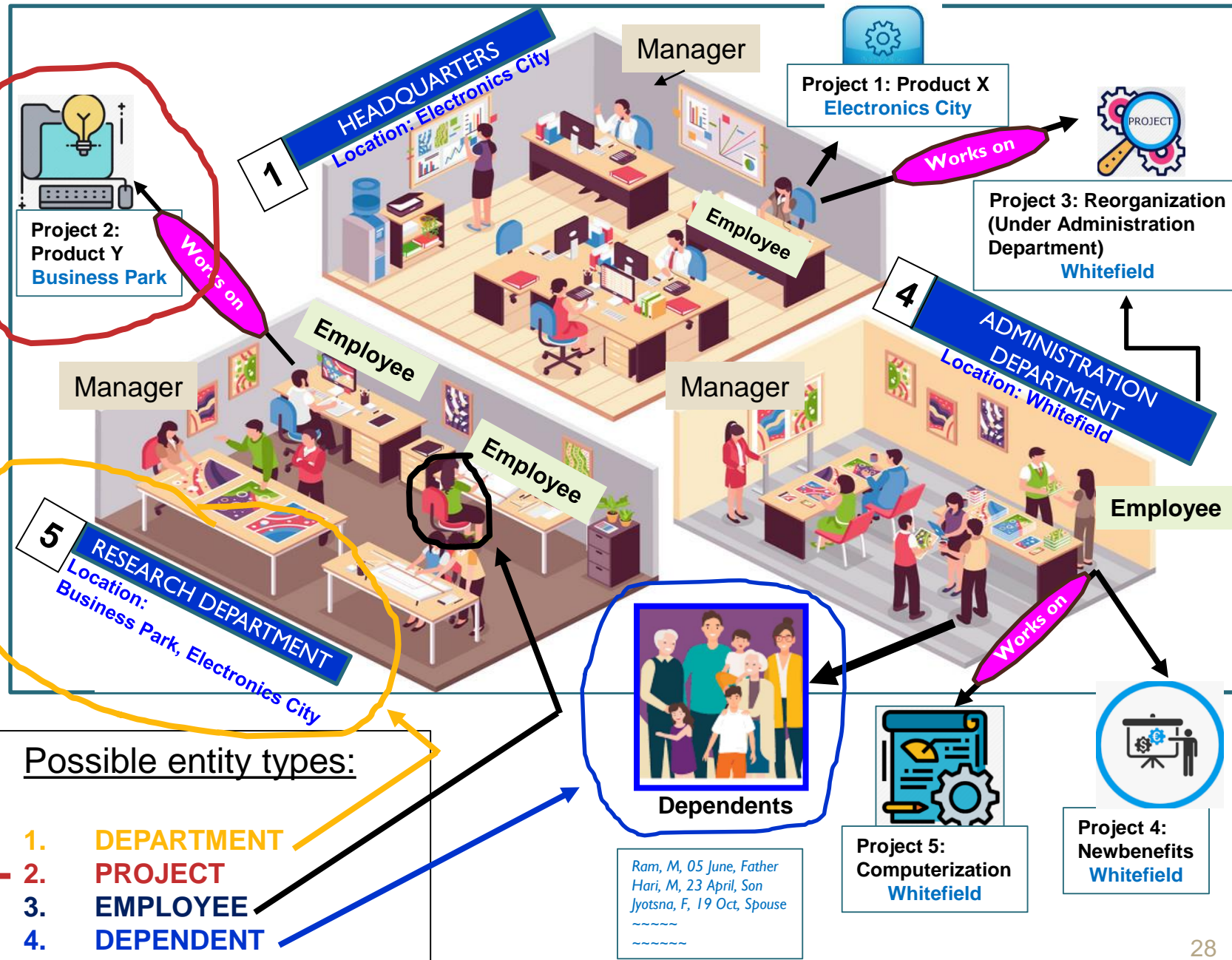
\* The Cartesian Product of sets  $A$  and  $B$  is defined as the **set of all ordered pairs  $(x, y)$  such** that  $x$  belongs to  $A$  and  $y$  belongs to  $B$ . For example, if  $A = \{1, 2\}$  and  $B = \{3, 4, 5\}$ , then the Cartesian Product of  $A$  and  $B$  is  $\{(1, 3), (1, 4), (1, 5), (2, 3), (2, 4), (2, 5)\}$ .

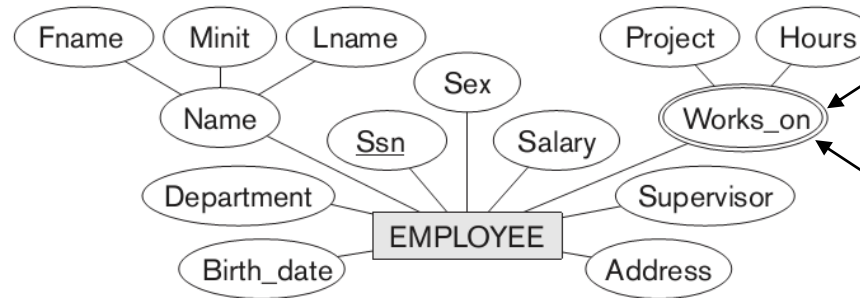
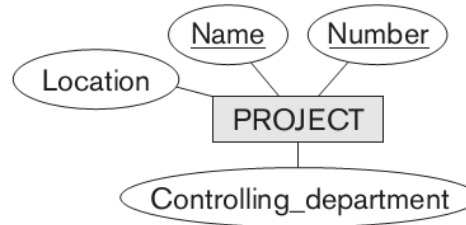
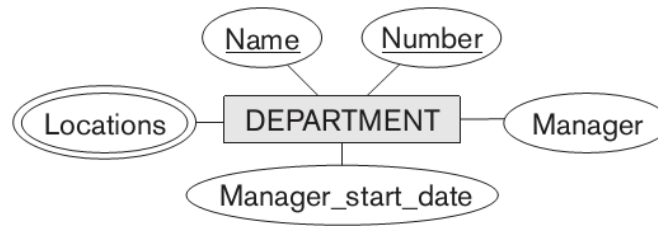


# Initial design of the company DB

- Based on the requirements, we can identify four entity types in the Company DB.
  1. DEPARTMENT
  2. PROJECT
  3. EMPLOYEE
  4. DEPENDENT

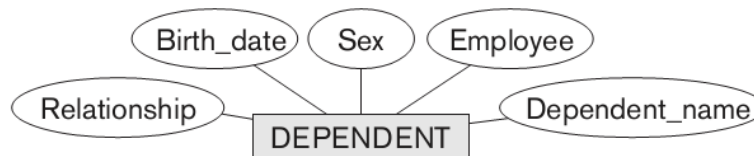
# COMPANY - DEPARTMENTS





Multivalued  
composite  
attribute

An employee can  
work on several  
projects.



Design of entity types for the Company DB.

# Relationship

- Whenever an attribute of one entity type refers to another entity type, some relationship exist.
  - Example, the attribute “Manager” of Department is an employee who “manages” the department. The Department of Employee refers to the department for which the employee “works”.
- In ER model, these references are not represented as attributes but as “relationships”.



# Relationship types, sets and instances

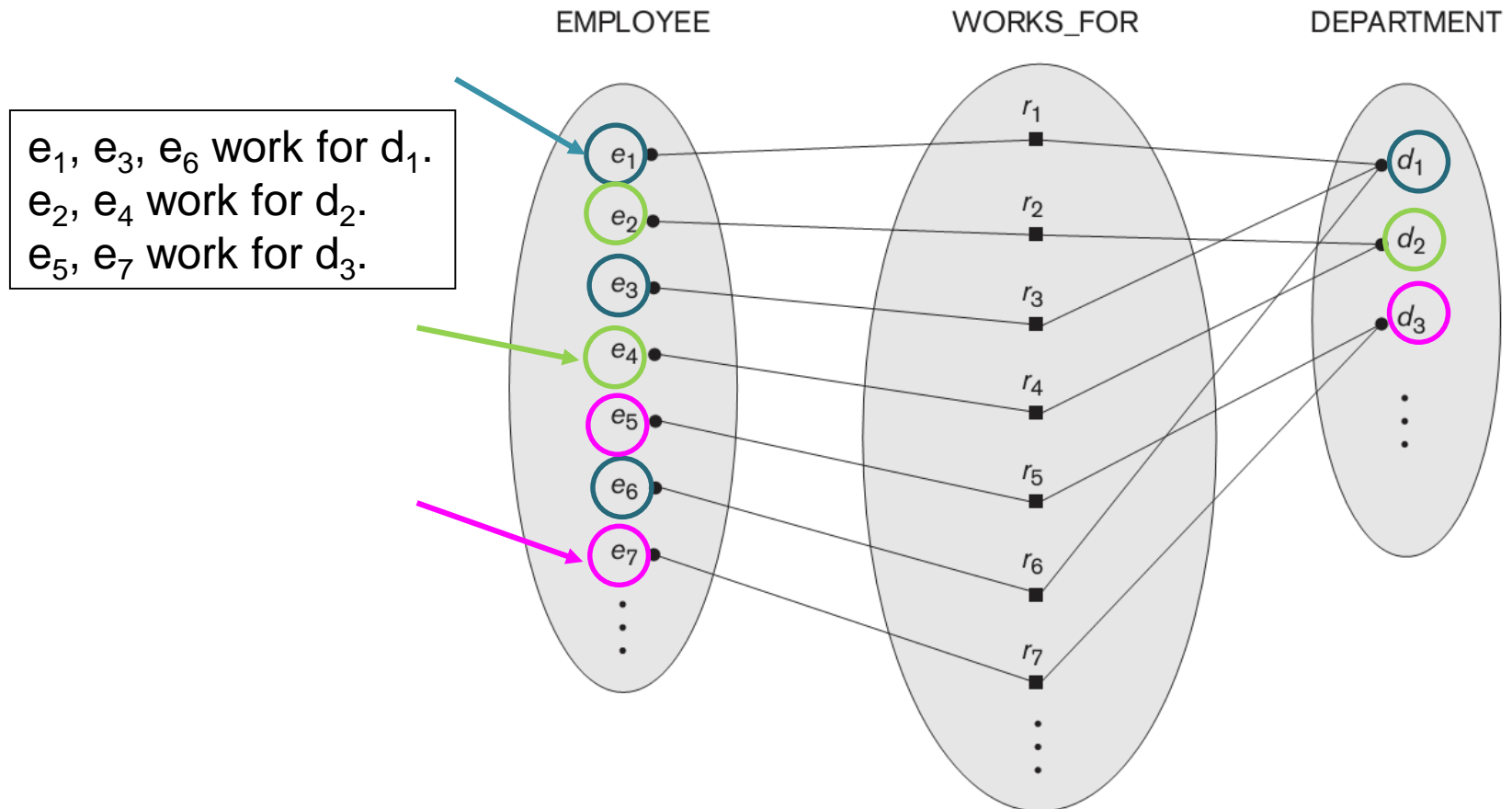
A **relationship type**  $R$  among  $n$  entity types  $E_1, E_2, \dots, E_n$  defines a set of associations—or a **relationship set**—among entities from these entity types. Similar to the case of entity types and entity sets, a relationship type and its corresponding relationship set are customarily referred to by the *same name*,  $R$ .

Mathematically,

relationship set  $R$  is a set of **relationship instances**  $r_i$ , where each  $r_i$  associates  $n$  individual entities  $(e_1, e_2, \dots, e_n)$ , and each entity  $e_j$  in  $r_i$  is a member of entity set  $E_j$ ,  $1 \leq j \leq n$ . Hence, a relationship set is a mathematical relation on  $E_1, E_2, \dots, E_n$ ; alternatively, it can be defined as a subset of the Cartesian product of the entity sets  $E_1 \times E_2 \times \dots \times E_n$ . Each of the entity types  $E_1, E_2, \dots, E_n$  is said to **participate** in the relationship type  $R$ ; similarly, each of the individual entities  $e_1, e_2, \dots, e_n$  is said to **participate** in the relationship instance  $r_i = (e_1, e_2, \dots, e_n)$ .

- So, each relationship instance  $r_i$  in  $R$  is an association of entities, where the association includes exactly one entity from each participating entity type.
- Each such relationship instance  $r_i$  represents the fact that the entities participating in  $r_i$  are related.

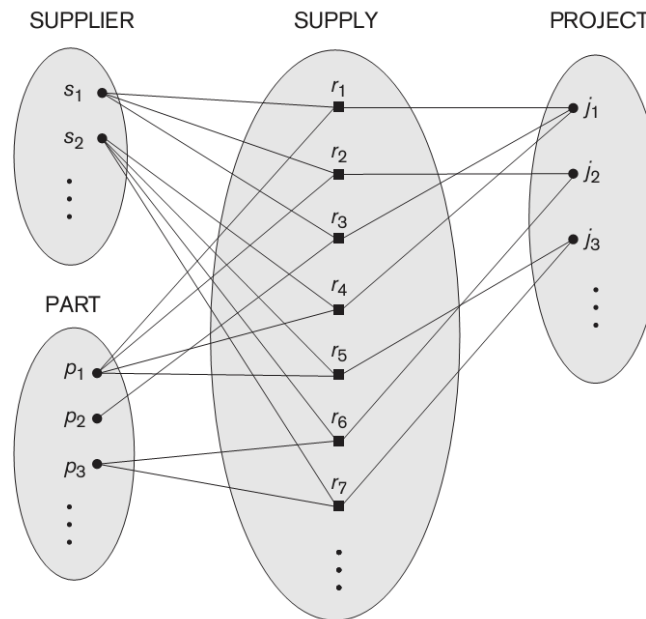
- Example, consider WORKS\_FOR between Employee and Department, which associates each employee with the department for which the employee works,
- So, each relationship instance in WORKS\_FOR associates one Employee and one Department entity.





# Relationship degree, role names, and recursive relationships

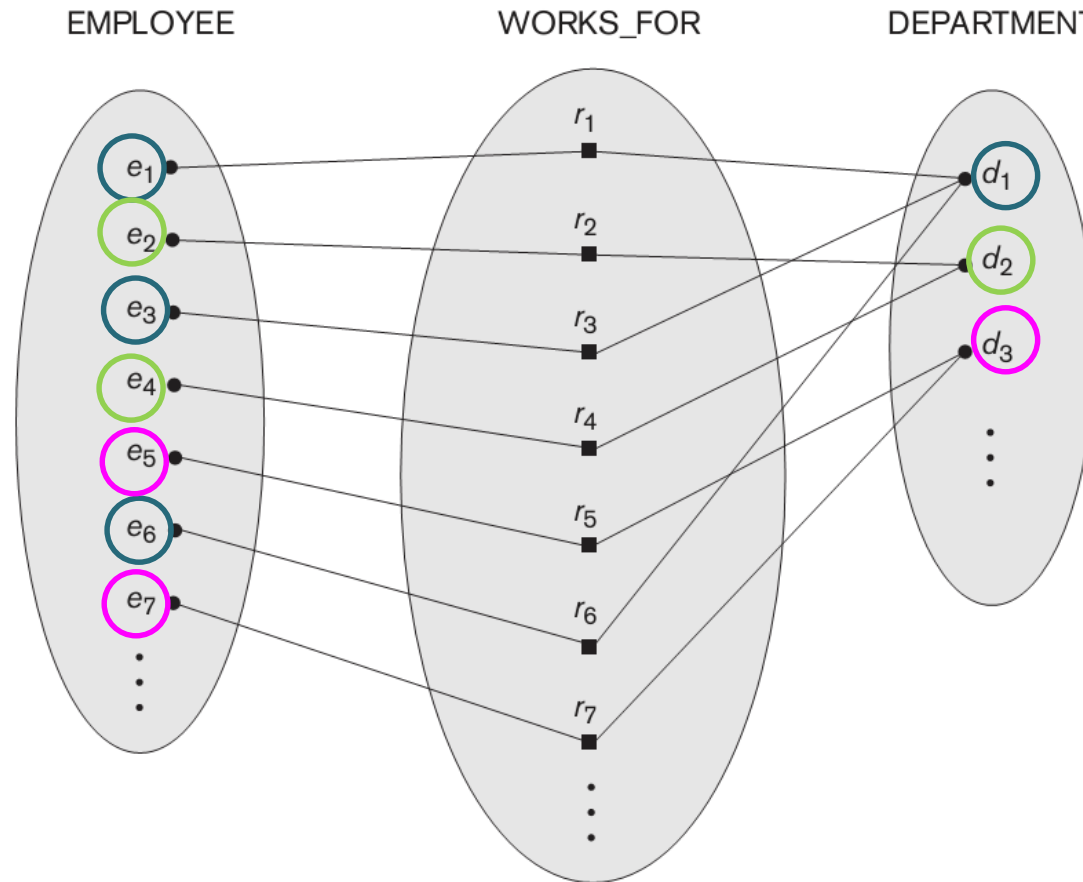
- Degree of relationship – It is the number of participating entity types. Example, WORKS\_FOR relationship is of degree “two” (binary). If degree is “three” it is ternary. Example,



- Here, each relationship instance  $r_i$  associates three entities – a supplier  $s$ , a part  $p$ , and a project  $j$  – whenever  $s$  supplies part  $p$  to project  $j$ .

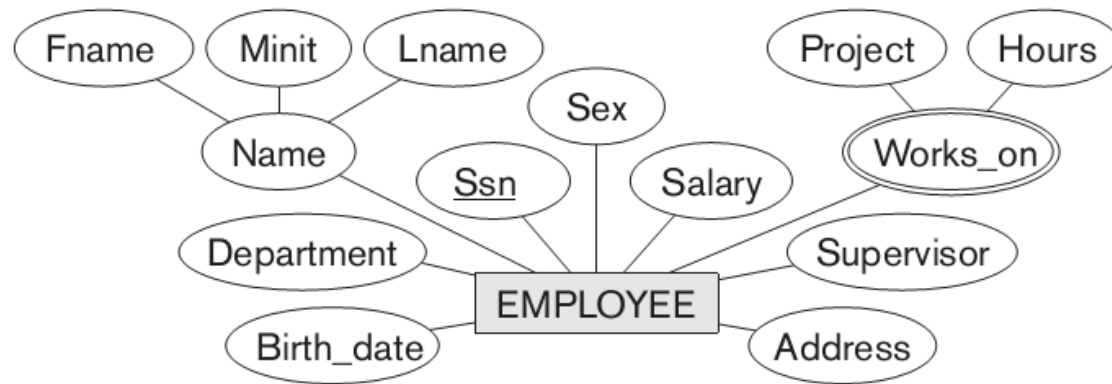
# Relationships as attributes

- Sometimes relationships can be expressed in “terms of attributes”.



- Consider for “WORKS\_FOR” relationship. One can think of an attribute called “Department” of EMPLOYEE, where value of “Department” of each EMPLOYEE entity is DEPARTMENT entity for which that employee works.

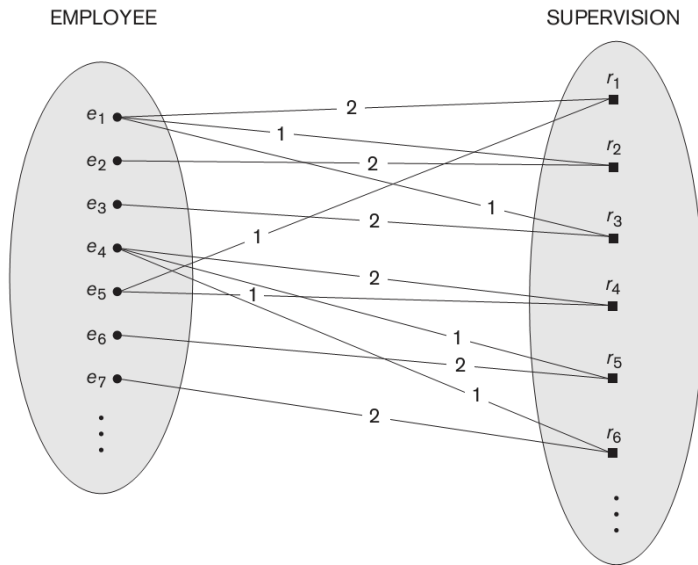
- Therefore, the value set for this “Department” attribute is the set of *all* DEPARTMENT entities (i.e. DEPARTMENT entity set).



- Therefore, there are either of these two attributes – Department of EMPLOYEE or Employees of DEPARTMENT – can represent the WORKS\_FOR relationship type. If both are represented, they are constrained to be inverses of each other.

# Role name and recursive relationships

- Each entity type that participates in a relationship type plays a role – it signifies the “role” that a participating entity from the entity type plays and helps to explain what the relationship means. Example, in WORKS\_FOR relationship, EMPLOYEE plays the role of “employee”.
- When the same entity type participates more than once in a relationship type in different roles, role names are important. These are called recursive relationships or self-referencing relationships.



A recursive relationship.

The line marked “1” represent the supervisor role, and those marked “2” represent the supervisee role.

e<sub>1</sub> supervises e<sub>2</sub> and e<sub>3</sub>,  
e<sub>4</sub> supervises e<sub>6</sub> and e<sub>7</sub> and  
e<sub>5</sub> supervises e<sub>1</sub> and e<sub>4</sub>.

- SUPERVISION relationship relates a supervisor to an employee, where both supervisor and employee entities are members of the EMPLOYEE entity set.
- Hence, EMPLOYEE entity type participates twice in SUPERVISION: once in the role of supervisor and once supervisee.

# Constraints on binary relationship types

- Relationships have certain constraints limiting the possible combinations of entities participating in the relationship set – often determined from the real world situation.
  - Example, a rule saying that “each employee must work for exactly one department”.
- There are two types of binary relationship constraints: *cardinality ratio* and *participation*.

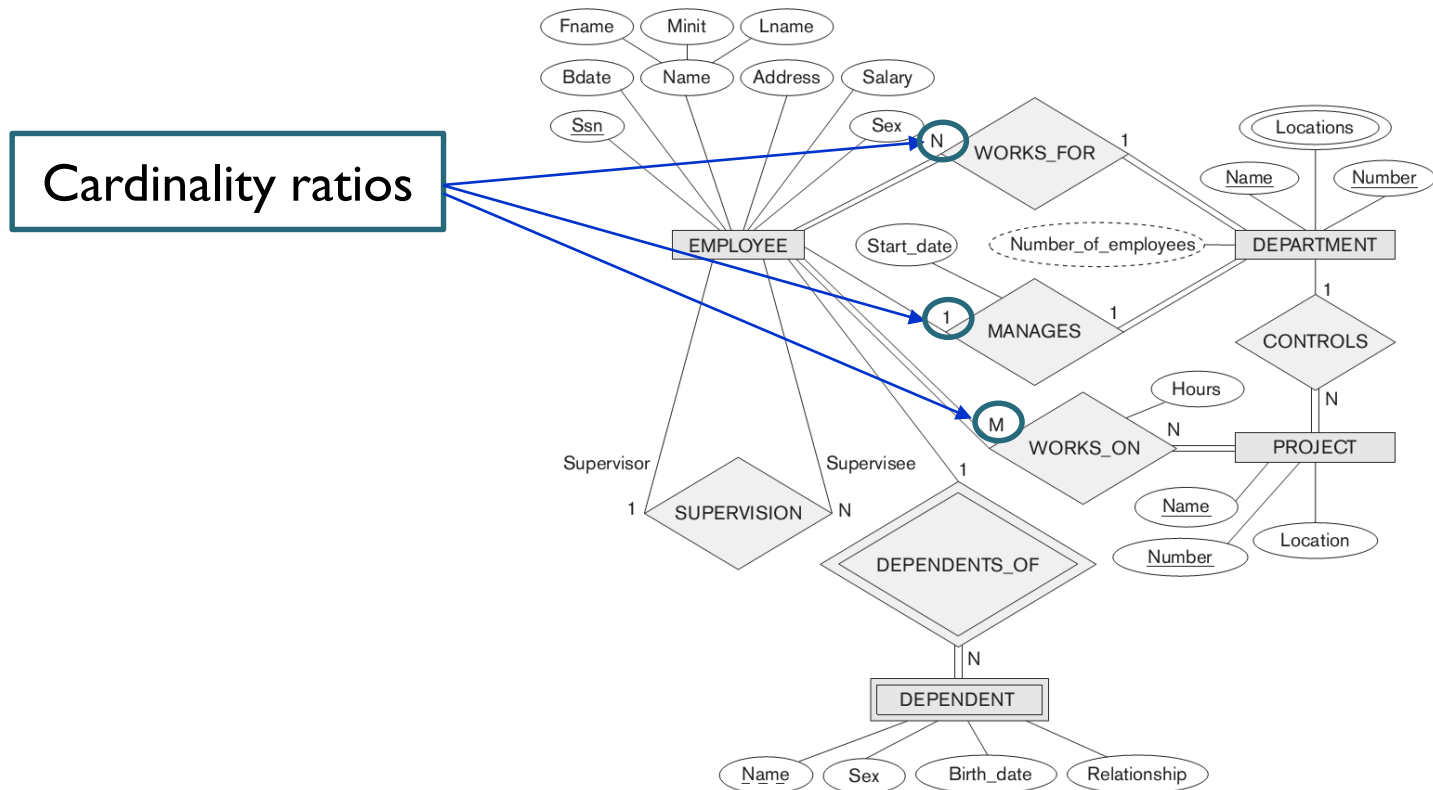


# Cardinality ratios

- Cardinality ratio for a binary relationship specifies the maximum number of relationship instances that an entity can participate in.
  - Example, in WORKS\_FOR relationship, DEPARTMENT:EMPLOYEE is of cardinality ratio 1:N, meaning that each department can be related to any number of employees but an employee can be related to at most one department.
- The possible cardinality ratios for binary relationship types are 1:1, 1:N, N:1 and M:N.
  - Example, 1:1 – MANAGES – relating a department entity to the employee who manages that department. This means that an employee can manage at most one department and a department can have at most one manager.

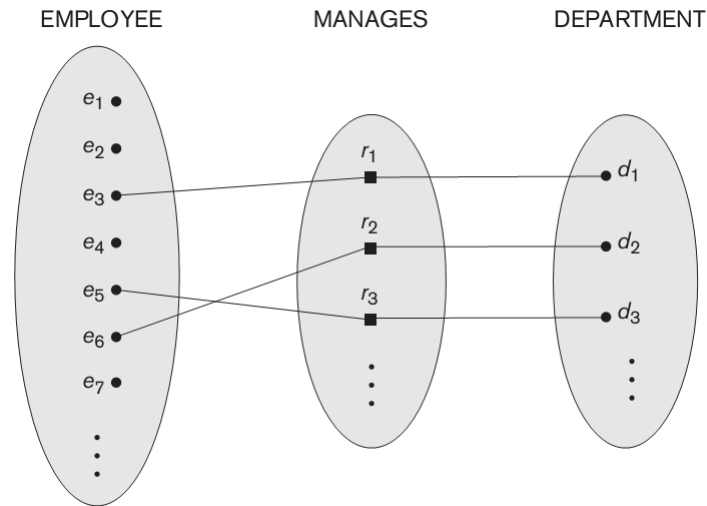
# Cardinality ratios

- Example, M:N – WORKS\_ON is of cardinality ratio M:N because an employee can work on several projects and a project can have several employees.



# Participation constraints and existence dependencies

- The participation constraint specifies whether the existence of an entity depends on its being related to another entity through the relationship type.
- This constraint specifies the minimum number of relationship instances that each entity can participate in – called “minimum cardinality constraint”.
- Two types of participating constraints – total (existence dependency) and partial.
  - Example, if every employee must work for a department, then an employee entity can exist only if it participates in at least one WORKS\_FOR relationship – therefore, EMPLOYEE in WORKS\_FOR is a total participation.



A 1:1 relationship, MANAGES.

- In figure above, every employee may not manage a department, so the participation of EMPLOYEE in the MANAGES relationship is partial.
  - It means that some or part of the set of employee entities are related to some department entity via MANAGES, but not necessarily all.
- Together, cardinality ratio and participation constraints are called “structural constraints” of a relationship.

# Attributes of relationship types

- Relationship types can also have attributes, similar to those of entity types.
  - Example, when an employee works on a particular project, to record the “number of hours per week”, we can have an attribute “Hours” for the WORKS\_ON relationship type.
- Attributes of 1:1 or 1:N relationship types can be migrated to one of the participating entity types.
  - Example, Start\_date attribute for the MANAGES relationship can be an attribute of either EMPLOYEE or DEPARTMENT although it belongs to MANAGES relationship.
  - So, value of Start\_date can be determined by participating department entity or by the participating employee (manager) entity.

# Attributes of relationship types

- For 1:N relationship, a relationship attribute can be migrated only to the entity type on the N-side of the relationship.
  - Example, each employee WORKS\_FOR at most one department, but a department can have many employees each with a different Start\_date.
  - So, in WORKS\_FOR, Start\_date can be included as an attribute of EMPLOYEE.
- However, in both 1:1 and 1:N, the decision where to place a relationship attribute – as a relationship type attribute or as an attribute of the participating entity type is subjective.



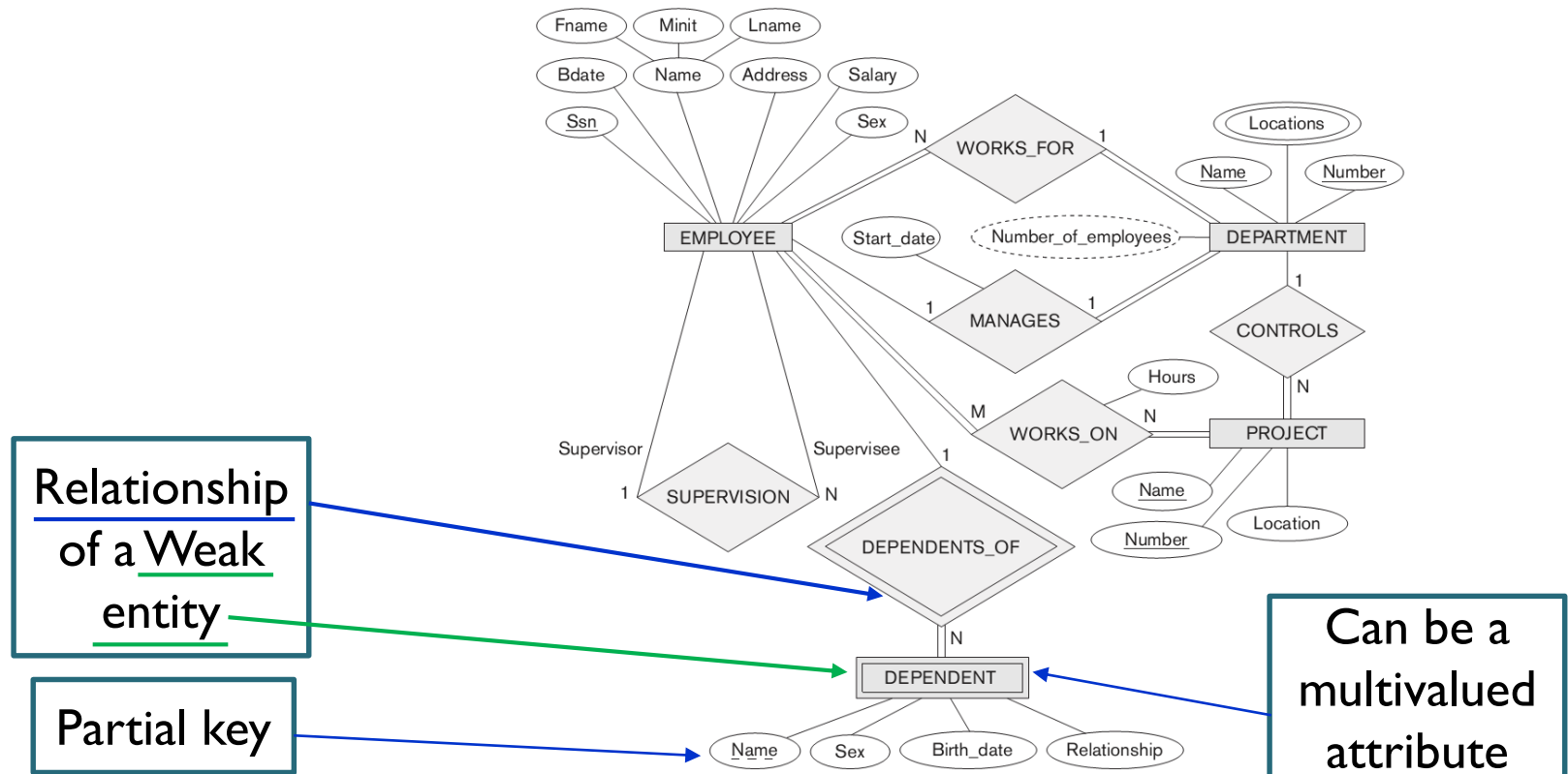
- For M:N (many-to-many) relationship types, some attributes can be determined by a combination of participating entities in a relationship.
  - Example, “Hours” attribute (the number of hours per week an employee works on a project) of the M:N relationship WORKS\_ON can be determined by an “employee-project” combination and not separately by either entity.

# Weak entity

- Entity that do not have “key attributes” of their own are called weak entity types.
- Regular entities that do have a “key attribute” are called strong entity types.
- Weak entities are identified by being related to specific entities from another entity type in combination with one of their attribute value.
  - These are called the identifying or (owner/parent/dominant) entity type and the relationship type that relates a weak entity type to its owner is called **identifying relationship** of the weak (child/subordinate) entity type.

- A weak entity type always has a total participation constraint (existence dependency) with respect to its identifying relationship because a weak entity cannot be identified without an owner entity.
  - Exceptions are for example, DRIVER\_LICENSE entity which cannot exist unless it is related to a PERSON entity, even though it has its own key (License\_number) and hence it is not a weak entity.
- Consider example, DEPENDENT (with attributes: First\_name, Birth\_date, Sex and Relationship) related to EMPLOYEE.
  - If two dependents of two employees have the same values for all the 4 attributes, then they are identified as distinct entities only after determining the associated employee's entity to which the dependent is related.
  - Therefore, each employ entity “owns” the dependent entities that are related to it.

- A weak entity type generally has a “partial key/discriminator”, which is the attribute that can uniquely identify weak entities that are related to the same owner entity. Example, “Name” of the DEPENDENT.



- Weak entity types can also be represented as complex (composite, multivalued) attributes or may have more than one identifying entity type.

# Refining the ER design for the COMPANY database

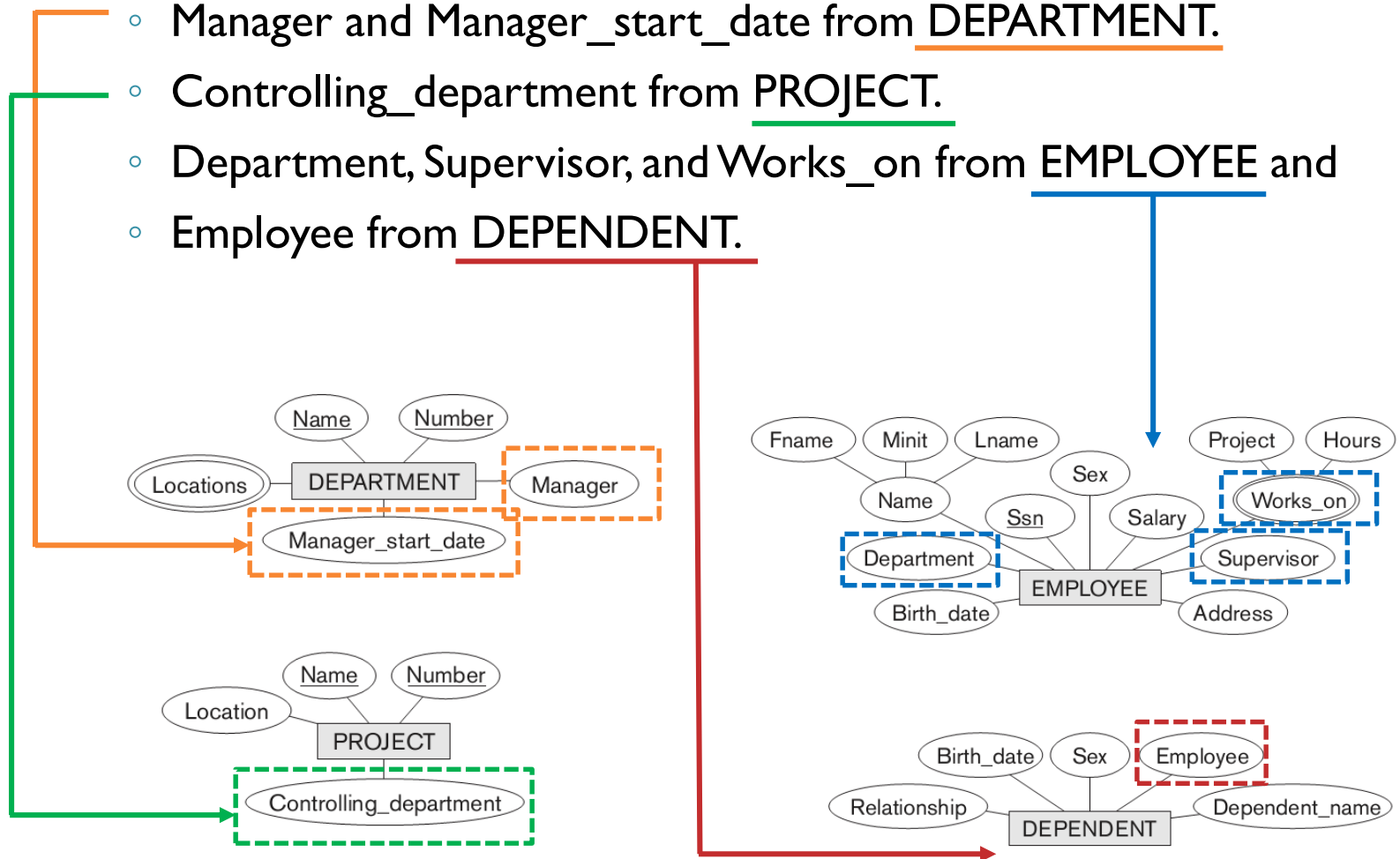
- Now, let's change the relationship attribute into “relationship types”.
- The structural constraints (cardinality ratio and participation constraint) of each relationship type are determined from the requirements.
- Summary of the six identified relationship types:
  1. MANAGES is a 1:1 relationship between EMPLOYEE and DEPARTMENT. Here, EMPLOYEE participation is partial. A DEPARTMENT must have a manager – implies total participation. Attribute “Start\_date” can be assigned to this relationship.
  2. WORKS\_FOR is a 1:N relationship between DEPARTMENT and EMPLOYEE. Both participants are total.

3. CONTROLS is a 1:N relationship between DEPARTMENT and PROJECT. The participation of PROJECT is total whereas DEPARTMENT is partial (Some departments may control no projects).
4. SUPERVISION is a 1:N relationship between EMPLOYEE (supervisor) and EMPLOYEE (supervisee). Both are partial as not every employee is a supervisor and not every employee has a supervisor (may be CEO of a Company).
5. WORKS\_ON is a M:N (many-to-many) relationship with attribute “Hours”. Both participants are total.
6. DEPENDENTS\_OF is a 1:N relationship between EMPLOYEE and DEPENDENT, which is also the identifying relationship for the weak entity type DEPENDENT. The participation of EMPLOYEE is partial and DEPENDENT is total.



- So, now we can remove all “attributes” (refer figure below) that have been refined into relationships such as

- Manager and Manager\_start\_date from DEPARTMENT.
- Controlling\_department from PROJECT.
- Department, Supervisor, and Works\_on from EMPLOYEE and
- Employee from DEPENDENT.



# True or False

In ER diagrams the focus is on representing the schemas rather than the instances because a schema does not change often, whereas the contents of the entity may change frequently.


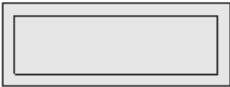
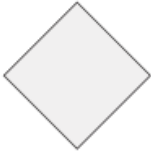
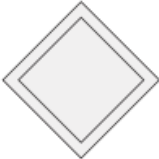


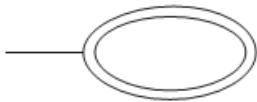
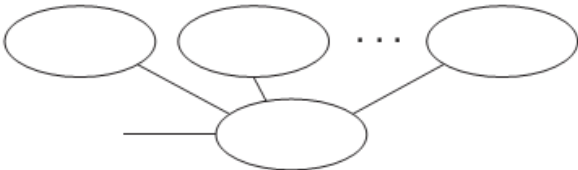
True

# Naming conventions and design issues

- Let us review the full ER diagram notations:
  - ❑ Regular (strong) entity types (EMPLOYEE, DEPARTMENT and PROJECT) are shown in rectangular boxes.
  - ❑ Relationship types such as WORKS\_FOR, MANAGES, CONTROLS and WORKS\_ON are depicted in diamond-shaped boxes attached to the participating entity types with straight lines.
  - ❑ Attributes are shown in ovals, and each attribute is attached by a straight line to its type or relationship type.

- ❑ Components of a composite attribute are attached to the oval as shown in Name attribute of EMPLOYEE.
- ❑ Multivalued attributes are shown in double ovals (example, Locations attribute of DEPARTMENT).
- ❑ Key attributes have their names underlined.
- ❑ Derived attributes are shown in dotted ovals (example, Number\_of\_employees attribute of DEPARTMENT).
- ❑ Weak entity types are distinguished by being placed in double rectangles and by having their identifying relationship placed in double diamonds (example, DEPENDENT entity type and the DEPENDENTS\_OF identifying relationship type).
- ❑ The partial key of the weak entity type is underlined with a dotted line.

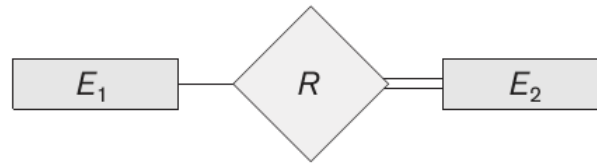
- ❑ The cardinality ratio is specified by attaching a 1, M, or N on each participating edge.
  - Example, 1:1 (DEPARTMENT:EMPLOYEE in MANAGES), 1:N (for DEPARTMENT:EMPLOYEE in WORKS\_FOR), and M:N (for WORKS\_ON).
- ❑ The participation constraint is specified by a single line for partial participation and by double line for total participation (existence dependency).

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identifying Relationship
	Attribute
	Key Attribute
	Multivalued Attribute
	Composite Attribute

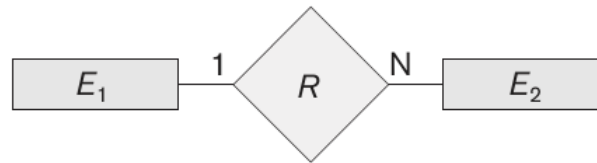




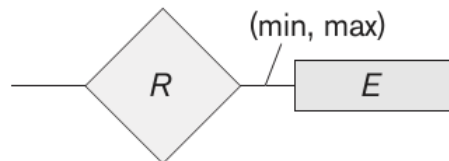
Derived Attribute



Total Participation of  $E_2$  in  $R$



Cardinality Ratio 1 : N for  $E_1 : E_2$  in  $R$

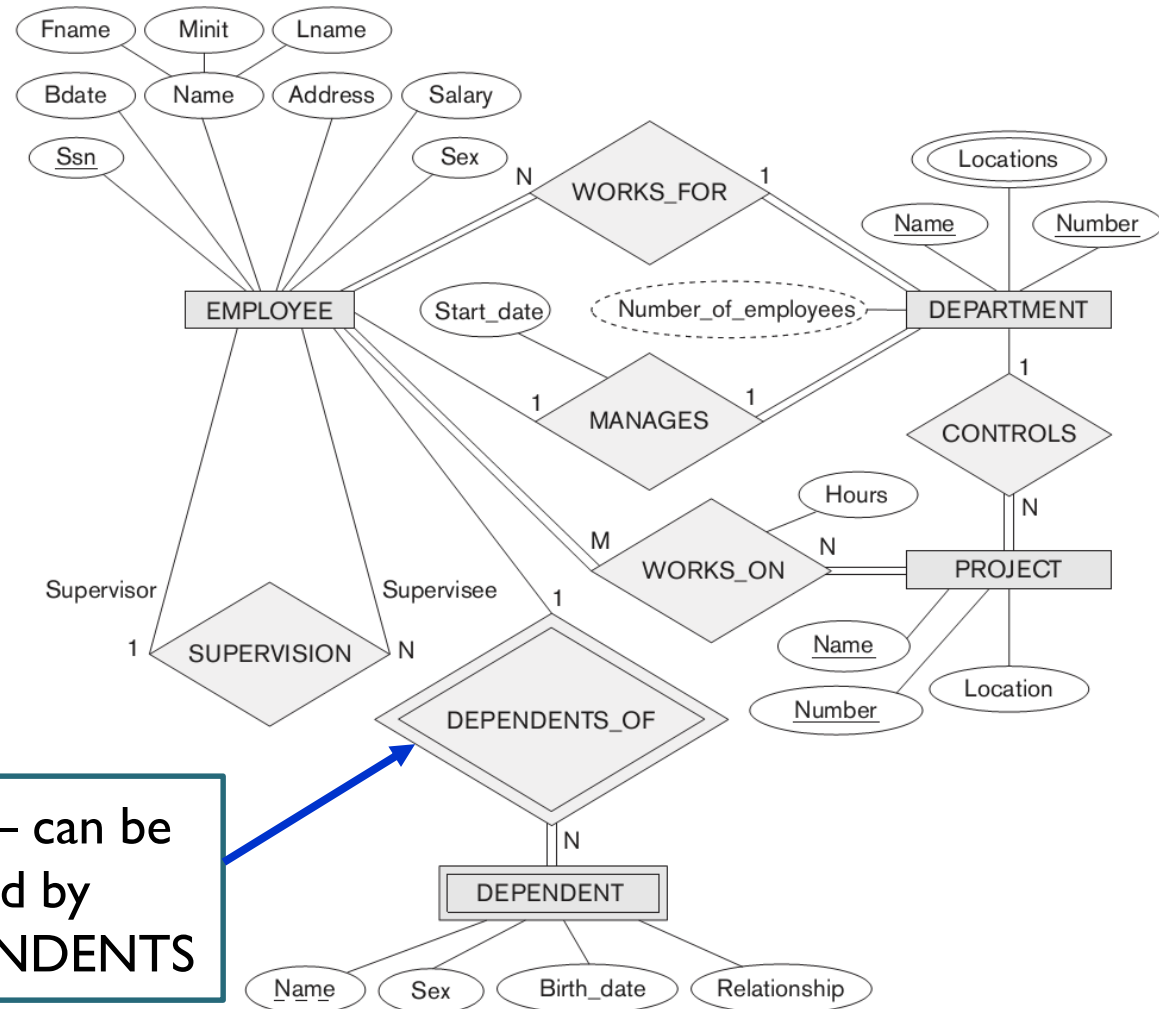


Structural Constraint (min, max)  
on Participation of  $E$  in  $R$

# Naming of schema constructs

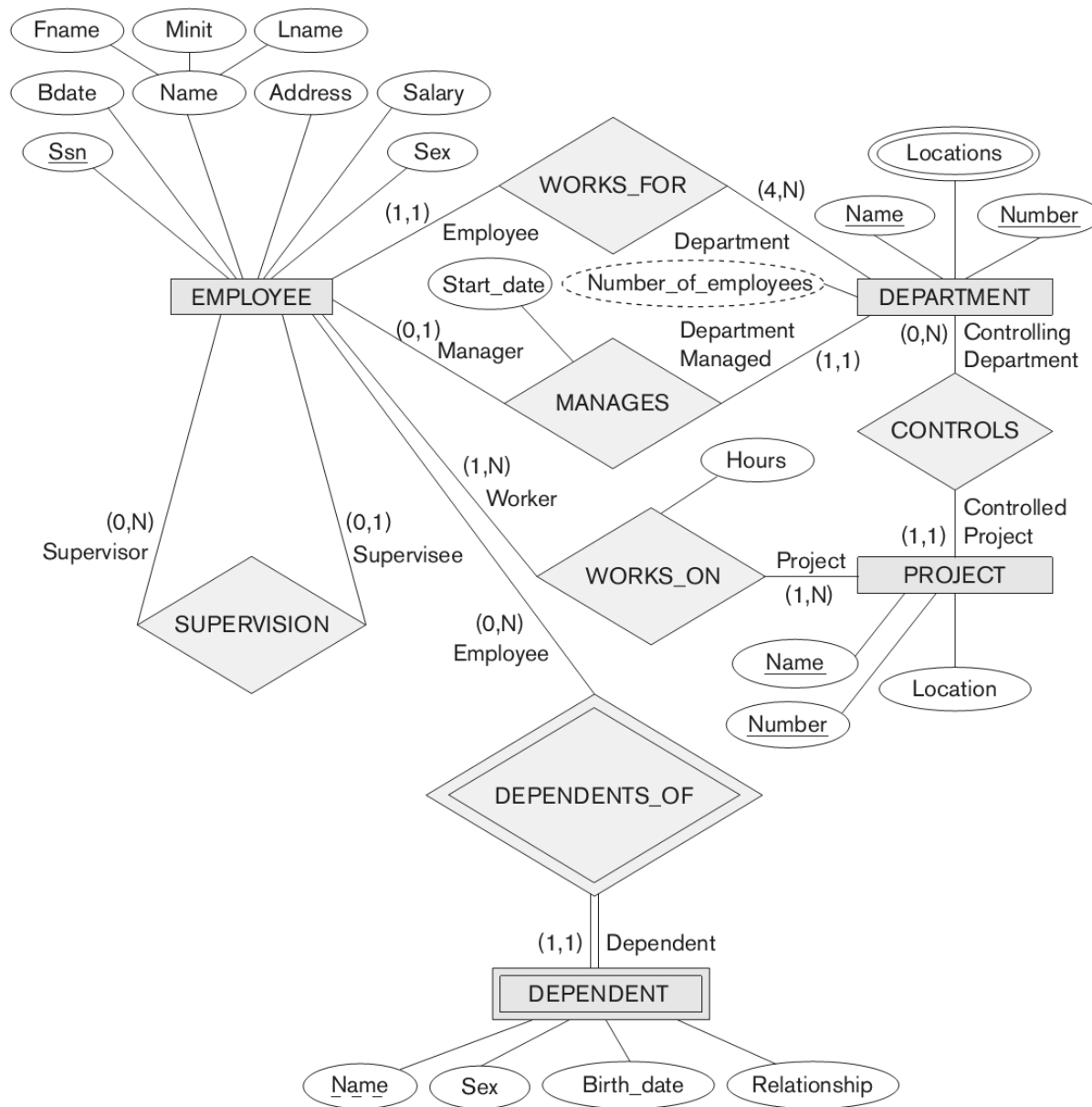
- Important → When designing a DB schema, the choice of names for entity, attributes, relationship types, and roles are not always straightforward.
- The names should be chosen in a way that convey, as much as possible, the meanings attached to the different constructs in the schema.
- Chose singular names for entity types.
- Entity type and relationship type names are in uppercase letters, attributes have their initial letter capitalised and role names are in lowercase letters.
- From the DB requirements description, nouns are entity types and verbs indicate relationships. Attribute names arise from additional nouns that describe the nouns corresponding to entity types.

- Binary relationships names of the schema are read from left to right and top to bottom.



# Design choices for ER conceptual design

- It is sometimes difficult to decide whether a particular concept should be modelled as an entity, an attribute or a relationship type.
- It is always good to refine the schema iteratively until most suitable design is reached. Some useful rules:
  - ✓ A concept may be first modelled as an attribute and then refined into a relationship because it is determined that the attribute is a reference to another entity type.
  - ✓ A pair of attributes that are inverses of one another are refined into a binary relationship.
  - ✓ The attribute that exist in several entity types can be made a “independent entity type” or vice-versa. Example, create a new entity DEPARTMENT with attribute Dept\_name and relate it to STUDENT, INSTRUCTOR and COURSE through appropriate relationships.



ER diagram for the company schema.

# UML class diagrams

- Class diagrams are an alternate to ER diagrams.
  - Here, entity types are modelled as classes and an entity in ER corresponds to an object in UML.
- 
- A class is displayed as a box with three sections: *class name* (similar to entity name), *attributes* and *operations* that can be applied to individual objects of the class (*not specified* in ER diagrams).
    - The operations given in each class are derived from the functional requirements of the application.
  - A composite attribute is modelled as a structured domain (example, Name attribute of EMPLOYEE).
  - A multivalued attribute is modelled as a separate class (example, LOCATION class).



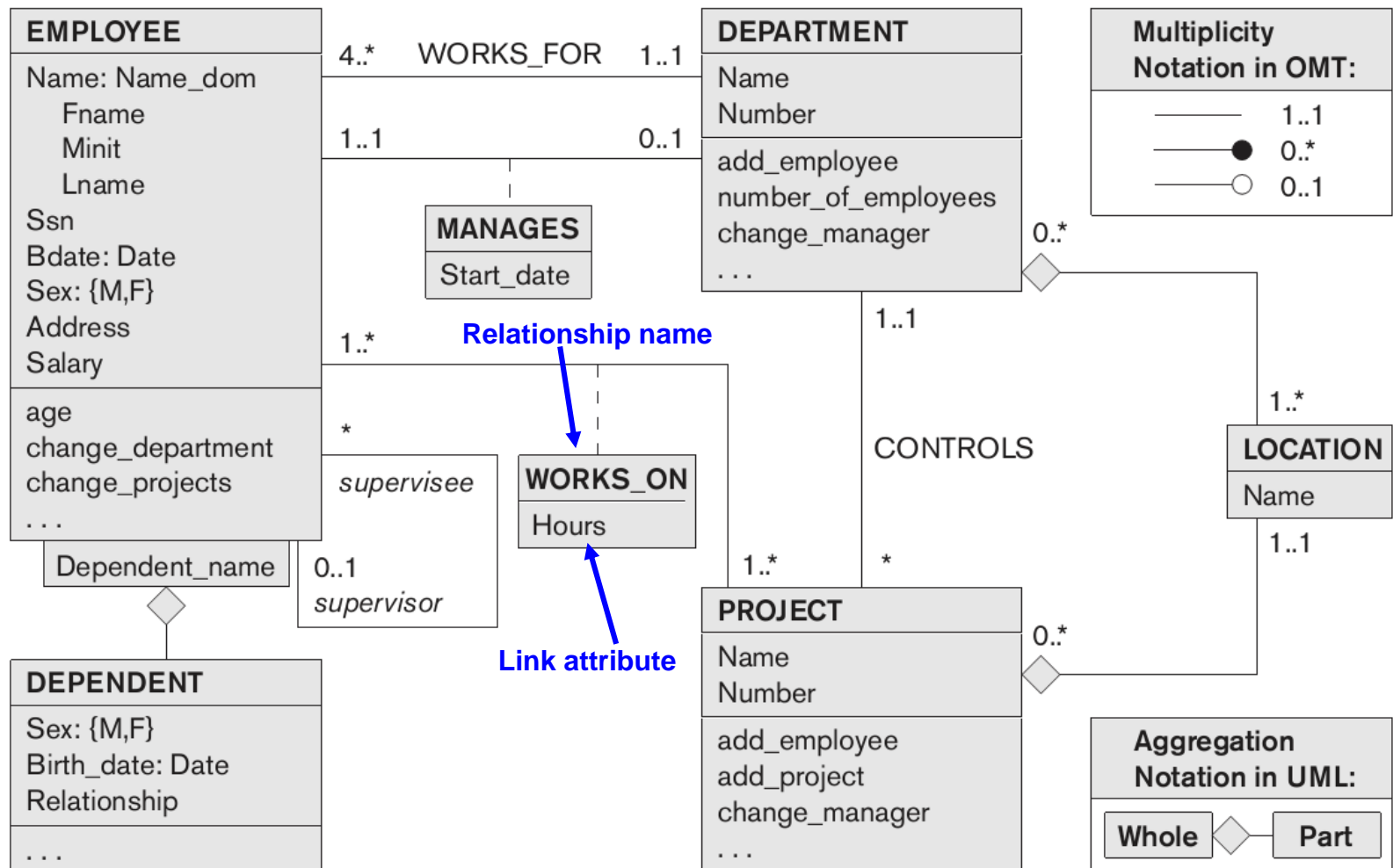


Figure: COMPANY conceptual schema in UML class diagram notation.

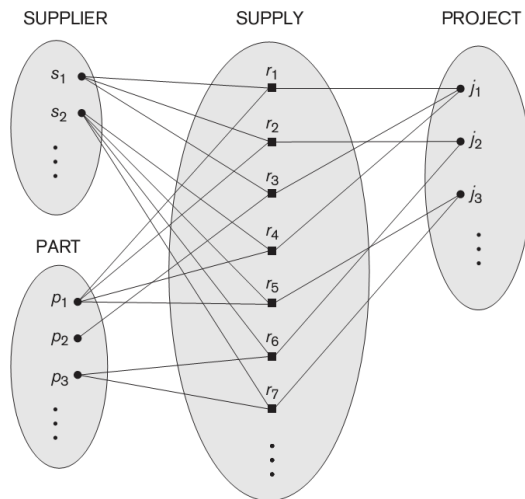
- Relationship types are called “associations” in UML and relationship instances are called “links”.
- A binary association is represented as a line connecting the participating classes (entity types) and may have a name.
- A relationship attribute, called a *link attribute* is placed in a box that is connected to the association’s line by a dashed line.
- A recursive relationship is called a reflexive association in UML.

- In UML, there are two types of relationships: association and aggregation.
  - Aggregation is meant to represent a relationship between a whole object and its component parts, example: locations of a department and the single location of a project as aggregations.
  - They do not have different structural properties and the choice of their usage is subjective.
- UML distinguishes between unidirectional (displayed as an arrow) and bidirectional (no arrow – this is the default) associations (or aggregations).

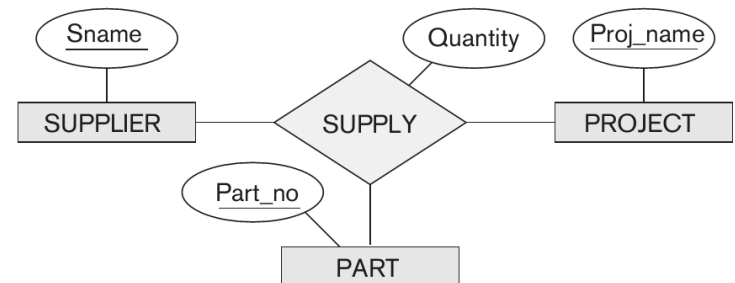
- Association (relationship) names are optional in UML.
- Weak entities are modelled using the qualified association (or qualified aggregation) representing both the identifying relationship and the partial key, which is placed in a box attached to the owner class. Example, DEPENDENT class and its qualified aggregation to EMPLOYEE.
- In UML, the partial key attribute (such as Dependent\_name) is called the *discriminator*.

# Higher degree relationship

1. When to choose higher degree versus binary relationships?
  2. How to specify constraints on higher-degree relationships?
- Choosing between binary and ternary (or higher-degree) relationships:



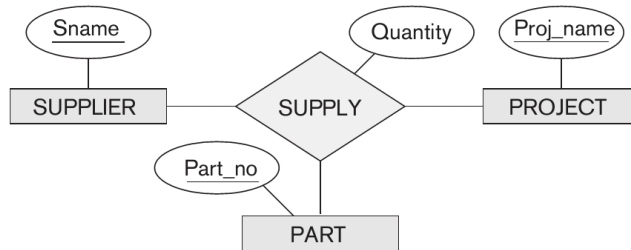
Here, each relationship instance  $r_i$  associates three entities – a supplier  $s$ , a part  $p$ , and a project  $j$  – whenever  $s$  supplies part  $p$  to project  $j$ .



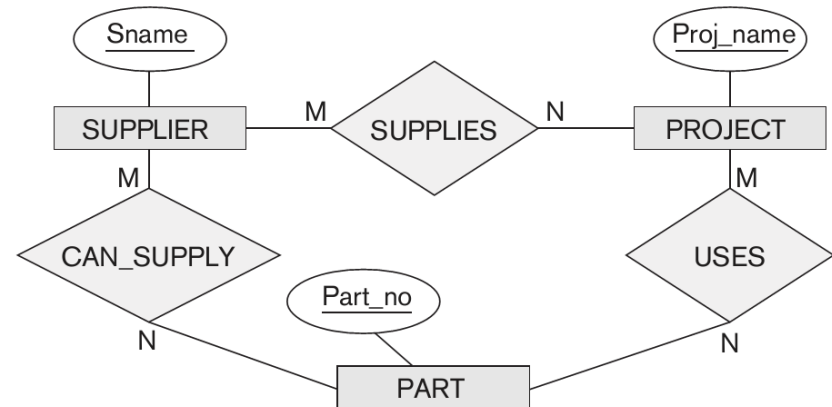
ER diagram for a ternary relationship: SUPPLY.

- In general, a relationship type  $R$  of degree  $n$  will have  $n$  edges in an ER diagram, one connecting  $R$  to each participating entity type.

- A ternary relationship type represents different information than do three binary relationship types.



ER diagram for a ternary relationship: SUPPLY.



ER diagram for three binary relationship types CAN\_SUPPLY, USES and SUPPLIES.

- Suppose that **CAN\_SUPPLY**, between **SUPPLIER** and **PART**, includes an instance  $(s, p)$  whenever supplier  $s$  can supply part  $p$  to any project; **USES**, between **PROJECT** and **PART**, includes an instance  $(j, p)$  whenever project  $j$  uses part  $p$ ; and **SUPPLIES**, between **SUPPLIER** and **PROJECT**, includes an instance  $(s, j)$  whenever supplier  $s$  supplies some part to project  $j$ .



- Important – The existence of three relationship instances  $(s, p)$ ,  $(j, p)$ , and  $(s, j)$  in `CAN_SUPPLY`, `USES`, and `SUPPLIES` does not necessarily imply that an instance  $(s, j, p)$  exists in the ternary relationship `SUPPLY` because they have a different meaning.
- So, it is tricky to decide whether a particular relationship should be of degree  $n$  or several relationship types of smaller degrees.

# Another example

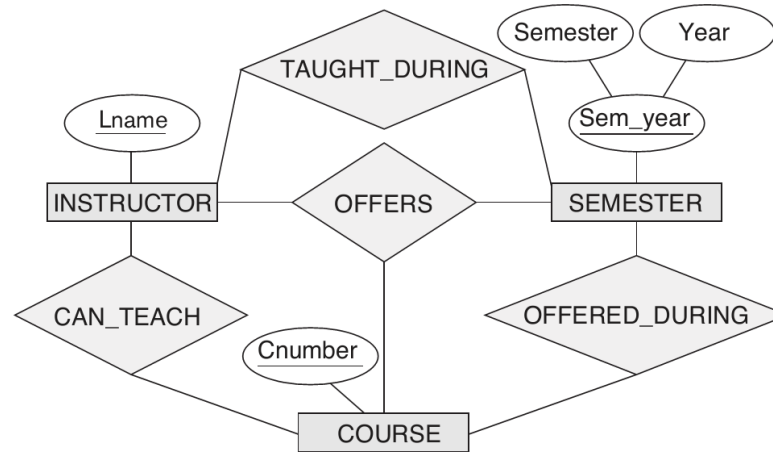
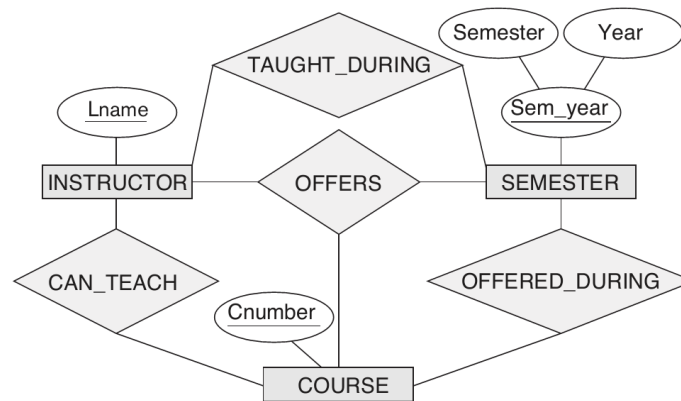
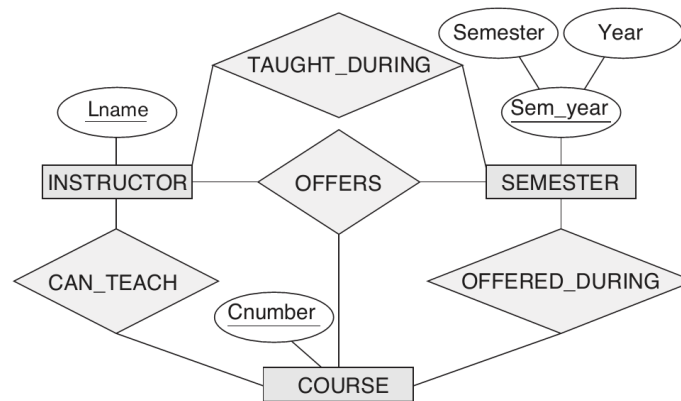


Figure: Example of ternary versus binary relationship types.

- The ternary relationship type **OFFERS** represents information on instructors offering courses during particular semesters, hence it includes a relationship instance  $(i, s, c)$  whenever INSTRUCTOR  $i$  offers COURSE  $c$  during SEMESTER  $s$ .



- CAN\_TEACH – relates a course to the instructors who can teach that course.
- TAUGHT\_DURING – relates a semester to the instructors who can teach that course.
- OFFERED\_DURING – relates a semester to the courses offered during that semester by any instructor.



- These binary and ternary relationships represent different information, but certain constraints should hold among the relationships.
- Example, the relationship instance  $(i, s, c)$  should not exist in OFFERS unless an instance  $(i, s)$  exist in TAUGHT\_DURING, an instance  $(s, c)$  exists in OFFERED\_DURING, and an instance  $(i, c)$  exists in CAN\_TEACH.
  - However, the reverse is not always true – we may have instances  $(i, s)$ ,  $(s, c)$  and  $(i, c)$  in the three binary relationship with no corresponding instance  $(i, s, c)$  in OFFERS.

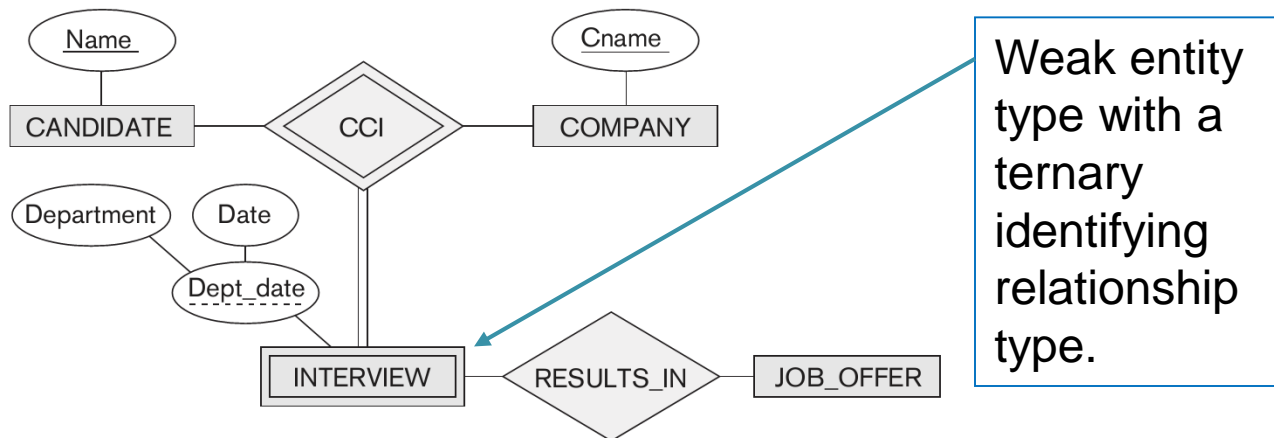
- Based on the meanings of the relationships, we can now infer the instances of TAUGHT\_DURING and OFFERED\_DURING from the instances in OFFERS, but we cannot infer the instances of CAN\_TEACH, therefore, TAUGHT\_DURING and OFFERED\_DURING can be left out.

- Sometimes, three binary relationships can replace a ternary relationship (with some additional constraints).
- Example, if the CAN\_TEACH relationship is 1:1 (an instructor can teach only one course, and a course can be taught by only one instructor), then the ternary relationship OFFERS can be left out because it can be inferred from the three binary relationships CAN\_TEACH, TAUGHT\_DURING and OFFERED\_DURING.

- It is also possible to have a weak entity type with a ternary identifying relationship type.
- So, a weak entity type will have several owner entity types.


### Example:

- Consider part of a DB that keeps track of candidates interviewing for jobs at various companies, which may be part of an employment agency database.



Weak entity type with a ternary identifying relationship type.



- 
- In the requirements, a candidate can have multiple interviews with the same company (different departments on separate dates), but the job offer is made based on one of the interviews.
  - Here, INTERVIEW is a weak entity with two owners – CANDIDATE and COMPANY with the partial key “Dept\_date”.
  - An INTERVIEW is uniquely identified by a candidate, a company, and combination of date and department of the interview.

1. When to choose higher degree versus binary relationships?
  2. How to specify constraints on higher-degree relationships?
- Constraints on ternary (or higher-degree) relationships:
  - Two notations exist for specifying different structural constraints on  $n$ -ary relationships, and should be used together on ternary or higher degrees.
  - The first notation is based on the cardinality ratio of binary relationships, where 1, M or N can be specified on each participation arc.
    - Example, SUPPLY relationship instances ( $s$  - *supplier*,  $j$  - *project*,  $p$  - *part*). Suppose the constraint is – “only one supplier supplies a particular part to a particular project”. So we place 1 on the SUPPLIER participation, and M, N on the PROJECT and PART.

- This specifies the constraint that a particular  $(j, p)$  combination can appear at most once in the relationship set because each such (PROJECT, PART) combination uniquely determines a single supplier.

- Hence, any relationship instance  $(s, j, p)$  is uniquely identified in the relationship set by its  $(j, p)$  combination, which makes  $(j, p)$  a “key” for the relationship set.

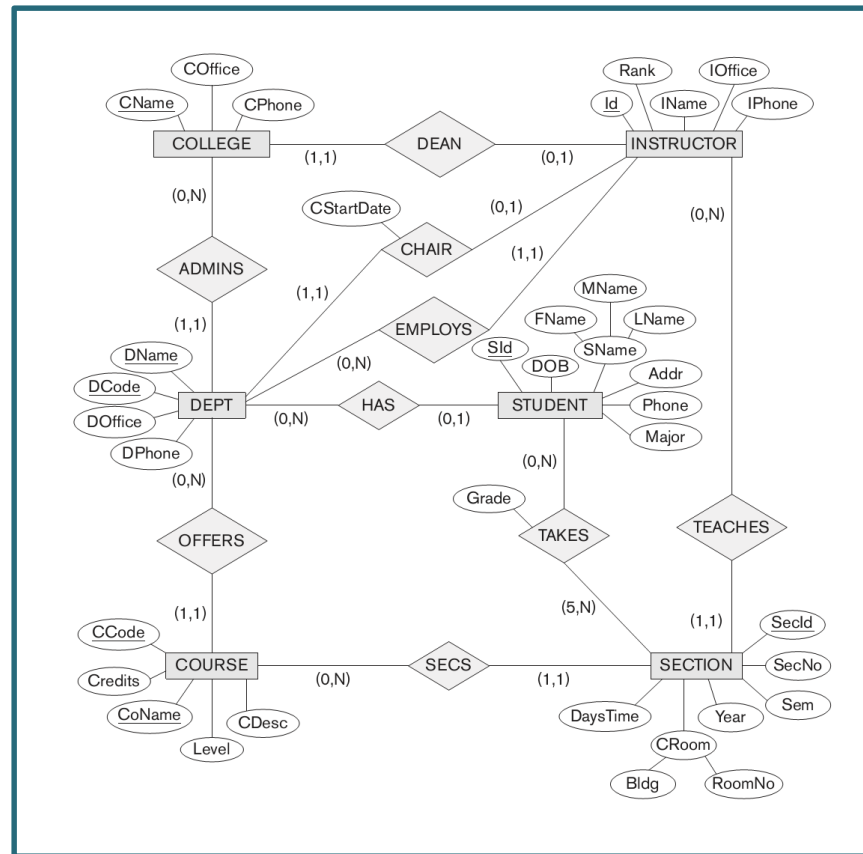
# True or False

If all the cardinalities are M or N, then the key will be the combination of all three participants.

True

- The second notation is based on the  $(min, max)$  which specifies that each entity is related to at least *minimum* and at most *maximum* relationship instances. These constraints place restrictions on how many relationship instances each entity can participate in.

# Final Example of a University DB






- A DB is designed to keep track of student enrollments in classes and their grades.
- Design specifications:

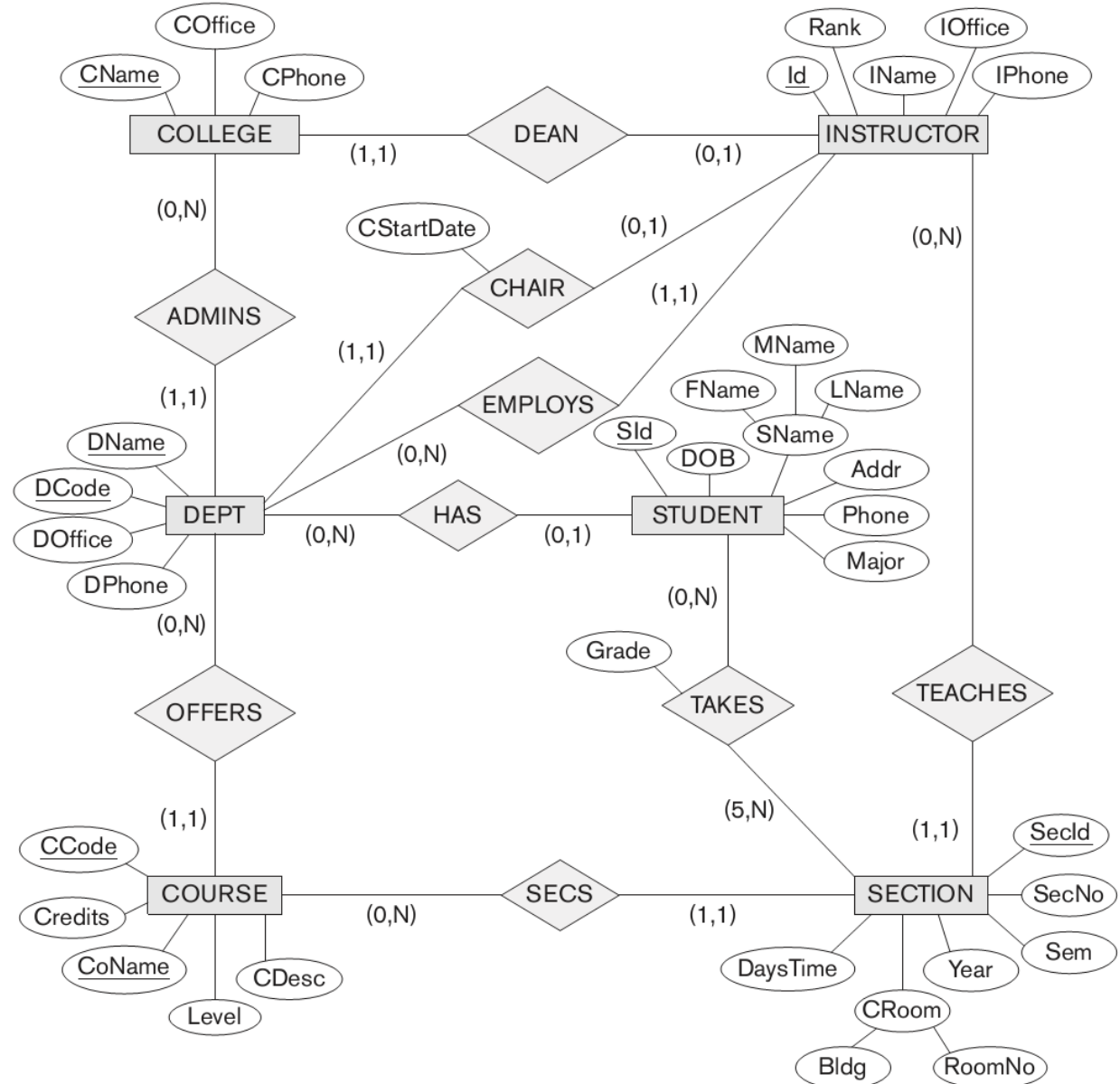
Note: Entity type names are in parentheses ().

1. The university is organised into colleges (COLLEGE) with its unique name (CName), a main office (COffice) and phone (CPhone) and a particular faculty member is the dean.
2. Each college has a number of academic departments (DEPT), each department has a unique name (DName), a unique code (DCode), a main office (DOffice) and phone (DPhone), and a Faculty member chairs a department on a start date (CStartDate).
3. A department offers a number of courses (COURSE) with a course name (CoName), a code number (CCode), a course level (Level: 1 (for freshman), 2 (sophomore), 3 (junior), 4 (senior), 5 (MS level), 6 (Ph.D. level)), credit hours (Credits) and course description (CDesc).

- 
4. There are instructors (INSTRUCTOR) with a identifier (Id), name (IName), office (IOffice), phone (IPhone) and rank (Rank). Each instructor works for one primary department.
  5. A DB will keep student data (STUDENT), name (SName), id (Sid), address (Addr), phone (Phone), major code (Major) and date of birth (DoB).
  6. Courses are offered as sections (SECTION). Each section is related to a single course and a single instructor, and has a unique section identifier (SecId) with a section number (SecNo: 1, 2, 3 for multiple sections offered during the same semester/year), semester (SEM), year (Year), classroom (CRoom), building code (Bldg) and room number (RoomNo), days/times to allow classes at the specified time (a constraint).
  7. A section must have at least 5 students.

- The following combinations are unique:
  1. (SecNo, Sem, Year, CCode): This specifies that the section numbers of a particular course must be different during each particular semester and year.
  2. (Sem, Year, CRoom, DaysTime): This specifies that in a particular semester and year, a classroom cannot be used by two different sections at the same days/time.
  3. (Sem, Year, DaysTime, Id of the INSTRUCTOR teaching the SECTION): This specifies that in a particular semester and year, an instructor cannot teach two sections at the same days/time. This rule will not apply if an instructor is allowed to teach two combined sections together in a particular university.

# ER diagram: A University DB



# Summary

- High level conceptual data model i.e. the entity-relationship (ER) model were discussed through COMPANY DB.
- Defined entities, attributes (simple or atomic, composite, multivalued, stored and derived), and NULL values.
- Discussed entity types, entity sets, key attributes, value sets (domain) of attributes, relationship types and relationship sets, participation roles of entity types.
- Discussed structural constraints – cardinality ratios (1:1, 1:M, 1:N) and participation constraints (total and partial).
- Another method of specifying structural constraints is to specify minimum and maximum numbers (min, max) on the participation of each entity type in a relationship.

# Summary

- Weak entity, owner entity types, partial key attributes.
- We saw the design of ER schema with entity, attributes and relationships types.
- Basic concepts of UML class diagram.
- Ternary and higher degree relationships.
- Final example of a University Database.





# Practice questions (check yourself)

1. Define the following: entity, relationship and attribute.
2. What are the different types of attributes?
3. What is a key attribute?
4. Discuss the role of NULL values in database.
5. What is degree of a relation? In this context, explain recursive relationships.
6. Discuss cardinality ratios and participation constraints.