Database Design

More on Relational Model

(And EER model)

NULL (1)

- What if an instance doesn't have a value?
  Examples:
    - Marriage date of a person is not known (True value is not yet known)
    - Student may have not had a pet (No suitable value)
- Exercise: List more cases where use of a NULL value would be appropriate.
- NULL values represent attributes whose values are unknown or do not exist for some entity instance. In general, it is a special value to indicate a lack of value.

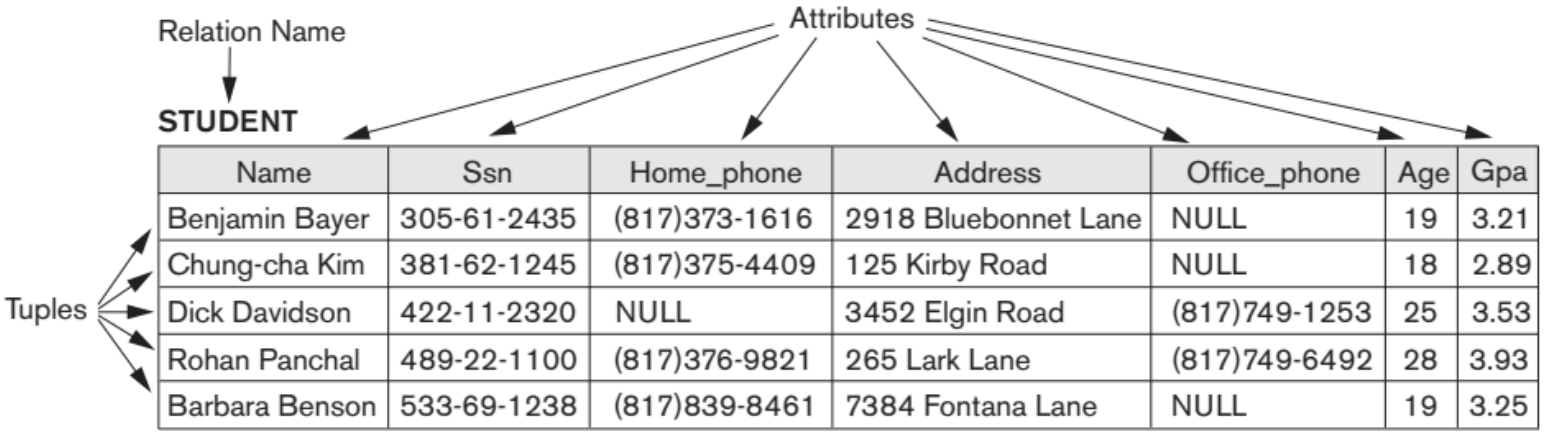
NULL (2)

Null: Example Usage: COMP9311 Student Schema:

- (zID, trimester id, ass1-mark, ass2-mark, exam-mark)

COMP9311 Student Instance/Fact

- 21T2 = (z0000001, 21T2, 16/25, 18/25, 50/50)
- What about someone from this 22T1? use NULL and update it later?



Name	Ssn	Home_phone	Address	Office_phone	Age	Gpa
Benjamin Bayer	305-61-2435	(817)373-1616	2918 Bluebonnet Lane	NULL	19	3.21
Chung-cha Kim	381-62-1245	(817)375-4409	125 Kirby Road	NULL	18	2.89
Dick Davidson	422-11-2320	NULL	3452 Elgin Road	(817)749-1253	25	3.53
Rohan Panchal	489-22-1100	(817)376-9821	265 Lark Lane	(817)749-6492	28	3.93
Barbara Benson	533-69-1238	(817)839-8461	7384 Fontana Lane	NULL	19	3.25

Recap: Referring to Another Relation

- ***Foreign key****: an attribute that keeps the value of a primary key of* another relation.
- A set of attributes from a relation schema R1 may be a foreign *key*, *FK*, if
  - the attributes have ***the same domains*** as the attributes in the primary key of another relation schema R_2, and
  - a value of FK in a tuple t_1 of R_1 either occurs as a value of PK for some tuple t_2 in R_2 or is null.
- *Closely related to the referential integrity*: The value of *FK* must occur in the other relation or be entirely NULL. (more on this later)

Relational Integrity Constraints

- We need to keep the relational database in a ***valid state***:

- Three integrity constraints are important
  1. **Key constraint**: candidate key values must be unique for every relation instance.
  2. **Entity integrity**: an attribute that is part of a primary key cannot be NULL.
  3. **Referential integrity**: closely related to foreign keys

- **Valid state**: a relation does not violate any integrity constraints.
- **Invalid state**: a relation violates at least one integrity constraint

Relational Integrity Constraints

- Q: How can a valid relation can it ever become invalid?
  A: Operations on the database can result in an invalid state.

- Before proceeding with an ***update***, we need to…
  - check that the result of the update will not be violate any integrity constraints.

Insertion: Key constraint violation

STUDENT:

Person#	Name
1	Dr C.C.Chen
3	Ms K.Juliff
4	Ms J.Gledill
5	Ms B.K.Lee

RESEARCHER:

Person #	Name
1	Dr C.C.Chen
2	Dr R.G.Wilkinson

COURSE:

Department	Degree
Psychology	Ph.D.
Comp.Sci.	Ph.D.
Comp.Sci.	M.Sc.
Psychology	M.Sc.

ENROLMENT:

Enrolment#	Supervisee	Supervisor	Department	Degree
1	1	2	Psychology	Ph.D.
2	3	1	Comp.Sci.	Ph.D.
3	4	1	Comp.Sci.	M.Sc.
4	5	1	Comp.Sci.	M.Sc.

1. Insert < 2, *Dr.V.Ciesielski* > into RESEARCHER Allowed?

Insertion: Entity integrity violation

STUDENT:

Person#	Name
1	Dr C.C.Chen
3	Ms K.Juliff
4	Ms J.Gledill
5	Ms B.K.Lee

RESEARCHER:

Person#	Name
1	Dr C.C.Chen
2	Dr R.G.Wilkinson

COURSE:

Department	Degree
Psychology	Ph.D.
Comp.Sci.	Ph.D.
Comp.Sci.	M.Sc.
Psychology	M.Sc.

ENROLMENT:

Enrolment#	Supervisee	Supervisor	Department	Degree
1	1	2	Psychology	Ph.D.
2	3	1	Comp.Sci.	Ph.D.
3	4	1	Comp.Sci.	M.Sc.
4	5	1	Comp.Sci.	M.Sc.

2. Insert < *Comp.Sci.,NULL* > into COURSE Allowed?

Insertion: Referential integrity violation

STUDENT:

Person#	Name
1	Dr C.C.Chen
3	Ms K.Juliff
4	Ms J.Gledill
5	Ms B.K.Lee

RESEARCHER:

Person#	Name
1	Dr C.C.Chen
2	Dr R.G.Wilkinson

COURSE:

Department	Degree
Psychology	Ph.D.
Comp.Sci.	Ph.D.
Comp.Sci.	M.Sc.
Psychology	M.Sc.

ENROLMENT:

Enrolment#	Supervisee	Supervisor	Department	Degree
1	1	2	Psychology	Ph.D.
2	3	1	Comp.Sci.	Ph.D.
3	4	1	Comp.Sci.	M.Sc.
4	5	1	Comp.Sci.	M.Sc.

3. Insert < 5, 6, 2, *Psychology*, *Ph.D.* > into ENROLMENT Allowed?

Summary on Insertion

- How can insertions cause an invalid state?
- To avoid an invalid state…
  - Check that the candidate keys are not already present
  - Check that the value of each foreign key either
    - –all null, or
    - –all non-NULL and occurs in the referenced relation.

Deletion: Constraint Checks

- How can deletions cause an invalid state?
- The record you're trying to delete could be a be referenced to as a foreign key in another relation.
- Example: Delete ***tuple*** with Person# = 2 from RESEARCHER

RESEARCHER:

Person#	Name
1	Dr C.C.Chen
2	Dr R.G.Wilkinson

ENROLMENT:

Enrolment#	Supervisee	Supervisor	Department	Degree
1	1	2	Psychology	Ph.D.
2	3	1	Comp.Sci.	Ph.D.
3	4	1	Comp.Sci.	M.Sc.
4	5	1	Comp.Sci.	M.Sc.

- What do we do?

Deletion: Constraint Checks

- We do need to delete tuples from relations sometimes, and it's possible for the record to referred in other relations.
- What can we do?

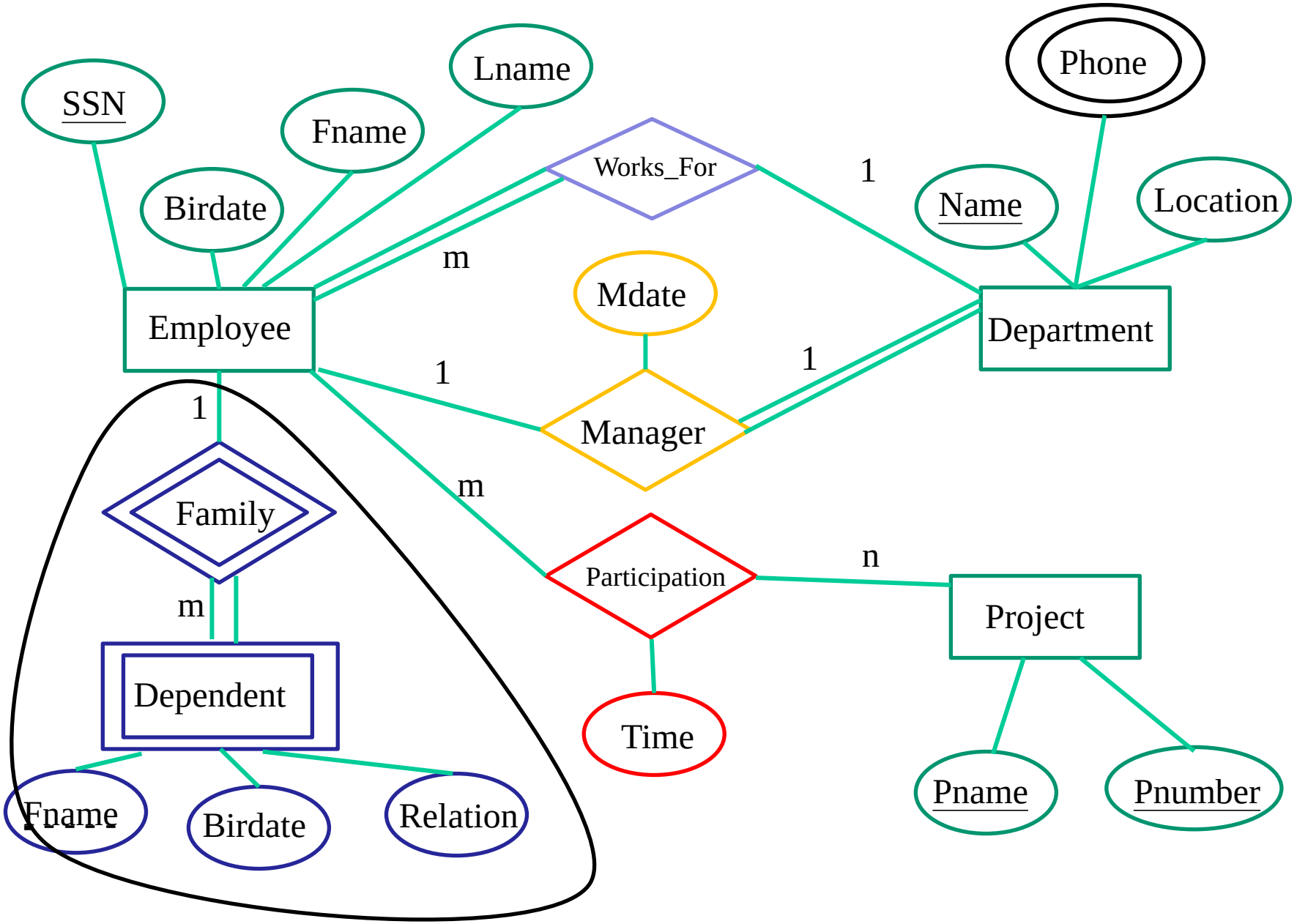
1. Delete it (*this requires another integrity check, possibly causing a cascade of deletions*), or
2. Set foreign key value to NULL (*note this can't be done if it is part of a primary key*)

Updating Values: Constraint Checks

- Can changing a value lead to an invalid state? Not unless you're modifying the value of a key.
- If the modified attribute is primary key
  - the same issues as deleting PK1 and then immediately inserting PK2.
  - make sure deletion and insertion don't violate any steps.
- if the modified attribute is foreign key
  - check that the new value refers to an existing tuple.
- Note: all relational integrity constraints are to do with the key values.

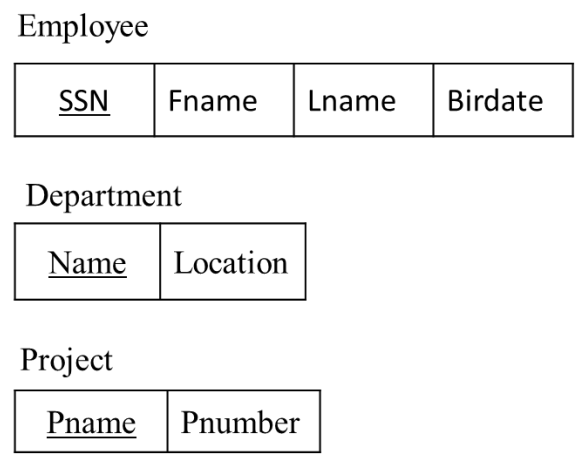
Mapping Weak Entity Types

- Step 2 : For each ***weak entity type*** W with the owner entity type E, create a new relation R with
  - Attributes :
    - all simple attributes (and simple components of composite attributes) of W,
    - and include the primary key attributes of the relation derived from E as the foreign key.
  - Key of R: foreign key to E and partial key of W.

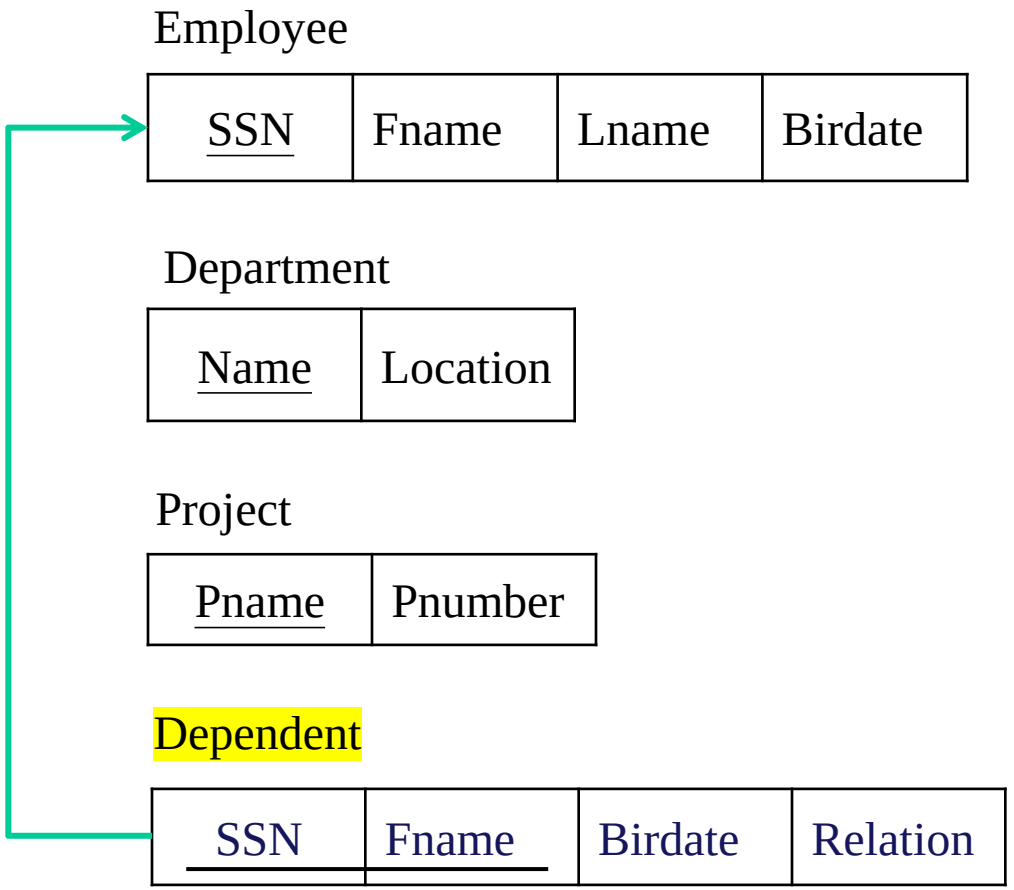


Mapping Weak Entity Types

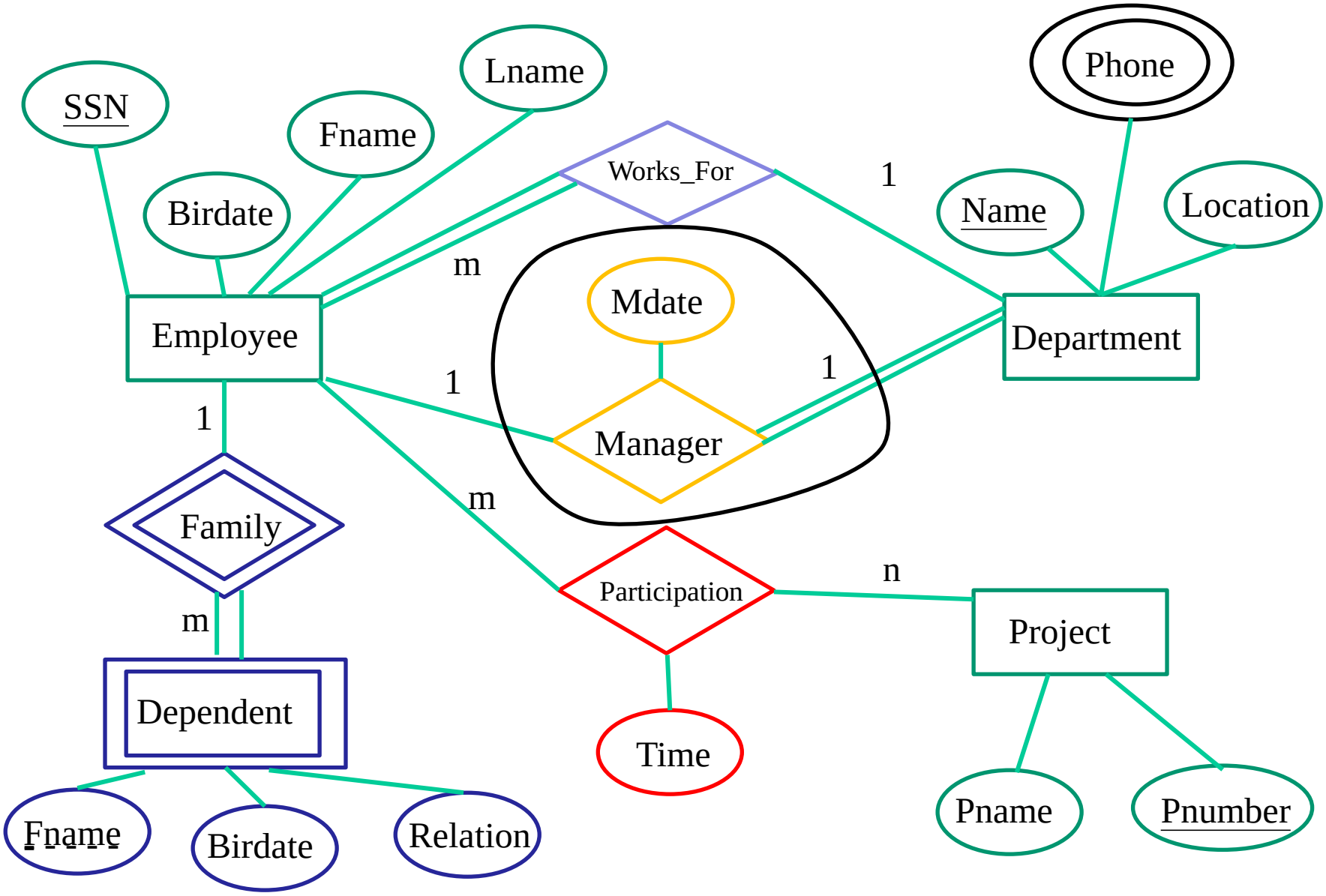
Mapping Weak Entity Types



before



after



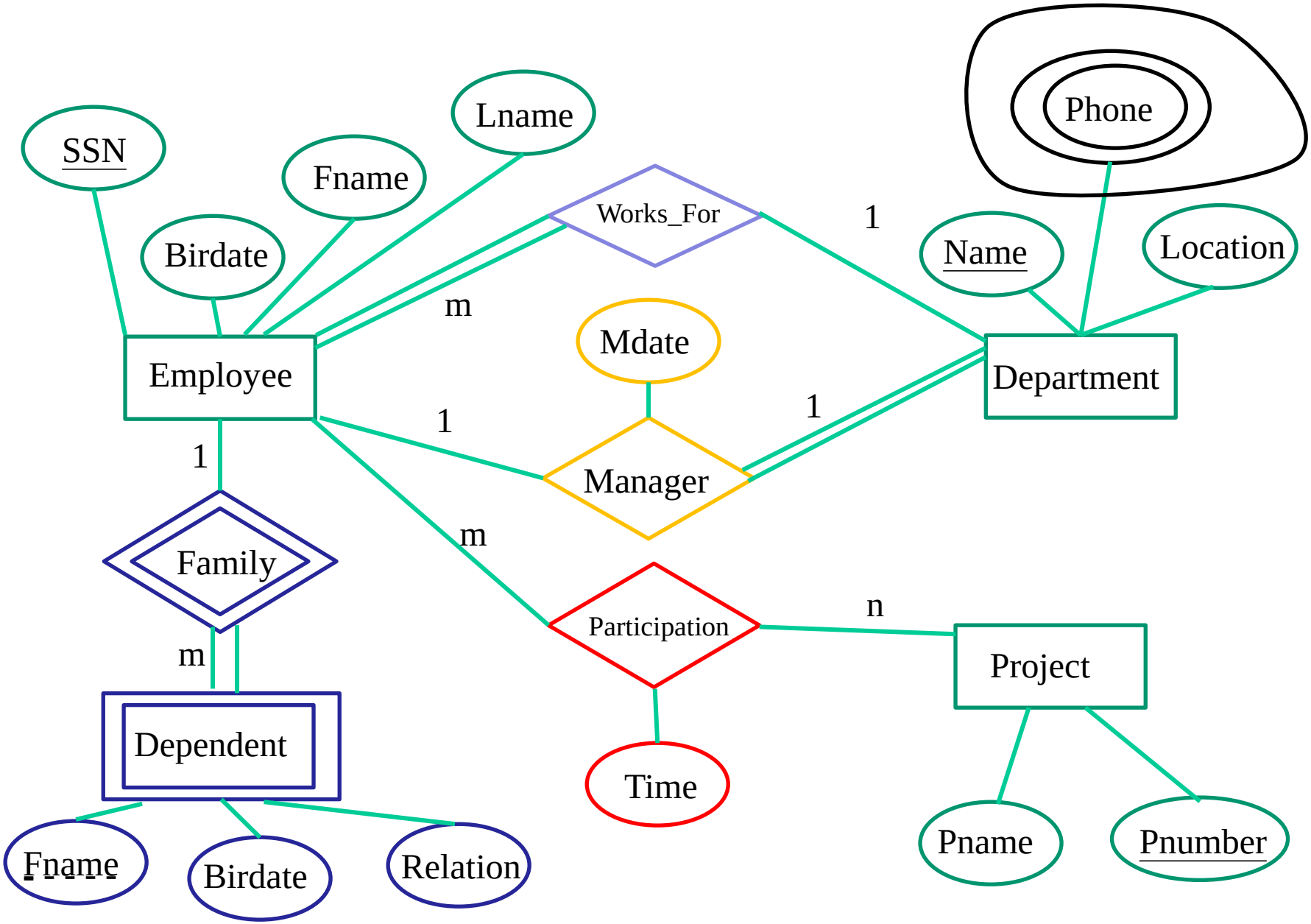
Mapping 1:1 Relationship Types

Mapping 1:1 Relationship Types

Step 3 : For each ***1:1 relationship type*** B. Let E and F be the participating entity types. Let S and T be the corresponding relations.

- Choose one of S and T (let S be one that participates totally if there is one).
- Add attributes from the primary key of T to S as a foreign key.
- Add all simple attributes (and simple components of composite attributes) of B as attributes of S.

(Alternatively: merge the two entity types and the relationship into a single relation, especially if both participate totally and do not participate in other relationships).



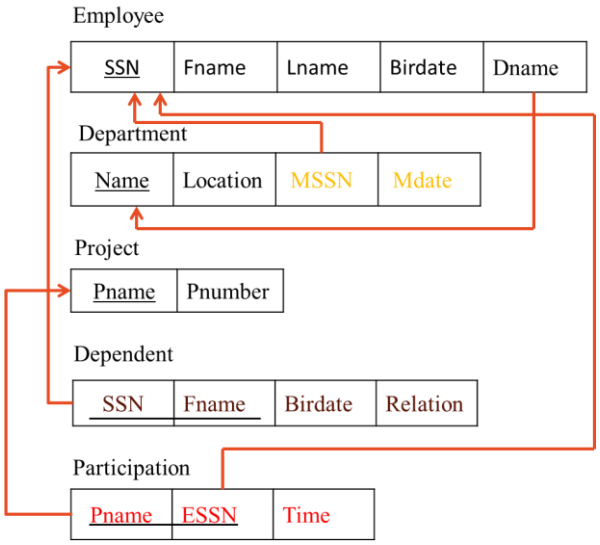
Mapping Multivalued Attributes

Mapping Multivalued Attributes

For each ***multivalued attribute*** A, where A is an attribute of E, create a new relation R.

- *If A is a multivalued simple attribute,*
  - Attributes of R = Simple attribute A, and key of E as a foreign key.
- *If A is a multivalued composite attribute,*
  - Attributes of R = All simple components of A, and key of E as a foreign key.

In both cases, the primary key of R is the set of all attributes in R.



before

Employee

SSN	Fname	Lname	Birdate	Dname

Department

Name	Location	MSSN	Msdate

Project

Pname	Pnumber

Dependent

SSN	Fname	Birdate	Relation

Participation

Pname	ESSN	Time

D_number

Dname	Dnumber

after

Mapping Multivalued Attributes

Mapping N-ary Relationship Types

For each ***n-ary relationship type*** (n > 2), create a new relation with

- Attributes : same for N:M relationship type.
- Key :
  - Also same for N:M relationship type, see exception below
  - The exception is that that if one of the participating entity types has participation ratio 1, its key can be used as a key for the new relation.

(Advice: binary relationships simpler to model)

Creating and Modifying Relations using SQL

- The subset of SQL that supports the **creation**, **deletion**, and **modification** of tables is called the Data Definition Language (DDL).
- To **create** the Student relation:

```
Create table Students
        (sid      CHAR(20),
         name     CHAR(30),
         login    CHAR(20),
         age      INTEGER,
         gpa      REAL)
```

- To **insert** a single tuple:

```
Insert
into      Students  (sid, name, login, age, gpa)
Values    (53688, 'Smith', 'smith@ee', 18, 3.2)
```

The list of column names can be omitted.

- Note that we can optimally omit the list of column names in the INTO clause and list the values in the appropriate order.
- It is a good style to be explicit about column names.

- Delete tuples using the **DELETE** command:

```
DELETE
FROM      Students S
WHERE  S.name = 'Smith'
```

- Modify the column values in an existing row using the **UPDATE** command:

```
UPDATE Students S
SET         S.age = S.age + 1, S.gpa = S.gpa – 1
WHERE  S.sid = 53688
```

- Note that
  - The WHERE clause is applied first and determines which rows are to be modified.
  - The set clause then determines how the rows are to be modified.
  - The expression on the right side of equals is the old value.

sid	name	login	age	gpa
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith#@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

```
UPDATE  Students S
SET     S.gpa = S.gpa – 0.1
WHERE   S.gpa >= 3.3
```

sid	name	login	age	gpa
50000	Dave	dave@cs	19	3.2
53666	Jones	jones@cs	18	3.3
53688	Smith	smith#@ee	18	3.2
53650	Smith	smith@math	19	3.7
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Integrity Constraints

- An integrity constraint (IC) is
  - a condition that is *specified on a database schema*, and
  - restricts the data that can be stored in an instance of the database.
- ICs are specified and enforced at different times.
  - Specified when the schema is defined.
  - Checked when relations are modified.

Key Constraints

- Key constraint
  - Certain minimal subset of the fields (candidate key) of a relation is a unique identifier for a tuple.
  - Out of all the available candidate keys, a database designer can identify a primary key.
    - The DBMS may create an index with the primary key.

- In SQL, we can declare
  - a key by the UNIQUE command
  - a primary key by the PRIMARY KEY constraint.

```
CREATE TABLE Students
        (sid        CHAR(20),
         name       CHAR(30),
         login      CHAR(20),
         age        INTEGER,
         gpa        REAL,
         UNIQUE (name, age),
         CONSTRAINT StudentsKey PRIMARY KEY ( sid))
```

Primary key

key

Constraint name (optional): if the constraint is violated, the constraint name is returned and can be used to identify the error

Foreign key constraints

- Sometimes, the information stored in a relation is linked to the information stored in another relation.
- If one of the relations is modified
  - The other must be checked,
  - Perhaps modified, to keep the data consistent.
- The most common IC involving two relations is a foreign key constraint.

Foreign key constraints

- Besides Students, suppose we have the following relation:

  Enrolled(sid: string, cid: string, grade: string)

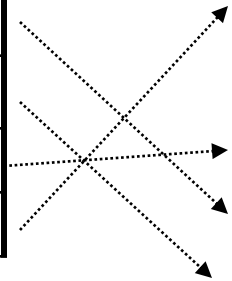
- To ensure that only genuine students can enroll in courses, any value that appears in the sid field of the Enrolled relation should also appear in the sid field of some tuple in the Students relation.
- The sid field of Enrolled is called a foreign key and refers to Students.

Foreign key constraints

- Formally, a foreign key is a set of fields in one relation r that is used to “refer” to a tuple in another relation s.
  - It must correspond to the primary key of s.
  - E.g. the foreign key in referencing relation ,Enrolled, must match the primary key of the referenced relation, Students.
    - i.e., same number of columns and compatible data types (may have different names).
- The foreign key condition specifies a referential integrity constraint.

Foreign key

cid	grade	sid
CSC101	C	53831
ERG202	B	53832
IEE321	A	53650
PSY203	B	53666



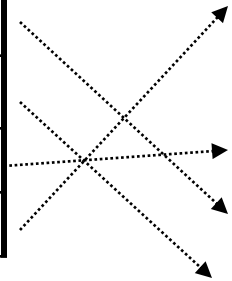
Primary key

sid	name	login	age	gpa
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith#@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

Primary key

Foreign key

cid	grade	sid
CSC101	C	53831
ERG202	B	53832
IEE321	A	53650
PSY203	B	53666



sid	name	login	age	gpa
50000	Dave	dave@cs	19	3.3
53666	Jones	jones@cs	18	3.4
53688	Smith	smith#@ee	18	3.2
53650	Smith	smith@math	19	3.8
53831	Madayan	madayan@music	11	1.8
53832	Guldu	guldu@music	12	2.0

- Cannot insert <55555, ART104,A> into Enroll, as there is no tuple in Students with sid = 55555.
- Cannot delete <53666, Jones, jones@cs, 18,3.4> from Students; since there is a tuple in Enroll with sid = 53666.

Foreign key constraints

- We can specify the foreign key constraints in SQL as follows:

```
CREATE TABLE Enrolled (
                    sid      CHAR(20),
                    cid      CHAR(20),
                    grade    CHAR(10),
                    PRIMARY KEY (sid, cid),
                    FOREIGN KEY (sid) REFERENCES Students)
```

- The foreign key constraint states that every sid value in Enrolled must also appear in Students.

Foreign key Violations (1)

SQL Provides several alternative ways to handle foreign key violations:

- What should we do if an Enrolled row is inserted, with a sid column value that does not appear in any row of the Students table?
  - The INSERT is simply reject.

Foreign key Violations (2)

SQL Provides several alternative ways to handle foreign key violations:

- What should we do if a Students row is deleted?
  1. Delete all Enrolled rows that refer to the deleted Students row.
  2. Disallow the deletion of the Students row if an Enrolled row refers to it.
  3. Set the sid column to the sid of some existing 'default' student, for every Enrolled row that refers to the deleted Student row.
  4. For every Enrolled row that refers to it, set the sid column to null.

Foreign key Violations (3)

SQL Provides several alternative ways to handle foreign key violations:

- What should we do if the primary key value of a Students row is updated?
  - The options here are similar to the previous case.

Foreign key Violations (4)

SQL allows us to choose any of the four options on DELETE and UPDATE (see next page)

```
CREATE TABLE Enrolled (
        sid     CHAR(20),
        cid     CHAR(20),
        grade   CHAR(10),
        PRIMARY KEY (sid, cid),
        FOREIGN KEY (sid) REFERENCES Students
                                ON DELETE CASCADE
                                ON UPDATE NO ACTION)
```

- The options are specified as part of the foreign key declaration:
  - NO ACTION (default)
    - The action (DELETE or UPDATE) is to be rejected.
  - CASCADE
    - If a Students row is deleted, all Enrolled rows that refer to it are to be deleted as well.
  - SET DEFAULT
    - Switch the enrollment to a ‘default’ student.
    - The default student is specified as part of the definition of the sid field in Enrolled

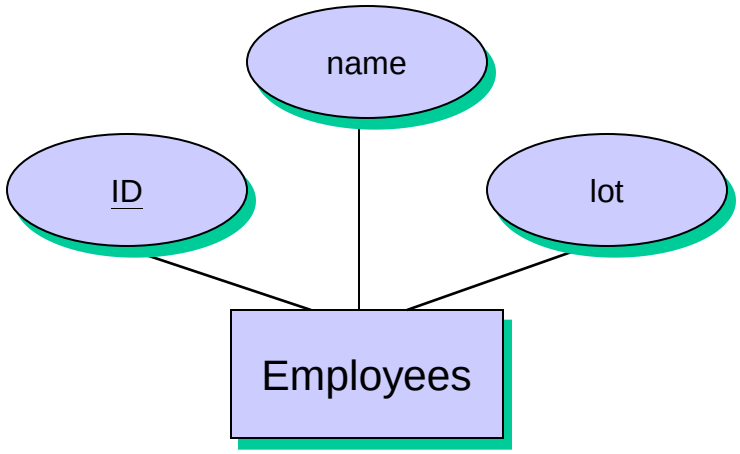
      E.g., sid CHAR(20) DEFAULT ‘53666’.
  - SET NULL
    - Allows the use of null as the default value.

Logical Database Design: ER to Relational

- The ER model is convenient for representing an initial, high-level database design.
- Given an ER diagram describing a database, a standard approach is taken to generate a relational database schema that closely approximates the ER design.

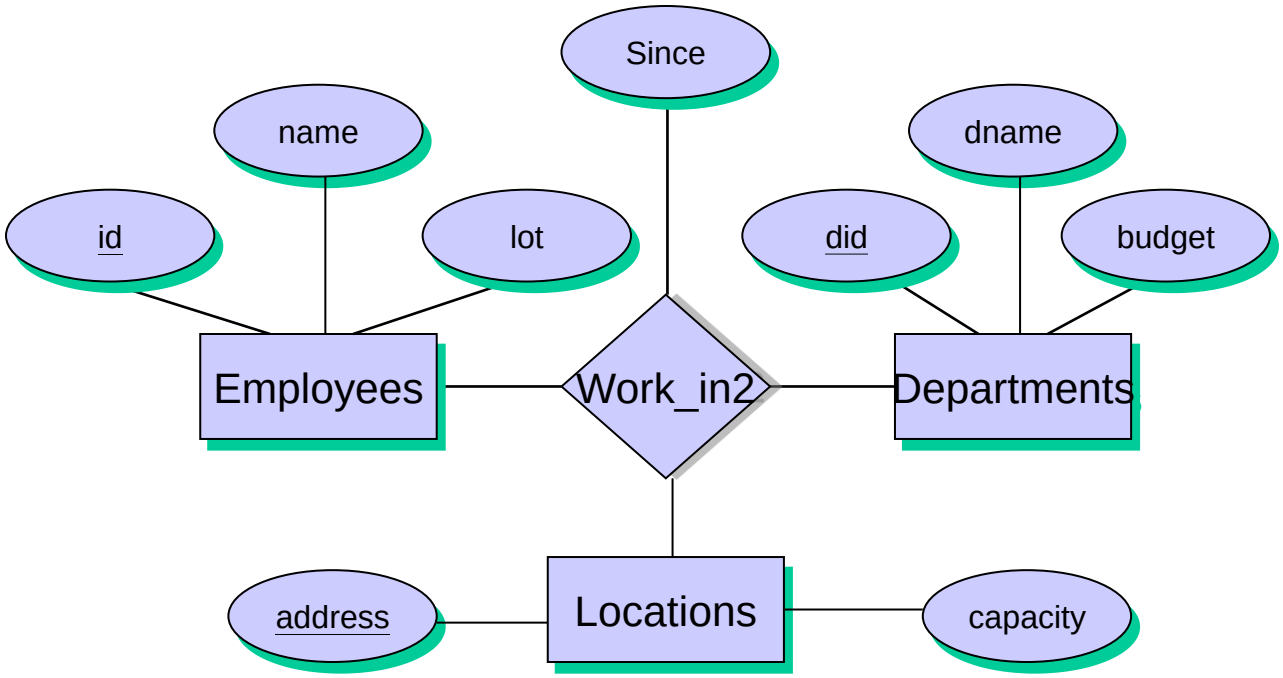
- This time let's look at it together with SQL DDL commands

- Entity Sets to Tables
  - An entity set is mapped to a relation in a straightforward way:
    - attribute of entity set → attribute of table.



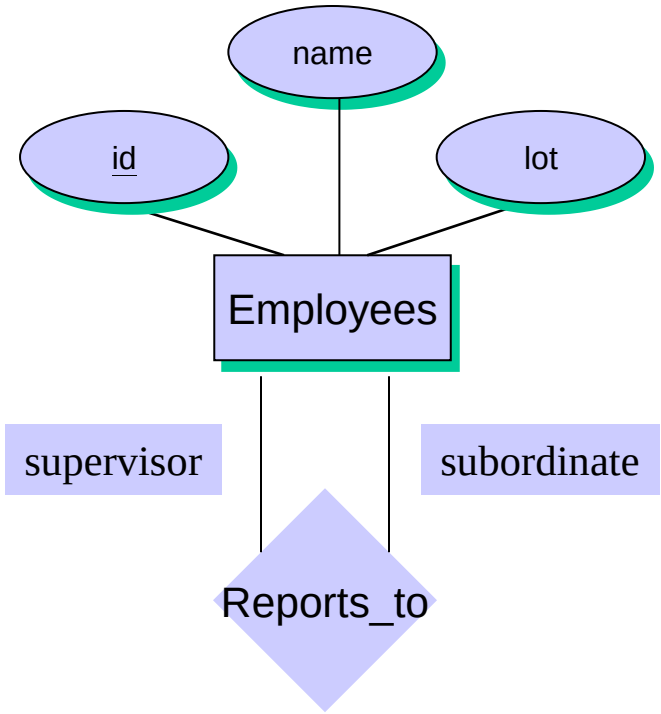

```
CREATE TABLE Employees (
        id          CHAR(11),
        name        CHAR(30),
        lot         INTEGER,
        PRIMARY KEY (id))
```

- Relationship Sets (without Constraints) to tables
  - Suppose there is no key and participation constraints.
  - Must be able to identify each participating entity and give values to the descriptive attributes of the relationship.
  - The attributes of the relation include:
    - The primary key attributes of each participating entity set, as foreign key fields.
    - The descriptive attributes of the relationship set.
  - The set of nondescriptive attributes is a superkey for the relation.
    - If there is no key constraints, this set of attributes is a candidate key.



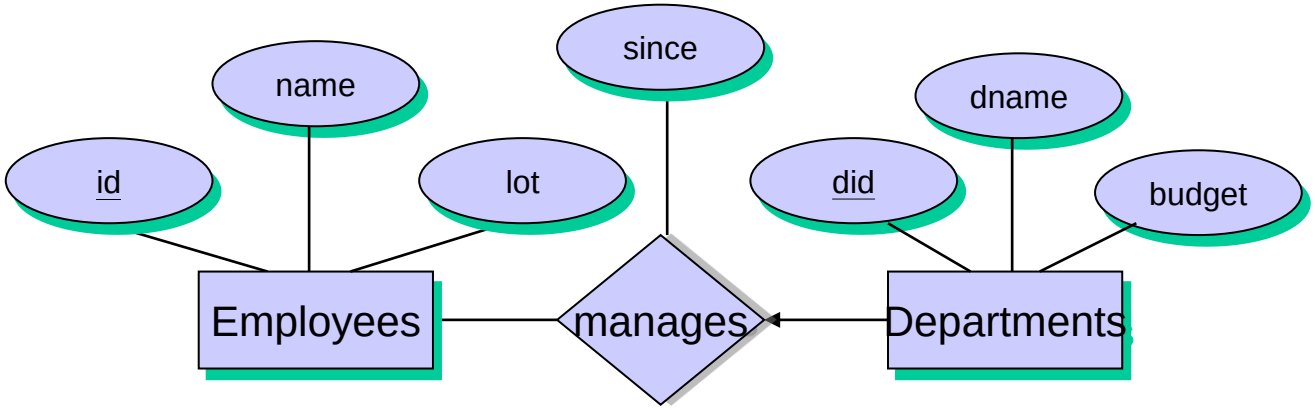

```
CREATE TABLE Works_In2(
        id          CHAR(11),
        did         INTEGER,
        address     CHAR(20),
        since       DATE,
        PRIMARY KEY (id,did,address),
        FOREIGN KEY (id) REFERENCES Employees,
        FOREIGN KEY (address) REFERENCES Locations,
        FOREIGN KEY (did) REFERENCES Departments)
```

Cannot take on null values

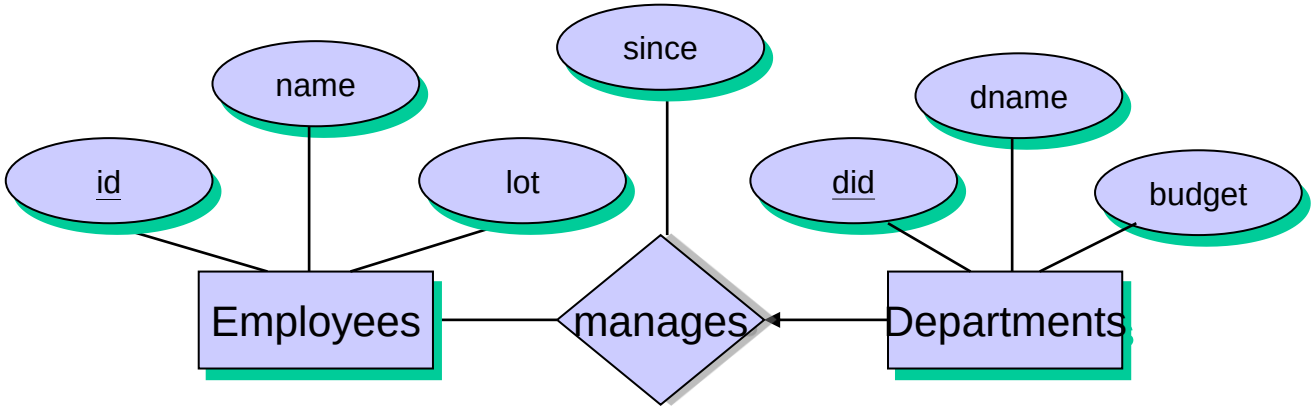



```
CREATE TABLE Reports_To(
        supervisor_id       CHAR(11),
        subordinate_id      CHAR(11),
        PRIMARY KEY (supervisor_id, subordinate_id),
        FOREIGN KEY (supervisor_id) REFERENCES Employees(id),
        FOREIGN KEY (subordinate_id) REFERENCES Employees(id))
```

We need to explicitly name the referenced field of Employees because the field name differs from the name(s) of the referring field(s).



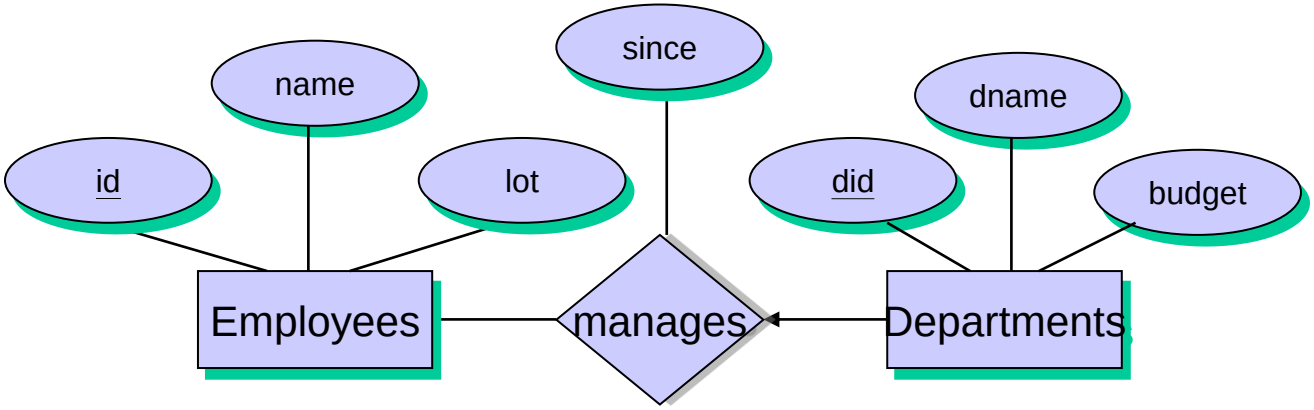
- Translating Relationship Sets with Key Constraints
  - If a relationship set involves *n* entity sets and some *m* of them are linked via arrows in the ER diagram.
    - The key of any one of these *m* entity sets constitutes a key for the relation to which the relationship set is mapped.
    - Hence, we have *m* candidate keys, and one of these should be designated as the primary key.



- The approach is to include the information about the relationship set in the table corresponding to the entity set with the key.
  - Avoids creating a distinct table for the relationship.
  - The drawback to this approach is that space could be waste if a lot of departments have no managers.

```
CREATE TABLE Dept_Mgr (
        did       INTEGER,
        dname     CHAR(20),
        budget    REAL,
        id        CHAR(11),
        since     DATE,
        PRIMARY KEY (did),
        FOREIGN KEY (id) REFERENCES Employees)
```

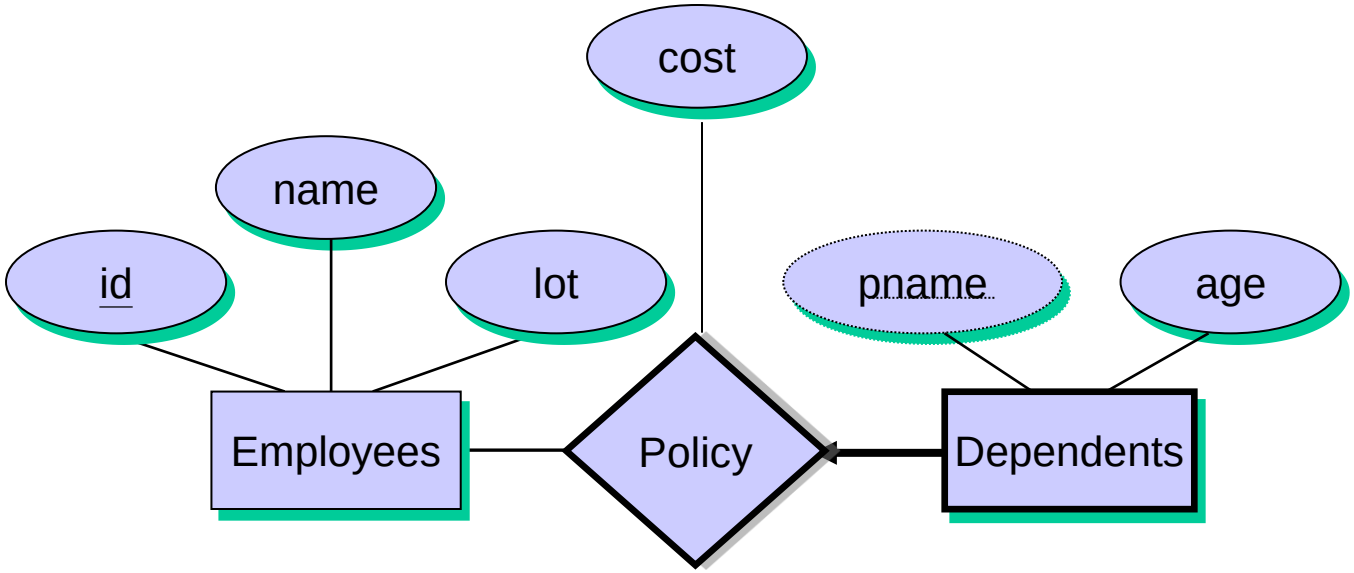
Another approach (not rec.)




```
CREATE TABLE Manages (
        id          CHAR(11),
        did         INTEGER,
        since       DATE,
        PRIMARY KEY (did),
        FOREIGN KEY (id) REFERENCES Employees,
        FOREIGN KEY (did) REFERENCES Departments)
```

As each department has at most one manager, no two tuples can have the same did value but differ on the id value. Therefore, did is itself a key for Manages; indeed, the set {did, id} is not a key because it is not minimal.

- Translating weak entity sets
  - The weak entity set always
    - participates in a one-to-many binary relationship,
    - has a key constraint
    - has total participation.
  - The 2nd approach (p.45) for translation relationship with key constraint can be used.
  - We must take into account that the weak entity has only a partial key.
  - Also when an owner entity is deleted, we may want all owned weak entities to be deleted.




```
CREATE TABLE Dep_Policy (
        pname     CHAR(20),
        age       INTEGER,
        cost      REAL,
        id        CHAR(11),
        PRIMARY KEY (pname,id),
        FOREIGN KEY (id) REFERENCES Employees
                ON DELETE CASCADE)
```

- The primary key is <pname,id>, since dependents is a weak entity.
- We have to ensure that every dependents entity is associated with an Employees entity (the owner), as per the total participation constraint, ie. id cannot be null. This is ensured because id is part of the primary key.
- The CASCADE option ensures that information about an employee's policy and dependents is deleted if the corresponding Employees tuple is deleted.

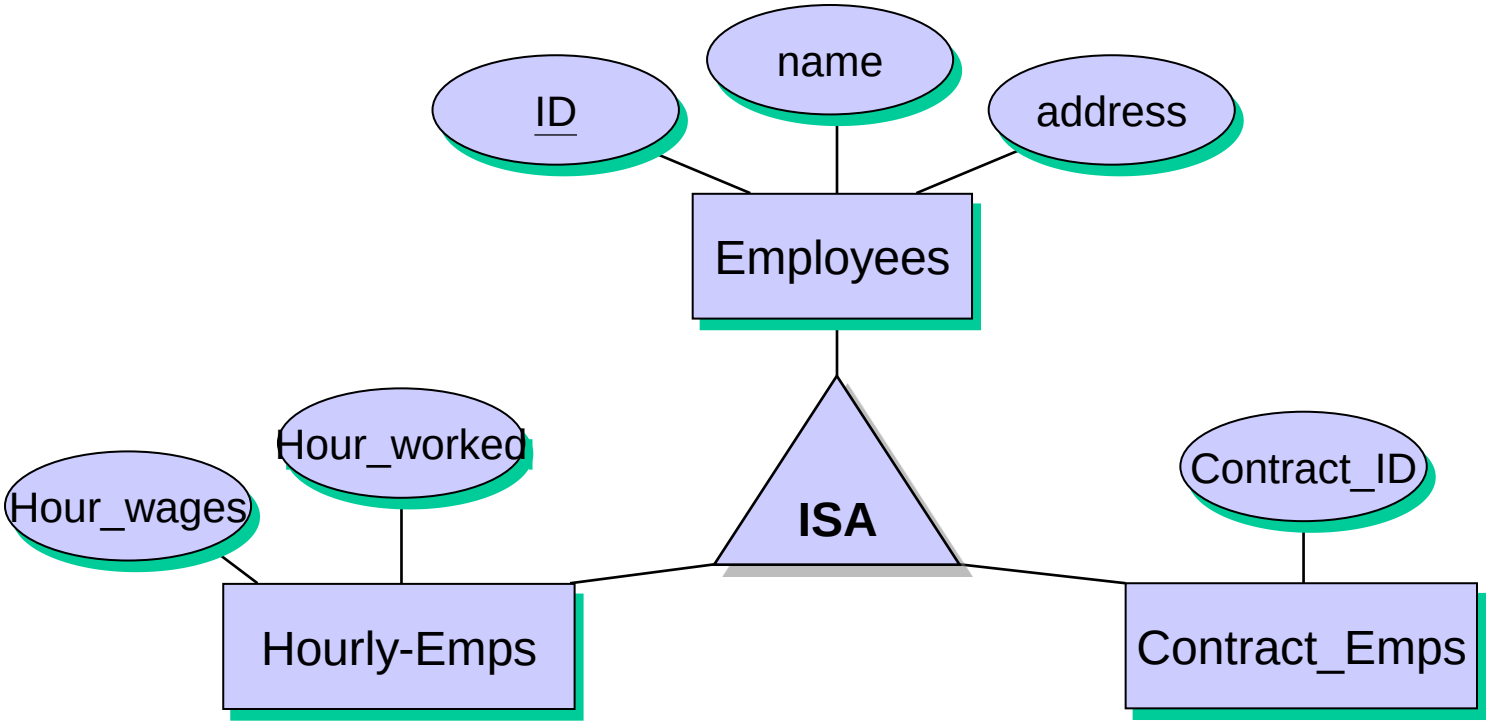
Database Design

More on Entity-relationship Model

(EER model)

Class Hierarchies

- Sometimes it is natural to classify the entities in an entity set into subclasses.



- Attributes are inherited by the entity set in the subclass.
  - E.g. the attributes defined for an Hourly_Emps entity are the attributes for Employees plus that of Houly_Emps.
- A class hierarchy can be viewed in one of the two ways.
  1. A class is specialized into subclasses.
  2. The subclasses are generalized by a superclass.

- We can specify two kinds of constraints with respect to ISA hierarchies,
  - Overlap constraints
    - Determine whether two subclasses are allowed to contain the same entity.

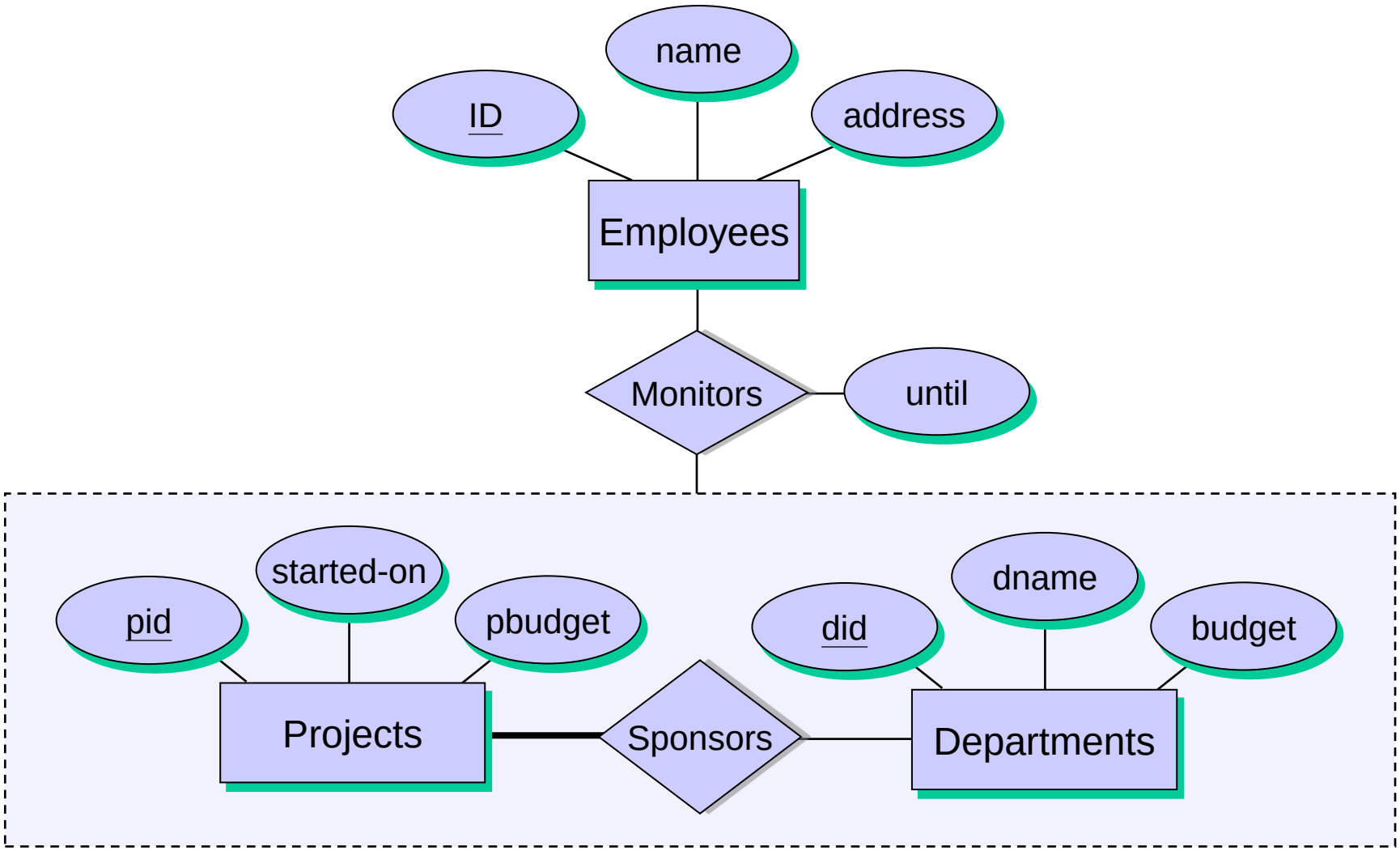
      E.g. Can an employee be an Hourly_Emps as well as a Contract_emps entity?
  - Covering constraints
    - Determine whether the entities in the subclasses collectively include all entities in the superclass.

      E.g. Does every Employee entity also have to be an Hourly_Emps or a Contract_emps.

- Reasons for using hierarchy:
  - Add descriptive attributes that make sense only for the entities in a subclass.
    - E.g. Hourly_wages does not make sense for a Contract_Emps entity.
  - Identify the set of entities that participate in some relationship.
    - E.g. we might wish to define the manages relationship so that the participating entity sets are Senior_Emps and Departments to ensure that only senior employees can be managers.

Recap: Aggregation

- Sometimes, we have to model a relationship between a collection of entities and *relationships*.
- Aggregation allows us to indicate that a relationship set (identified through a dashed box) participates in another relationship set.
- Example
  - Suppose we have an entity set called projects.
  - Each projects entity is sponsored by one or more departments.
  - A department that sponsors a project might assign employees to monitor the sponsorship.
  - Intuitively, monitors should be a relationship set that associates a Sponsors relationship (rather than a projects or departments entity) with an Employees entity.

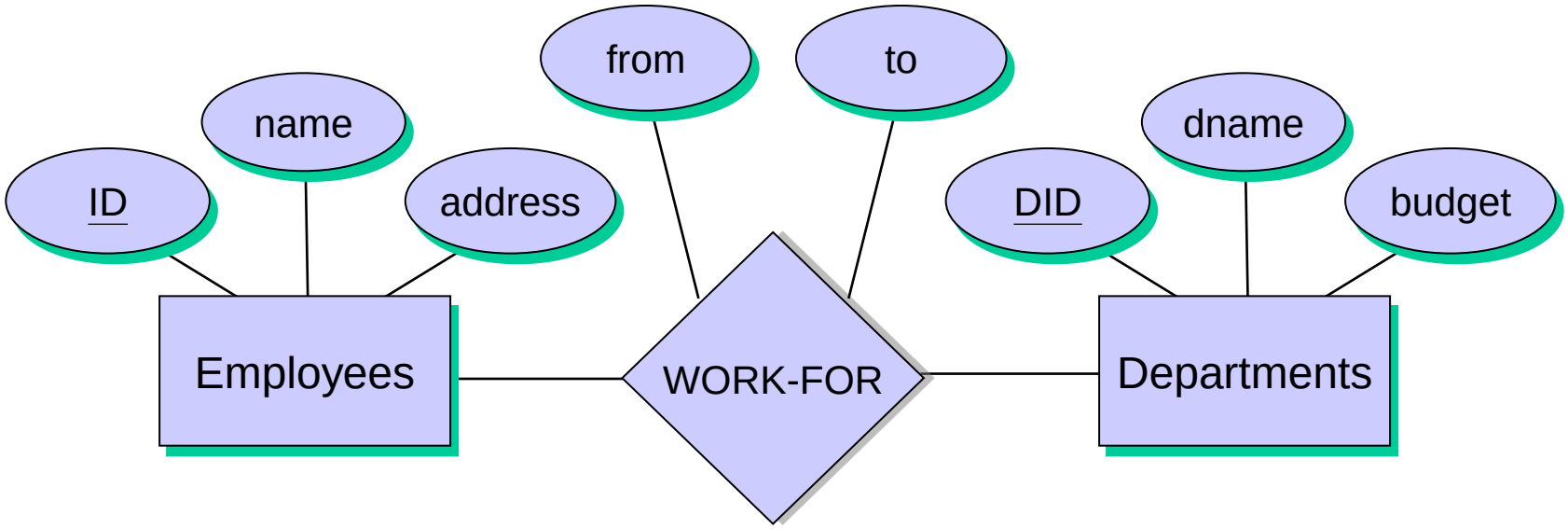

name
ID
address
Employees
Monitors
until
started-on
pid
pbudget
dname
did
budget
Projects
Sponsors
Departments

Conceptual Design E-R model

- Developing an ER diagram presents several choices, including the following:
  - Should a concept be modeled as an entity or an attribute?
  - Should a concept be modeled as an entity or a relationship?
  - What are the relationship sets and their participating entity sets? Should we use binary or ternary relationships?
  - Should we use aggregation?

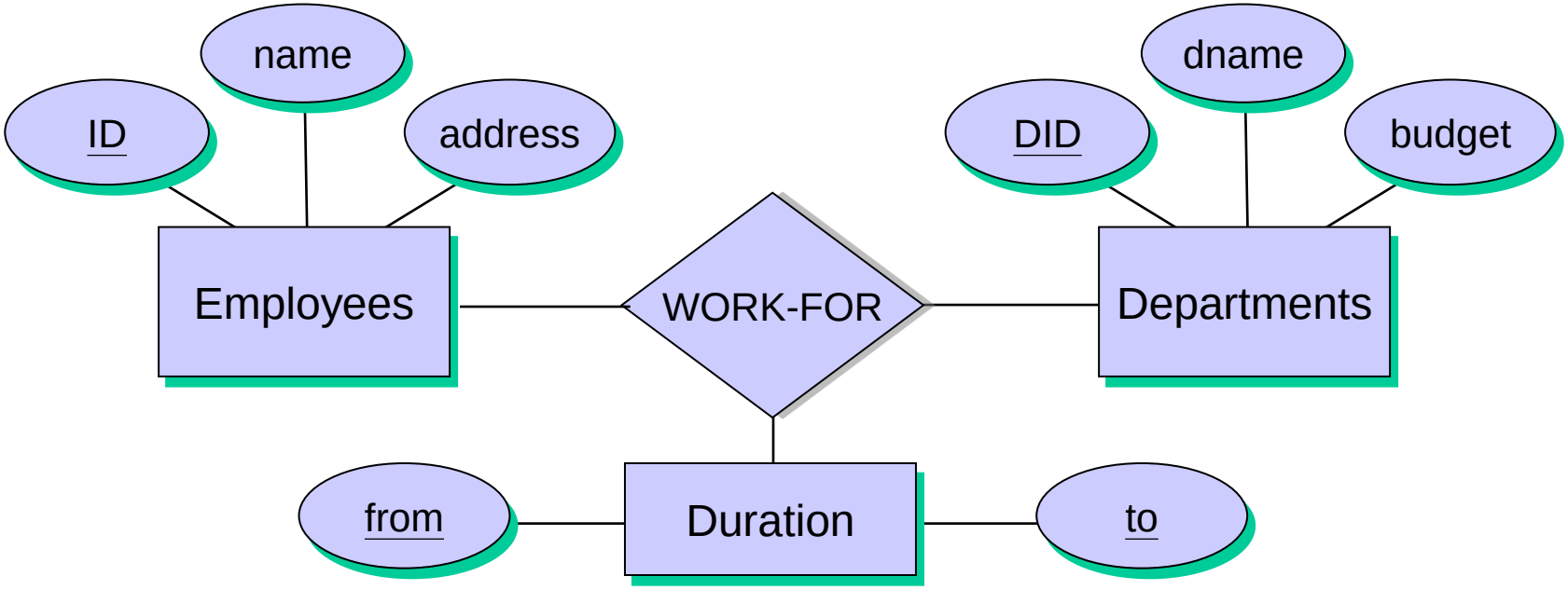
- Entity versus Attribute
  - Should *address* be an attribute of Employees or an entity (connected to Employees by a relationship)?
    1. If we have several addresses per employee, *address* must be an entity (*attributes cannot be set-valued*)
    2. If the structure (city, street, etc) is important, e.g. want to retrieve employees in a given city, address must be modeled as an entity (*attribute values are atomic*)

- Should duration of an employee working in a department be an attribute or an entity?

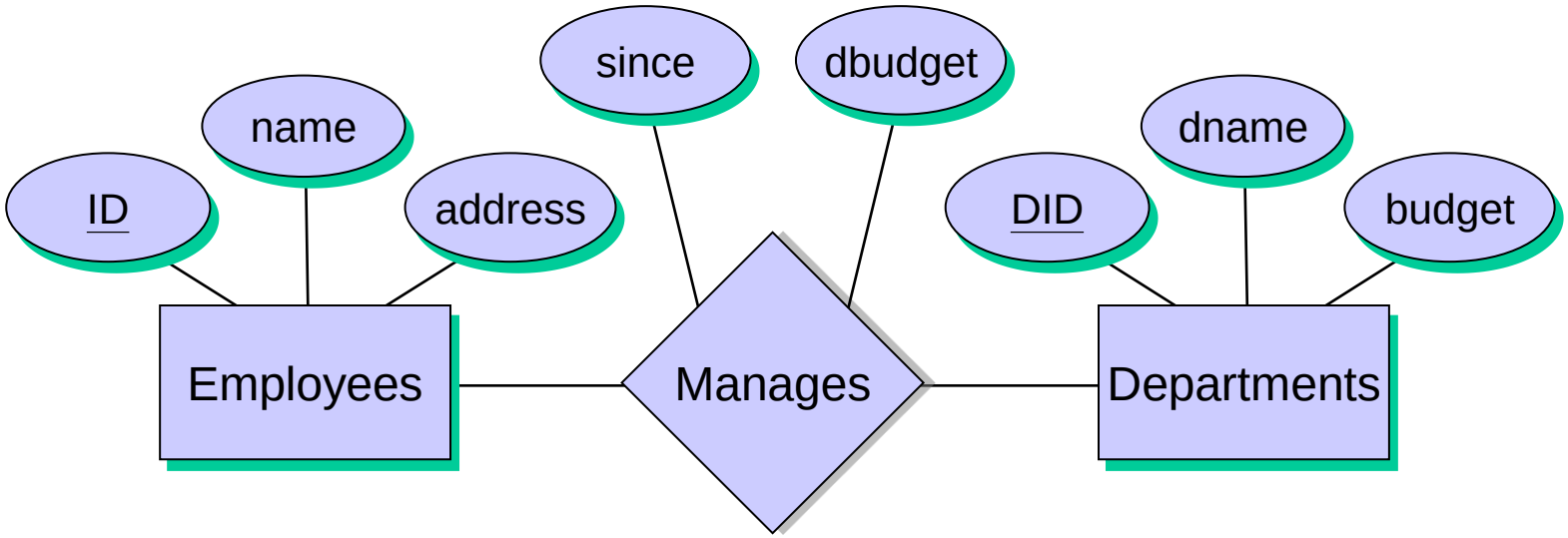


- This ER model does not allow an employee to work in a department for two or more periods.
  - Because a relationship is uniquely identified by the participating entities.

- The problem can be addressed by introducing an entity set called Duration.

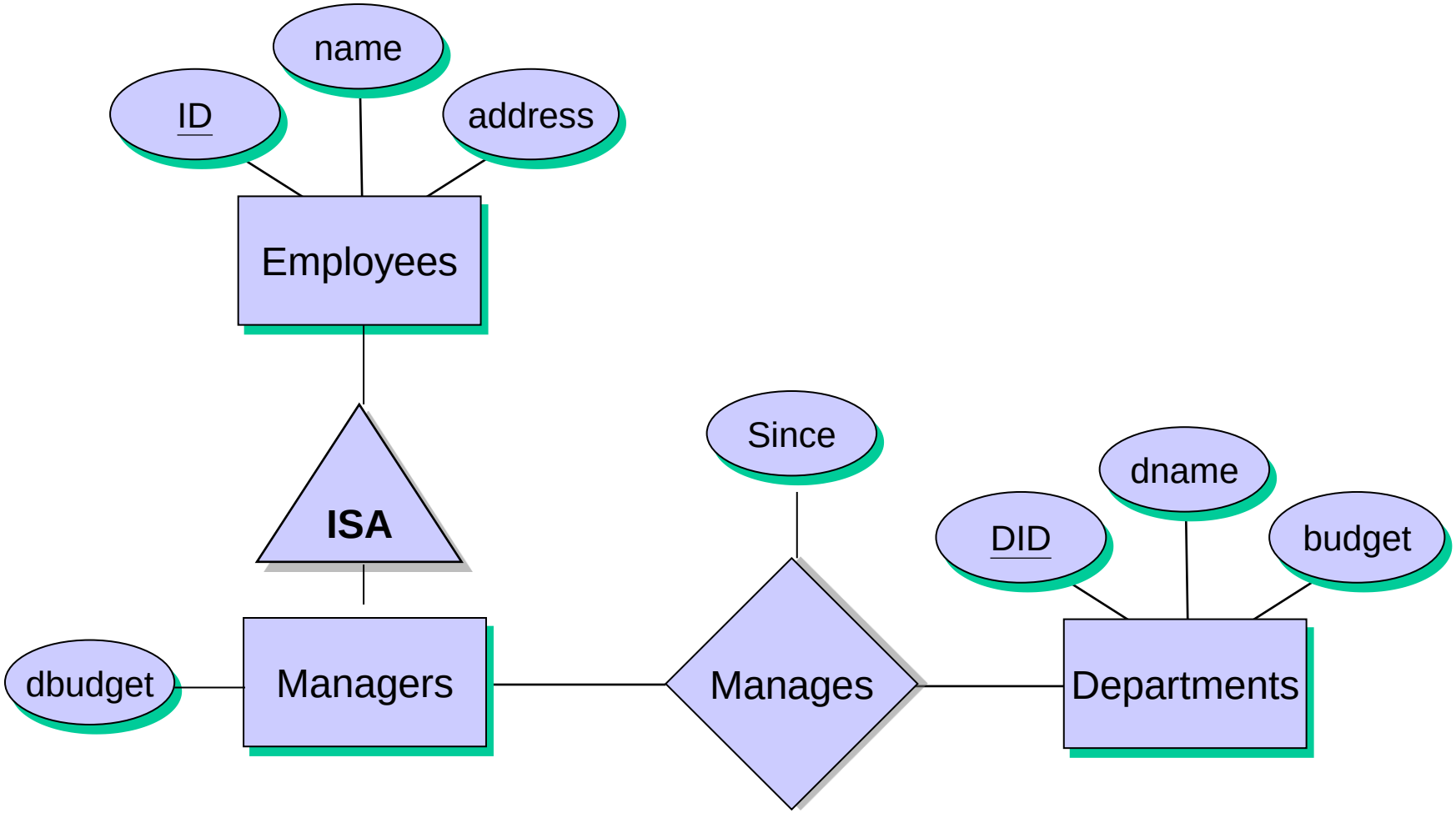


- Entity versus Relationship
  - Suppose each department manger is given a discretionary budget (dbudget).



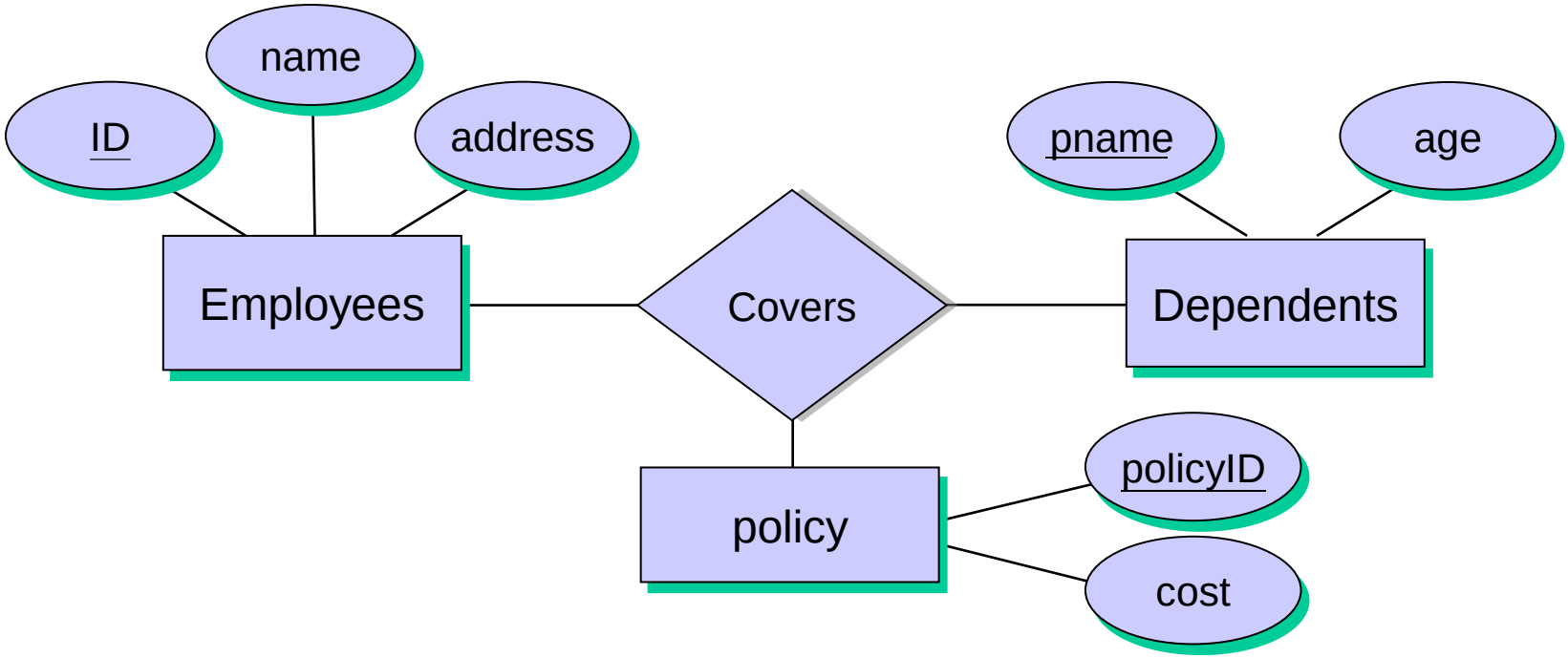
  - This approach is natural if we assume that a manager receives a separate discretionary budget for each department.

- What if the discretionary budget is a sum that covers all departments managed by that employee?
  - In this case, each manages relationship that involves a given employee will have the same value in the dbudget field, leading to redundant storage of the same information.
  - It is also misleading; it suggests that the budget is associated with the relationship, when it is actually associated with the manger.



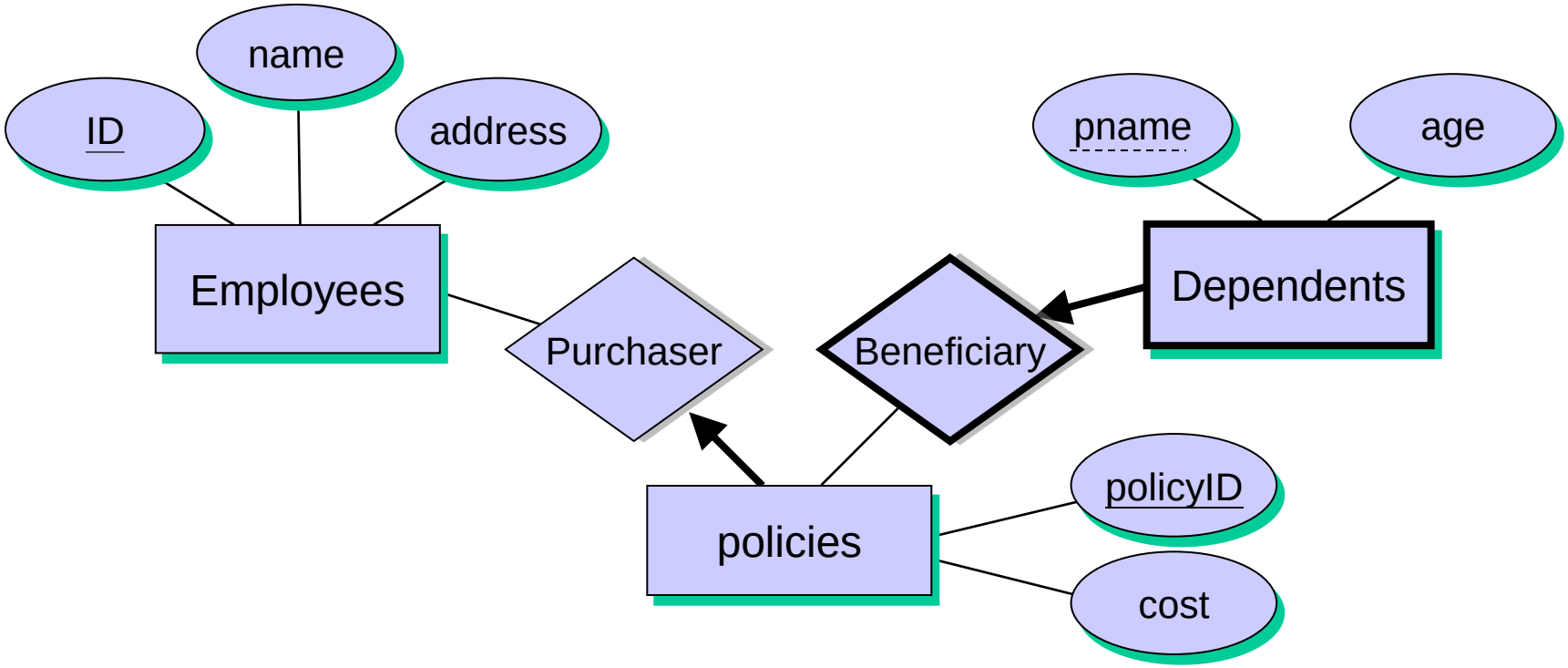
- We can address the problem by introducing a new entity set called managers by the ISA hierarchy.

- Binary versus Ternary Relationships
  - The following ER diagram models the situation that an employee can own several policies, each policy can be owned by several employees, and each dependent can be covered by several policies.

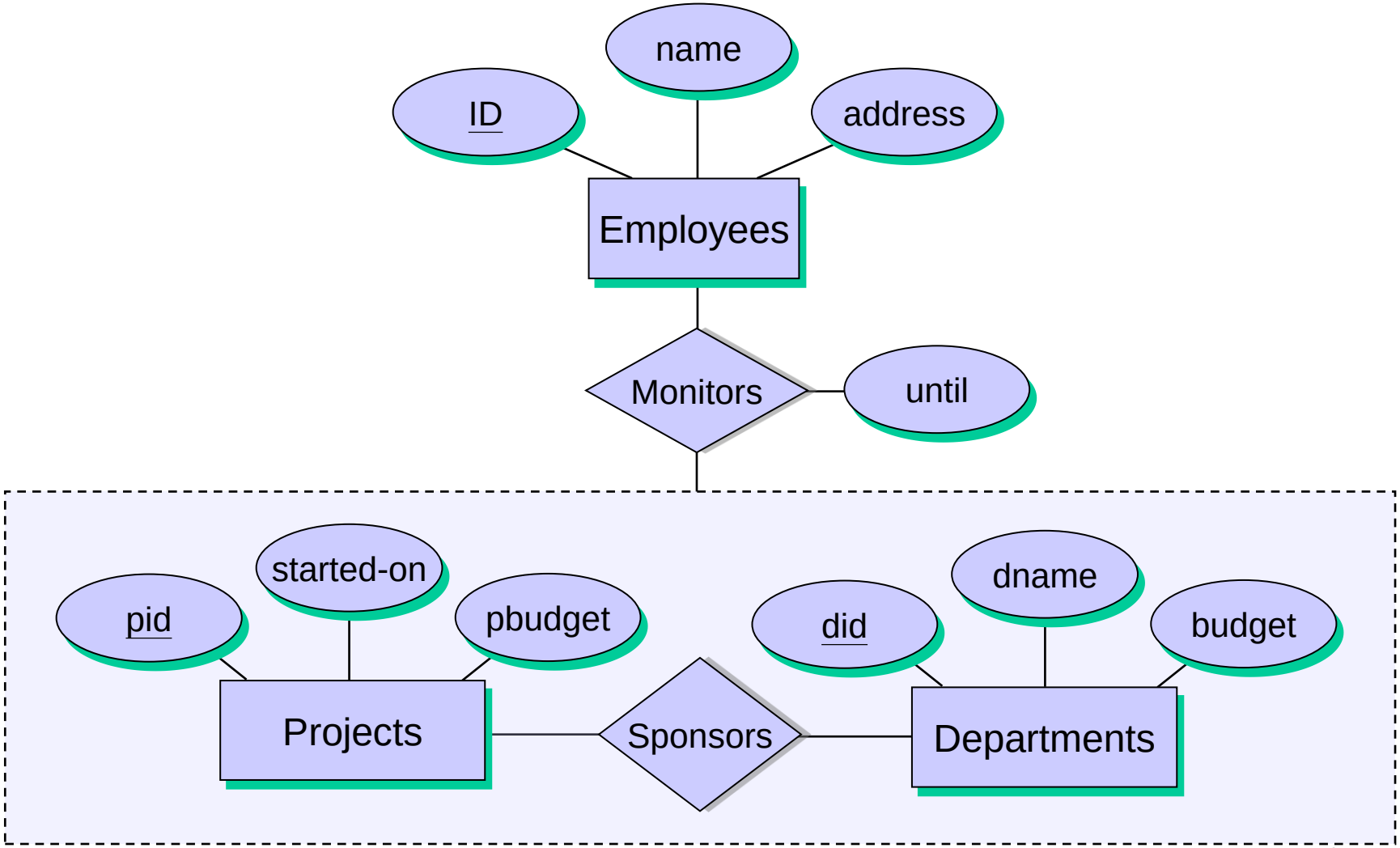


- Suppose we have the following additional requirements:
  - A policy cannot be owned jointly by two or more employees. (Impose a key constraint on Policies with respect to Covers, but it introduces a side effect that each policy can cover only one dependent).
  - Every policy must be owned by some employee. (Impose a total participation constraint on Policies, it is acceptable if each policy covers at least one dependent).
  - Dependents is a weak entity set, and each dependent entity is uniquely identified by taking pname in conjunction with the policyid of a policy entity.

- Here is a solution

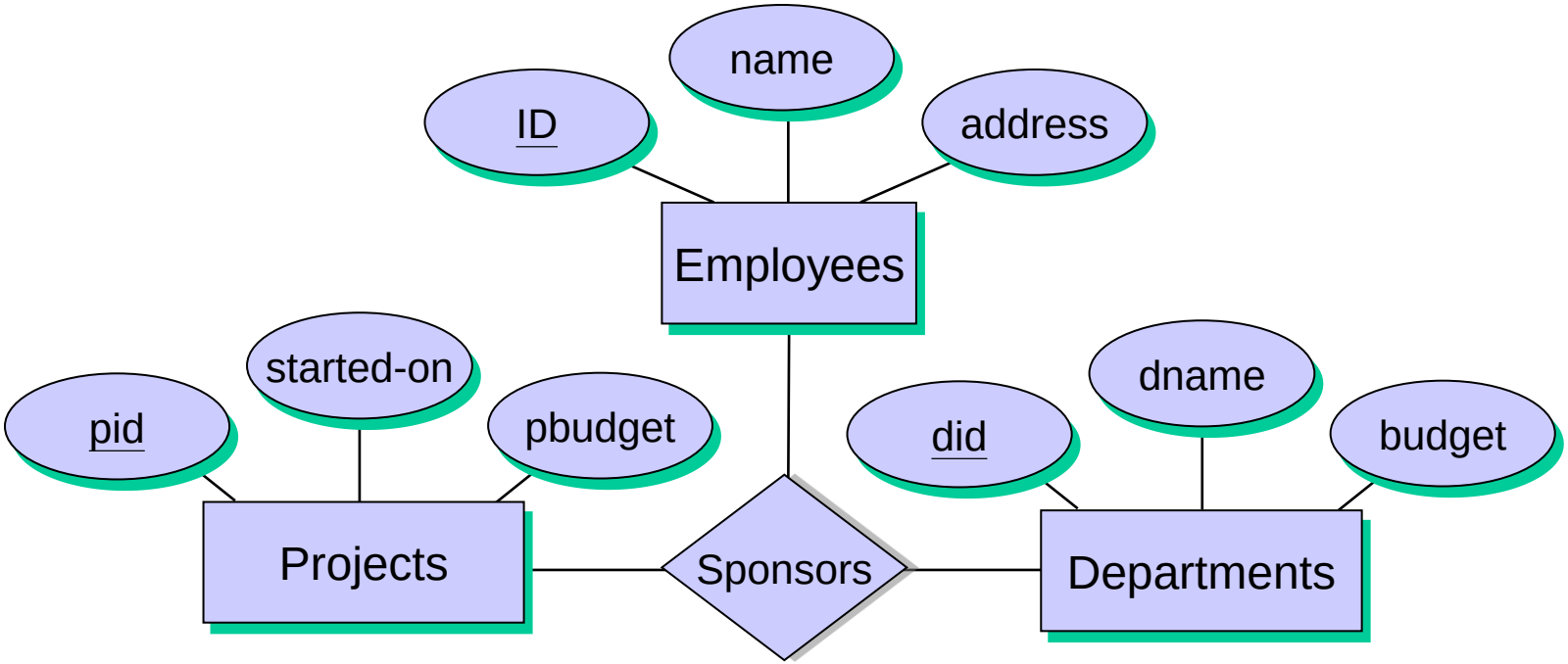


Aggregation versus Ternary Relationships



- According to the previous ER diagram:
  - A project can be sponsored by any number of departments,
  - A department can sponsor one or more projects,
  - Each sponsorship is monitored by one or more employees.

- If we don’t need to record the *until* attribute of Monitors, we might use a ternary relationship.



- However, this design cannot express the constraint that each sponsorship can be monitored by at most one employee.

- If aggregation is used, it is easy for us to draw an arrow from the aggregated relationship sponsors to the relationship monitors.