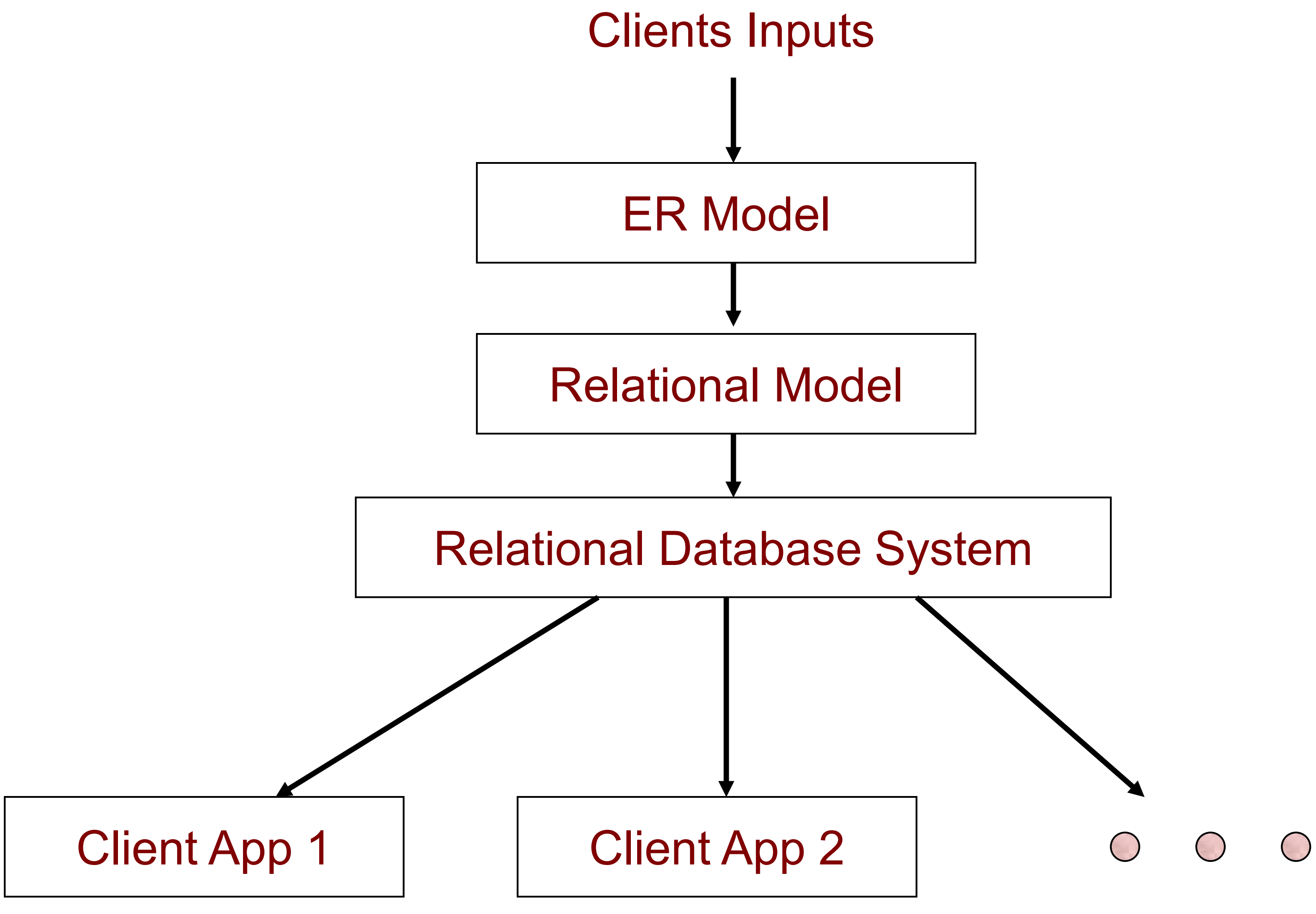
SE240: Introduction to Database Systems

Lecture 03: Relational Model

Outline

- **Relational Model**
  - **Relational Model Concepts**
  - Relational Database Schemas
- ER-to-Relational Mapping
  - Translating traditional ER diagrams
  - Translating Class Hierarchy

Database Modeling



Introduction

- The relational model was first introduced by Ted Codd of IBM Research in 1970
- Attracted due to its simplicity, elegance and mathematical foundations
- The model uses the concept of a mathematical relation – which looks like a table of values
- The **SQL Query Language** was developed by IBM in the 1970's

Why Study the Relational Model?

- The relational model is by far the dominant data model
- It is the foundation for the leading DBMS products of many vendors such as Oracle, Microsoft SQL Server, etc.
- A major strength of the relational model is that it supports simple, powerful querying of data
- Queries can be written intuitively, and the DBMS can perform efficient evaluation

The Relational Data Model

- The relational Model of Data is based on the concept of a *Relation, a* Relation is a mathematical concept based on the ideas of sets
- Represents data as a collection of relations
- **Table** of values
  - Row
    - Represents a collection of related data values
    - Fact that typically corresponds to a real-world entity or relationship
  - Table name and column names
    - Interpret the meaning of the values in each row *attribute*

Relation Name/Table Name

Attributes/Columns (collectively as a schema)

STUDENT

Name	<u>**Student-id**</u>	**Age**	**CGA**
Chan Kin Ho	99223367	23	11.19
Lam Wai Kin	96882145	17	10.89
Man Ko Yee	96452165	22	8.75
Lee Chin Cheung	96154292	16	10.98
Alvin Lam	96520934	15	9.65

Tuples/Rows

$R(A_1, A_2, A_3, A_4)$

- **Relation** ↔ **table**; denoted by $R(A_1, A_2, ..., A_n)$ where R is a **relation name** and $(A_1, A_2, ..., A_n)$ is the **relation schema** of R
- **Attribute (column)** ↔ denoted by A_i
- **Tuple (Record)** ↔ row
- **Attribute value** ↔ value stored in a table cell
- **Domain** ↔ legal type and range of values of an attribute, denoted by $dom(A_i)$

Dom(Age): [0-100]
Dom(EmpName): 50 alphabetic chars
Dom(Salary): non-negative integer

Relation Schema

- The Schema is a description of a Relation:
  - Denoted by $R(A_1, A_2, ...A_n)$, R is the name of the relation
  - The attributes of the relation are $A_1, A_2, ..., A_n$
  - All legal values of an attribute is called its domain
- Example:
- STUDENT(Name, Student-id, Age, CGA)
  - STUDENT is the relation name
  - Defined over the four attributes: Name, Student-id, Age, CGA
  - A STUDENT state may include 5 STUDENTs; another 250 STUDENTs

Relations States are Sets

- Relation State r, or r(R): a $\boxed{\text{specific state}}$ of relation R – this is a set of tuples (rows)

  subset

  - r(R) = {t_1, t_2, …, t_n} where each t_i is an n-tuple, t_i = <v_1, v_2, …, v_n> where each v_j element-of dom(A_j)
  - **Mathematical relation** of degree n on the domains dom(A_1), dom(A_2), ..., dom(A_n)
  - **Subset** of the **Cartesian product** of the domains that define R: $r(R) \subseteq dom(A_1) \times dom(A_2) \times ... \times dom(A_n)$
- All tuples in a relation state r(R) form a set
- By definition, there cannot be duplicates in a set
- By definition, set elements (tuples) are **not ordered**, even though tuples frequently appear to be in the tabular form

Example

- Let $R(A_1, A_2)$ be a relation schema:
  - Let $dom(A_1) = \{ 0,1 \}$
  - Let $dom(A_2) = \{ a, b, c \}$
- Then: $dom(A_1) \times dom(A_2)$ is all possible combinations: {<0,a>, <0,b>, <0,c>, <1,a>, <1,b>, <1,c>}
- The relation state $r(R) \subseteq dom(A_1) \times dom(A_2)$
- For example: r(R) could be {<0,a>, <0,b>, <1,c>}
  - this is one possible state r of the relation R, defined over A_1 and A_2
  - it has three 2-tuples: <0,a>, <0,b>, <1,c>

Values in Relations

- We refer to **component values** of a tuple t by:
  - $t[A_i]$ or $t.A_i$
  - This is the value v_i of attribute A_i for tuple t
- Similarly, $t[A_u, A_v, ..., A_w]$ refers to the subtuple of t containing the values of attributes $A_u, A_v, ..., A_w$, respectively in t
- All values are considered *atomic* (indivisible)
- Each value in a tuple must be from the domain of the attribute for that column, i.e., $v_i = t[A_i] \in dom(A_i)$
- Allow a special **NULL** value is used to represent values that are ***unknown*** or ***inapplicable*** to certain tuples

Values in Tuple

- All values are considered *atomic* (indivisible):
  - Flat relational model
    - Composite and multivalued attributes not allowed
    - First normal form assumption
  - Multivalued attributes
    - Must be represented by separate relations
  - Composite attributes
    - Represented only by simple component attributes in basic relational model

NULL Values

- **NULL** in a tuple:
  - Represent the values of attributes that may be unknown or may not apply to a tuple
  - Meanings for **NULL** values
    - Value unknown
    - Value exists but is not available
    - Attribute does not apply to this tuple (also known as value undefined)
- IMPORTANT: **NULL** ≠ **NULL** may be NULL due to different causes

Schema Definition in SQL

- The relation schema is

  Customer(customer-name, customer-street, customer-city)

  or

Customer	customer-name	customer-street	customer-city

- The primary key is underlined in the above

```
CREATE TABLE Customer
(
    customer-name     CHAR(20)   NOT NULL,
    customer-street   CHAR(30),
    customer-city     CHAR(30),
    PRIMARY KEY (customer-name)
)
```

Schema Definition in SQL

- To remove a relation from an SQL database, we use the **drop table** command:

  drop table r

- We use the alter table command to add or delete attributes to an existing relation
- All records in the relation are assigned null for a new attribute.

  alter table customer **add** phone char(10)

  alter table customer **drop** phone

Summary

Informal Terms	Formal Terms
Table	Relation
Column Header	Attribute
All possible Column Values	Domain
Row	Tuple
Table Definition	Schema of a Relation
Populated Table	State of the Relation

Outline

- **Relational Model**
  - Relational Model Concepts
  - **Relational Database Schemas**
- ER-to-Relational Mapping
  - Translating traditional ER diagrams
  - Translating Class Hierarchy

Relational Databases

- **Relational database schema** S
  - Set of relation schemas $S = \{R_1, R_2, ..., R_m\}$
  - Set of integrity constraints (ICs)
- **Relational database state**
  - Set of relation states $DB = \{r_1, r_2, ..., r_m\}$ $r \subseteq dom(A_1) \times \dots \times dom(A_n)$
  - Each r_i is a state of R_i and such that the r_i relation states satisfy integrity constraints specified in IC
  - **Invalid state**: Does not obey all the integrity constraints ICs
  - **Valid state**: Satisfies all the constraints in the defined set of integrity constraints ICs

Relational Integrity Constraints (IC)

- Constraints are **conditions** that must hold on **all** valid relation states.
- There are three *main types* of constraints in the relational model:
  - **Domain** constraint
    - Every value in a tuple must be from the *domain of its attribute* (or it could be **NULL**, if allowed for that attribute)
  - **Key** constraints
  - **Entity integrity** constraints
  - **Referential integrity** constraints ?
- ICs are specified when the schema is defined
- ICs are checked when relations are modified

Domain Constraints

- The value of an attribute is limited to its domain.
- A domain can impose rules on both formats and valid value ranges.
  - A salary value cannot be negative
  - An employee's name cannot be **NULL**
    - This is called the **NOT NULL** constraint
- Example:

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

```
CREATE DOMAIN hourly-wage NUMERIC(5,2)
CONSTRAINT wage-value-test CHECK (value > 4.00)
```

- The domain hourly-wage is a decimal number with 5 digits, 2 of which are placed after the decimal point
- The domain has a constraint that ensures that the hourly wage is greater than 4.00

Key Constraints

- Certain minimal subset of the fields (candidate key) of a relation is a unique identifier for a record
- 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
- **Key** of R:
  - Is a set of attributes K of R with the following condition:
    - No two tuples in any valid relation state r(R) will have the same value for K (that is, for any distinct tuples t1 and t2 in r(R), $t1[K] \neq t2[K]$) no duplicate key tuples
- **Candidate Key** of R:
  - A "minimal" key

Key Constraints

- If a relation has several **candidate keys**, one is chosen to be the **primary key**
  - The primary key attributes are <u>underlined</u>
- The primary key is used to *reference* the tuple from another tuple
  - General rule:
    - choose as primary key the smallest of the candidate keys (in terms of space)

      space
    - choice is sometimes subjective

Key in SQL

- 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)

Entity Integrity

minimal set of key ⇒ CK(single attribute) $\xrightarrow{\text{one of them}}$ PK(single attribute)

- The *primary key attributes* PK of each relation schema R in S **CANNOT** have **NULL** values in any tuple of r(R)
  - This is because primary key values are used to *identify* the individual tuples
  - t[PK] ≠ **NULL** for any tuple t in r(R)
  - If PK has several attributes, **NULL** is not allowed in any of these attributes

No primary key value can be **NULL**

Referential Integrity

- Referential Integrity (RR) constraint specified between two relations. Maintains consistency among tuples in two relations
- Tuples in the **referencing relation** R1 have attributes FK (called **foreign key** attributes) that reference the **primary key** attributes PK of the **referenced relation** R2
  - A tuple t1 in R1 is said to **reference** a tuple t2 in R2 if t1[FK] = t2[PK]
- A RR constraint can be displayed in a relational database schema as a directed arc **FROM** R1.FK **TO** R2.PK.

Foreign Key Constraint

- A **foreign key** is a set of attributes in one relation *R1* that is used to refer to a tuple in another relation *R2*
- Statement of the constraint
  - The value in FK of the **referencing relation** R1 can be **either**:
    - (1) a value of an existing primary key value of a corresponding primary key PK in the **referenced relation** R2, <u>or</u>
    - (2) a **NULL**
  - In case (2), the FK in R1 should **NOT** be a part of its own primary key (because of Entity Integrity Constrain)

Example 1: Relational Schema Diagram

≠ NULL

- Student(Student-id, Student-Name) no foreign keys

relation

entity

- Take(Student-id, Course-id, semesterNo) PK: (stu_id, course_id)

≠ NULL

- Course(Course-id, Course-Name) no foreign keys
- (Student-id, Course-id) in relation Take is a primary key
- Student-id, course-id in relation Take are foreign keys

≠NULL

Student	stu_id	stu_name

no foreign keys

foreign key$_1$ = Take.stu.id → student.stu_id

- Draw a relational schema diagram specifying the foreign keys for this schema

Take	stu_id	course_id	semester_NO

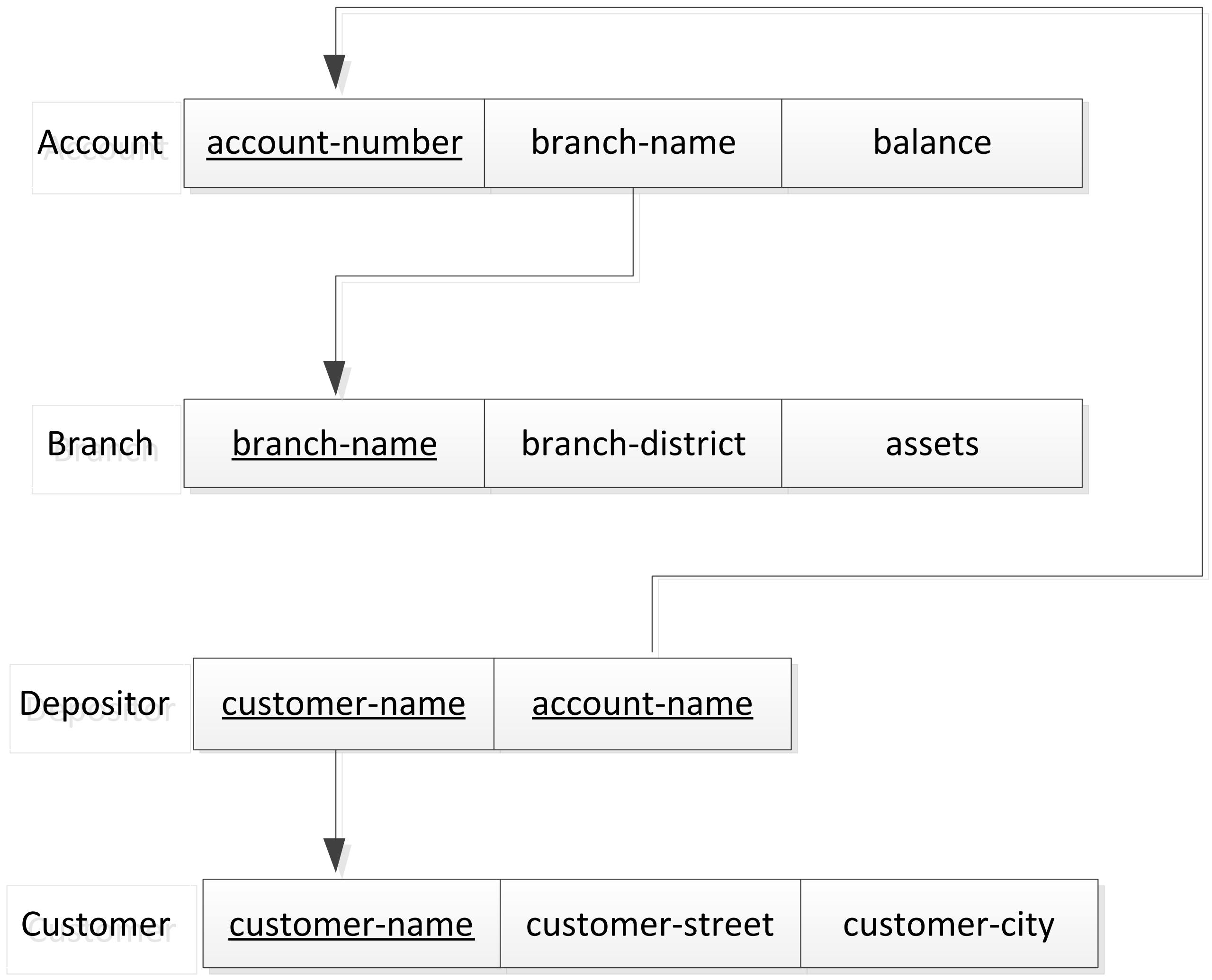
FK$_2$ = Take.course_id → course.course_id

≠NULL

Course	course_id	course_name

no foreign keys

Example 2: Foreign Key



Example 2: Foreign Key

Account

account-number	branch-name	balance

```
CREATE TABLE account
(
    account-number   CHAR(10)   NOT NULL,
    branch-name      CHAR(15),
    balance          INTEGER,
    PRIMARY KEY (account-number)
    FOREGIN KEY (branch-name) REFERENCES branch
)
```

Branch

branch-name	branch-district	assets

Example 2: Foreign Key

Depositor

customer-name	account-number

```sql
CREATE TABLE depositor
(
   customer-name    CHAR(20)  NOT NULL,
   account-number   CHAR(10)  NOT NULL,
   PRIMARY KEY (customer-name, account-number),
   FOREIGN KEY (customer-name)  REFERENCES customer
   FOREIGN KEY (account-number) REFERENCES account
)
```

Account

account-number	branch-name	balance

Customer

customer-name	customer-street	customer-city

Populated Database State

- Each *relation* will have many tuples in its current relation state
- The **relational database state** is a union of all the individual relation states
- Whenever the database is changed, a new state arises
- Basic operations for changing the database:
  - INSERT a new tuple in a relation
  - DELETE an existing tuple from a relation
  - MODIFY an attribute of an existing tuple

Possible Violations for INSERT

- Domain constraint:
  - if one of the attribute values provided for the new tuple is not of the specified attribute domain
- Key constraint:
  - if the value of a key attribute in the new tuple already exists in another tuple in the relation
- Referential integrity:
  - if a foreign key value in the new tuple references a primary key value that does not exist in the referenced relation
- Entity integrity:
  - if the primary key value is null in the new tuple

Example:

Account(account-number, branch-name. balance)
Branch(branch-name, branch-district, assets)
Depositor(customer-name, account-number)
Customer(customer-name, customer-street, customer-city)

cannot add (KLN, 111, 3) to Account

cannot add (Chris, 222) to Depositor

cannot add (Mary, 999) to Depositor

Customer

customer-name	customer-street
Tim	Main
Smith	North
Mary	North

Branch

branch-name	branch-district	assets
CUHK	Shatin	1000
CMTR	Central	2000
SKCR	Shatin	4000

Account

branch-name	account-number	balance
CUHK	222	2
CUHK	777	5
CMTR	333	1
SKCR	444	5
SKCR	888	5

Depositor

customer-name	account-number
Tim	222
Mary	888
Tim	444
Smith	777
Tim	333

Possible Violations for DELETE

- Referential constraints
  - If the primary key value of the tuple being deleted is referenced from other tuples in the database
- Options of remedies:
  - **RESTRICT**: reject the deletion
  - **CASCADE**: delete the record in the referencing table
  - **SET NULL**: set the foreign keys of the referencing tuples to NULL
  - **SET DEFAULT**: set the foreign keys of the referencing tuples to default value

Example:

Account(account-number, branch-name. balance)
Branch(branch-name, branch-district, assets)
Depositor(customer-name, account-number)
Customer(customer-name, customer-street, customer-city)

Cannot simply delete Customer Tim

Cannot simply delete branch CUHK

Customer

customer-name	customer-street
Tim	Main
Smith	North
Mary	North

Branch

branch-name	branch-district	assets
CUHK	Shatin	1000
CMTR	Central	2000
SKCR	Shatin	4000

Account

branch-name	account-number	balance
CUHK	222	2
CUHK	777	5
CMTR	333	1
SKCR	444	5
SKCR	888	5

Depositor

customer-name	account-number
Tim	222
Mary	888
Tim	444
Smith	777
Tim	333

Example: Foreign Key

- Cascading delete:
  - Delete Customer Tim, cascaded delete on depositor
  - Delete branch CUHK, cascaded delete on account

```
CREATE TABLE account
(
    branch-name        CHAR(15),
    account-number     CHAR(10)     NOT NULL,
    balance            INTEGER,
    PRIMARY KEY (account-number),
    FOREIGN KEY (branch-name)   REFERENES branch
                            ON DELETE CASCADE
                            ON UPDATE CASCADE
)                          [ON UPDATE NO ACTION]
```

Example: Foreign Key

Branch	branch-name	branch-district	assets

```
CREATE TABLE branch (
    branch-name        CHAR(20)    DEFAULT 'CUHK',
    branch-district    CHAR(30),
    assets             INTEGER,
    PRIMARY KEY (branch-name) )
```

```
CREATE TABLE account (
    branch-name        CHAR(15)    DEFAULT 'CUHK',
    account-number     CHAR(10)    NOT NULL,
    balance            INTEGER,
    PRIMARY KEY (account-number),
    FOREGIN KEY (branch-name)  REFERENCES branch
                ON DELETE CASCADE
                ON UPDATE CASCADE )
               [ON DELETE SET DEFAULT]
               [ON DELETE SET NULL]
```

Possible Violations for UPDATE

- Constraint violations depending on the attribute being updated:
  - Updating the primary key (PK):
    - Similar to a DELETE followed by an INSERT
    - Need to specify similar options to DELETE
  - Updating a foreign key (FK):
    - May violate referential integrity
  - Updating an ordinary attribute (neither PK nor FK):
    - Can only violate domain and business rules constraints

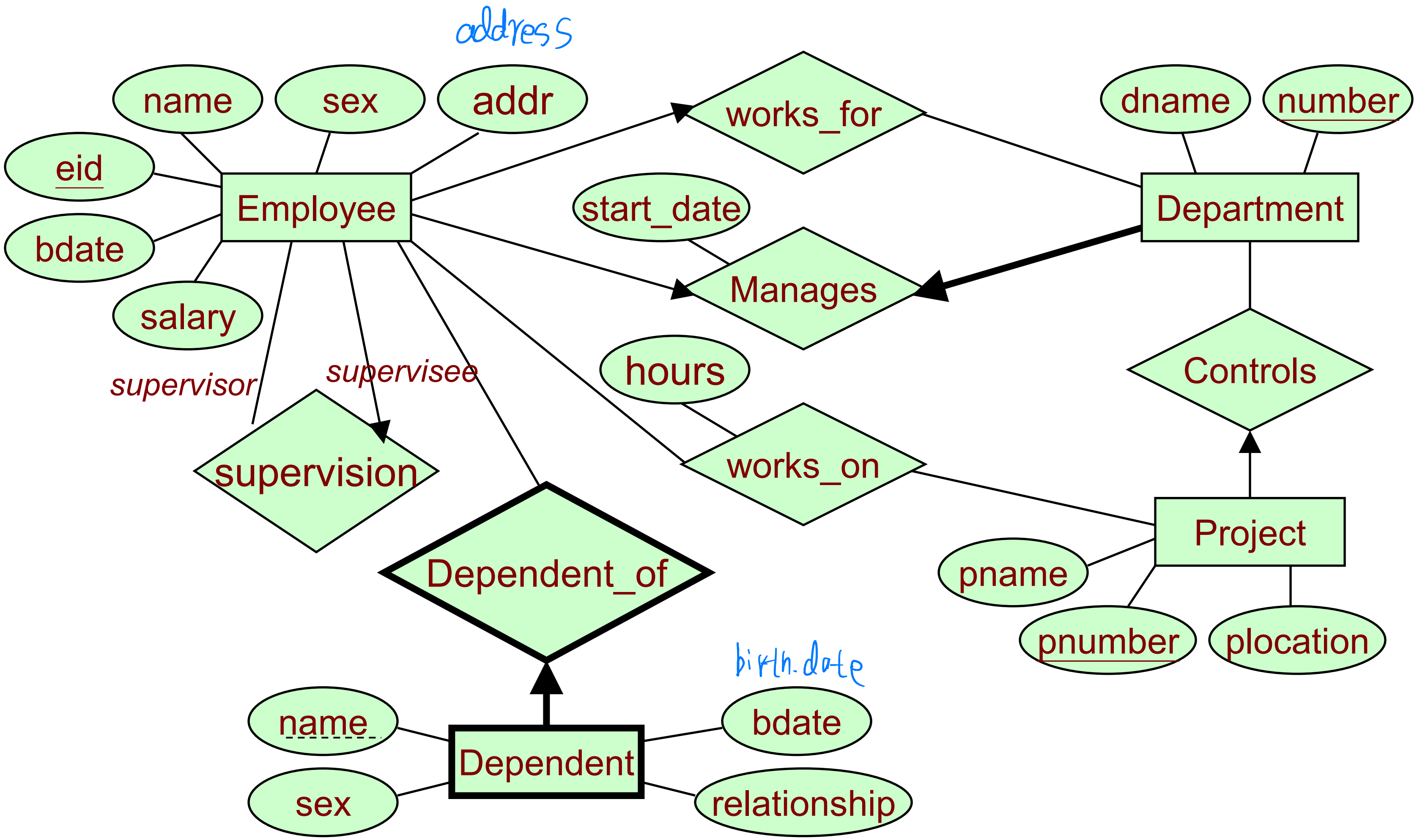
Outline

- Relational Model
  - Relational Model Concepts
  - Relational Database Schemas
- **ER-to-Relational Mapping**
  - **Translating traditional ER diagrams**
  - Translating Class Hierarchy

ER-to-Relational Mapping

- Typically, database designers begin with the ER model, which is very **expressive** and **user-friendly** to human
- Then, the ER model is mapped to the relational model for DBMS manipulations
- Database queries and updates will be written according to the relational model

Running Example



ER to Relational Model

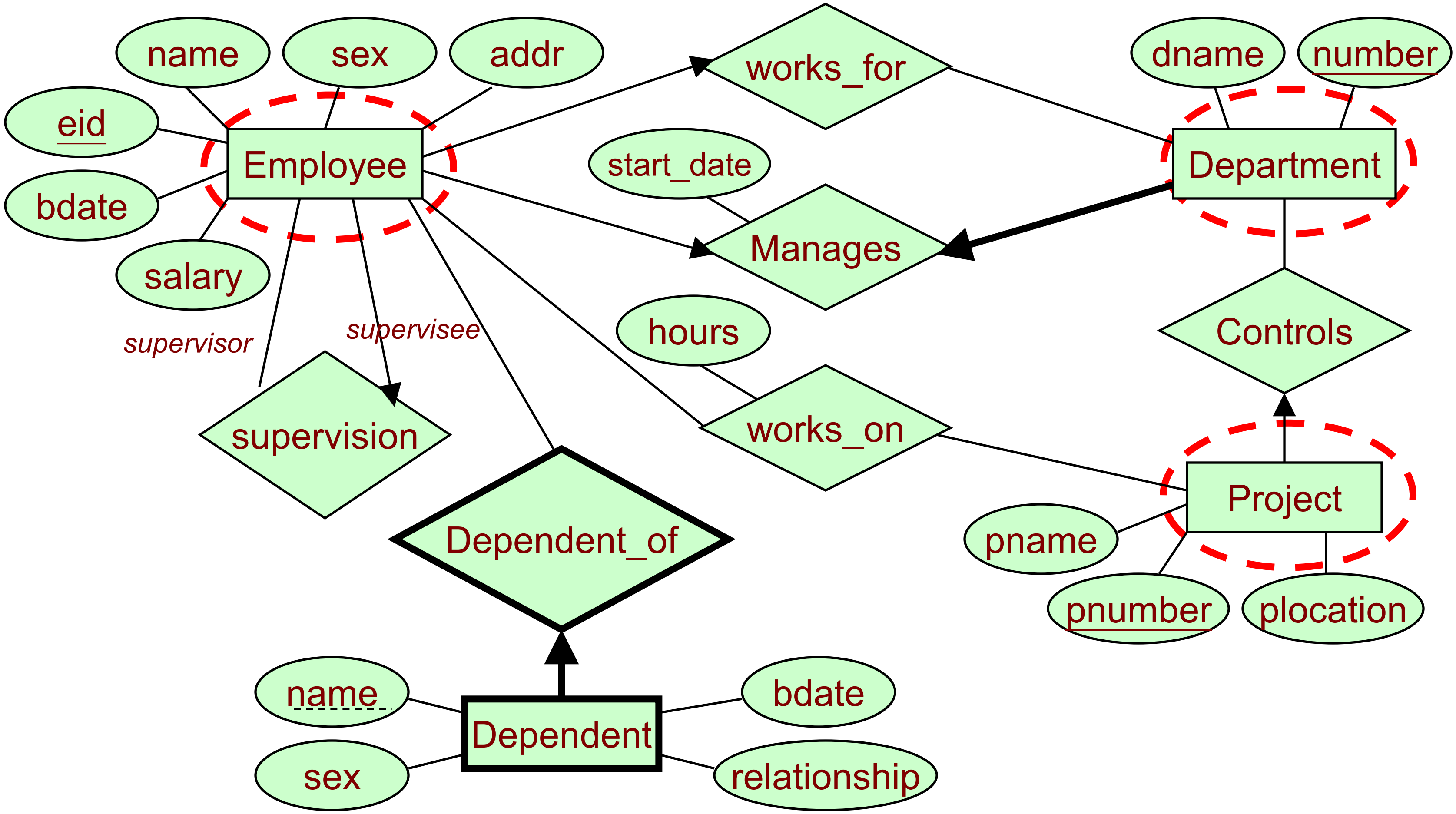
1. Convert entities first
   - From $\boxed{\text{Strong Entity}}$ to $\boxed{\text{Weak Entity}}$
2. Convert relations
   - From 1-to-1, 1-to-many (many-to-1) to many-to-many
   - From binary relation to non-binary relation

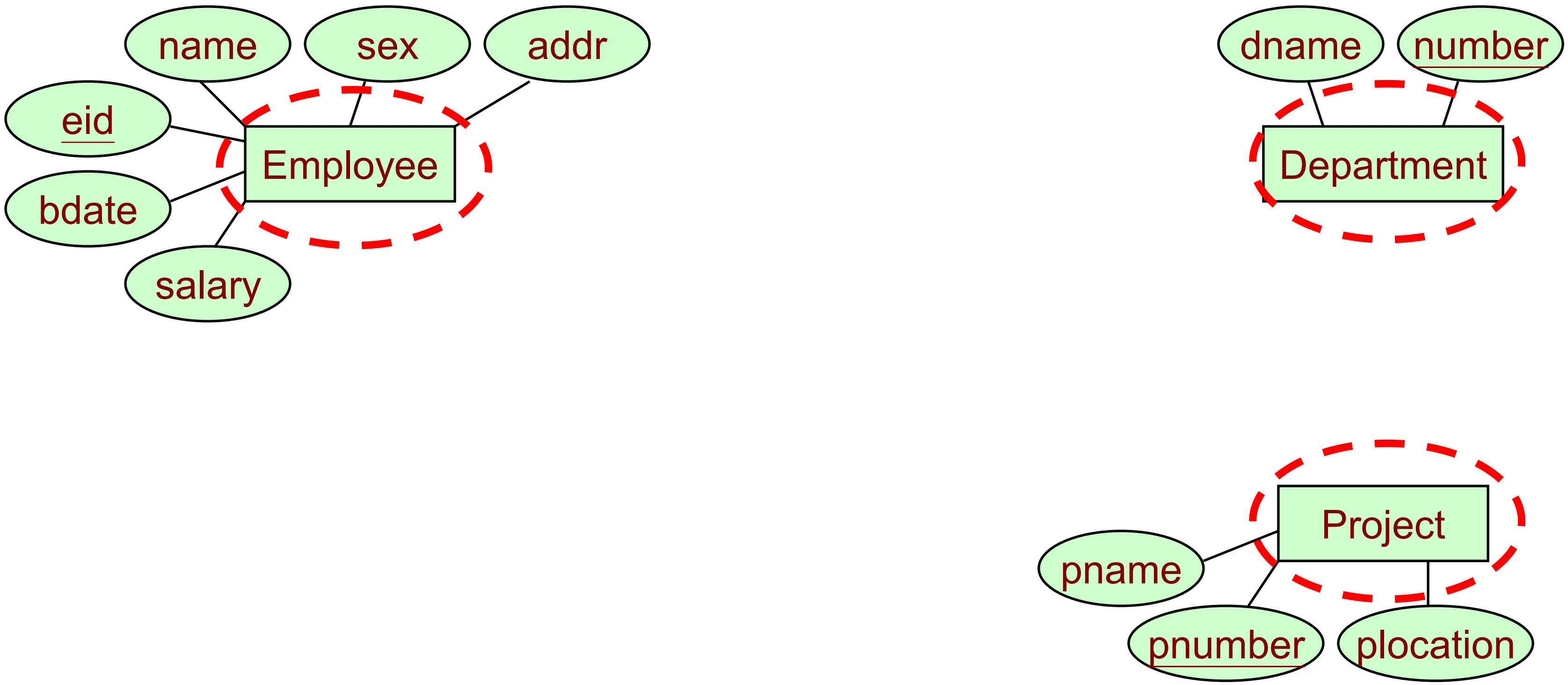
Steps

- ER-to-Relational Mapping Algorithm
  - **Step 1:** Mapping of Regular Entity Types
  - **Step 2:** Mapping of Weak Entity Types
  - **Step 3:** Mapping of Binary 1:1 Relation Types
  - **Step 4:** Mapping of Binary 1:N Relationship Types.
  - **Step 5:** Mapping of Binary M:N Relationship Types.
  - **Step 6:** Mapping of N-ary Relationship Types.

Step 1 (Strong Entity)

- For each strong entity set E in the ER schema,
  - create a relation schema R that includes all the attributes of E
  - choose one set of key attributes of E as a primary key for R
  - if the chosen key of E is composite, the set of simple attributes that form it will together form the primary key of R


name
sex
addr
eid
bdate
salary
Employee
works_for
dname
number
Department
start_date
Manages
supervisor
supervisee
hours
Controls
supervision
works_on
Project
pname
pnumber
plocation
Dependent_of
name
sex
Dependent
bdate
relationship


name
sex
addr
eid
Employee
bdate
salary
dname
number
Department
Project
pname
pnumber
plocation

Step 1 (Strong Entity)

- Example
- We create the relation schemas EMPLOYEE, DEPARTMENT and PROJECT

EMPLOYEE

name	eid	bdate	addr	sex	salary

DEPARTMENT

dname	dnumber

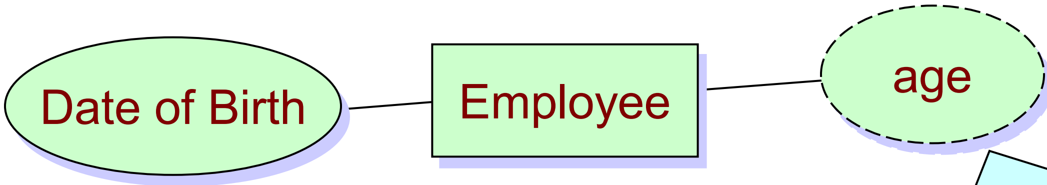
PROJECT

pname	pnumber	plocation

Step 1 (Strong Entity)

(omitted)

- If there is a derived attribute, what should we do?



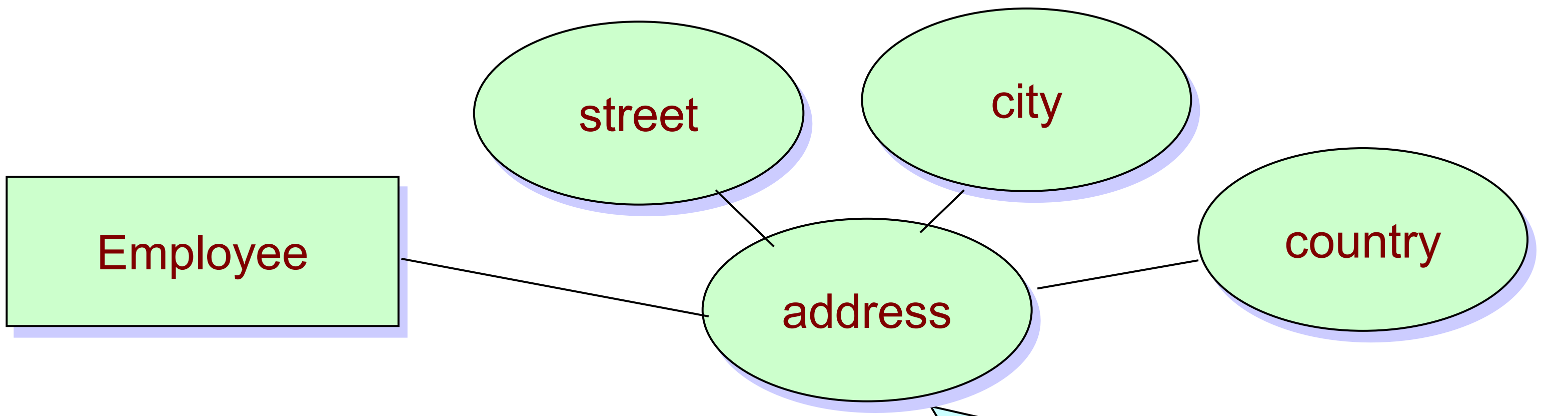
We have two choices.

Choice 1: Include this derived attribute
Adv: We can directly obtain the value of the derived attribute
Disadv: We may encounter some data inconsistencies

Choice 2: NOT include this derived attribute
Adv: We can avoid data inconsistency
Disadv: We need to perform some operations to obtain the value of the derived attribute

Step 1 (Strong Entity)

- If there is a composite attribute, what should we do?



We have two choices.

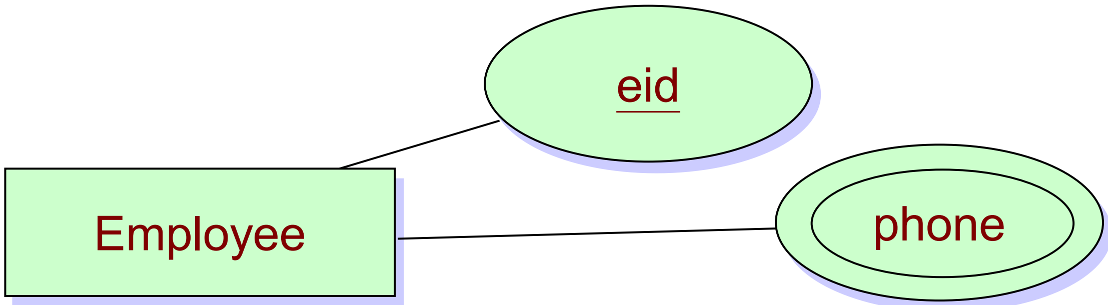
Choice 1: Include the high-level attribute only (e.g., address)

Choice 2: Include all low-level attributes (e.g., street, city, country)

prefer

Step 1 (Strong Entity)

- If there is a multi-valued attribute, what should we do?



We have two choices.

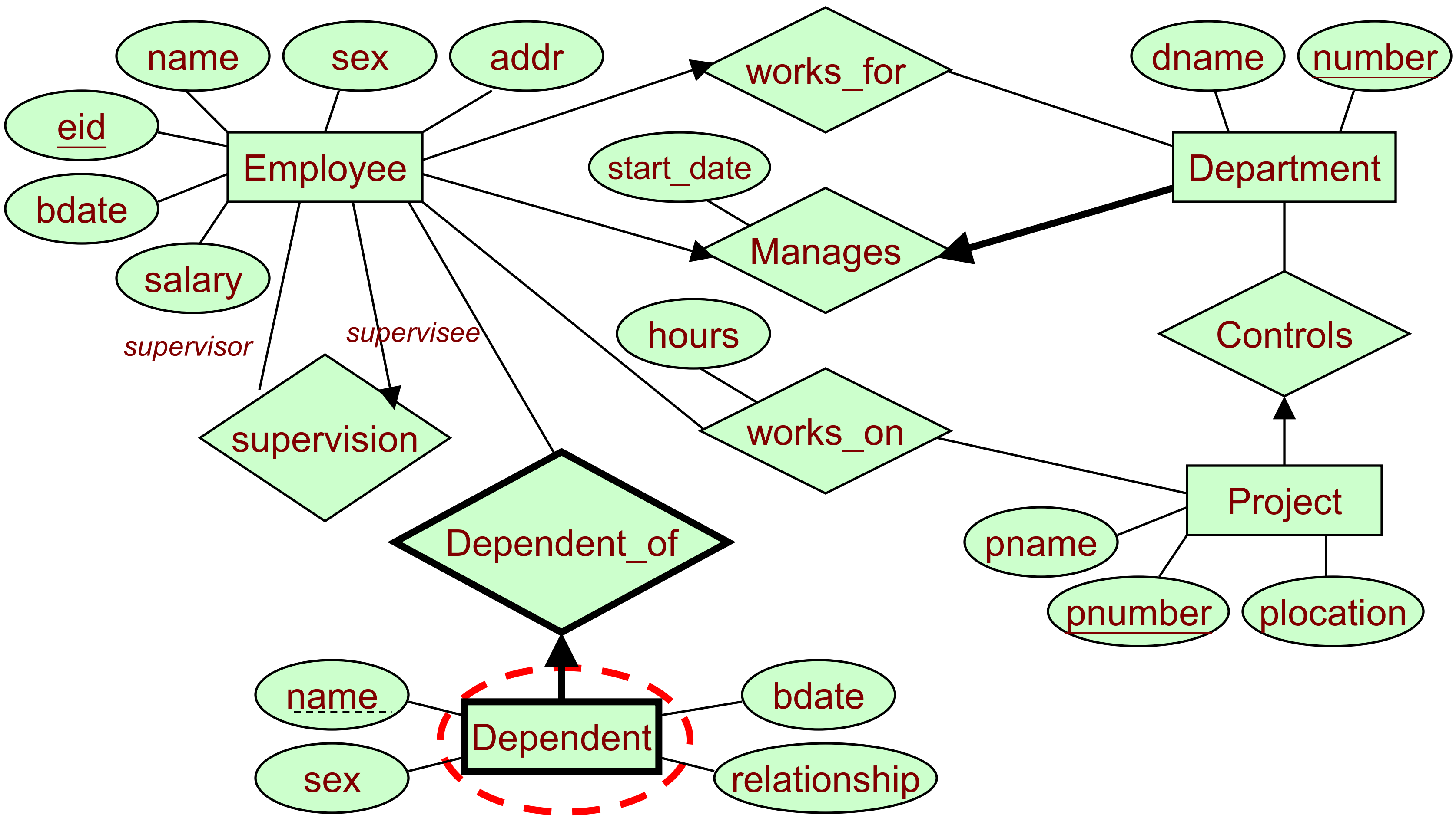
Choice 1: Include one attribute only (e.g., phone)

Choice 2: Create another table containing the primary key of the entity set and the multi-valued attribute. e.g., create a schema PhoneTable (eid, phone) foreign key

Step 2 (Weak Entity)

- For each **weak entity set** W in the ER model,
  - Create a relation schema R, and include all attributes
  - In addition, include the primary key(s) of the owner(s)
  - The primary key of R is the combination of the primary key(s) of the owner(s) and the discriminator of the weak entity set W

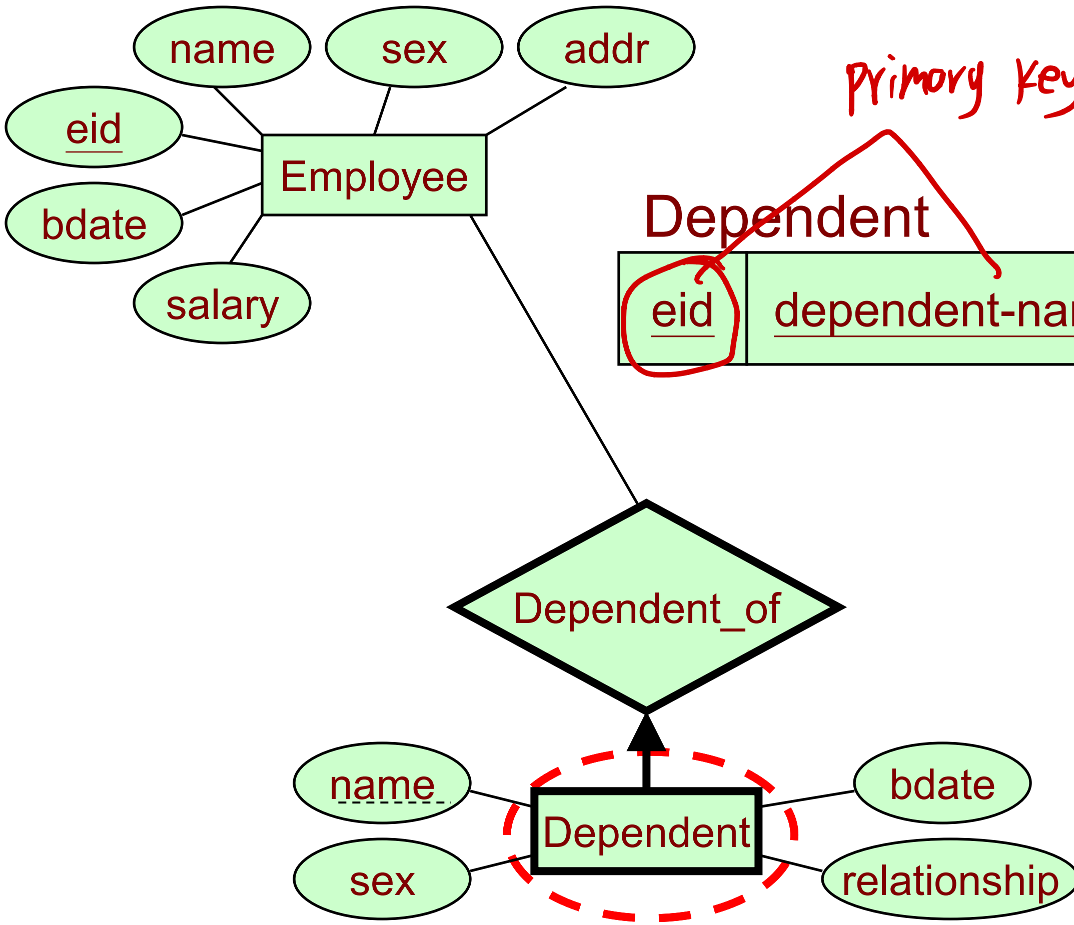
partial key


name
sex
addr
eid
bdate
salary
Employee
works_for
start_date
Manages
dname
number
Department
Controls
hours
works_on
supervisor
supervisee
supervision
Dependent_of
Project
pname
pnumber
plocation
name
sex
Dependent
bdate
relationship

Primary key

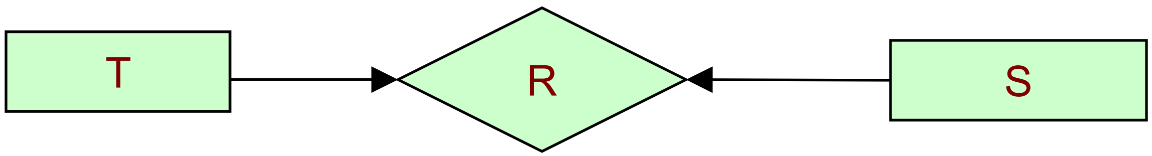
Dependent

<u>eid</u>	<u>dependent-name</u>	sex	bdate	relationship

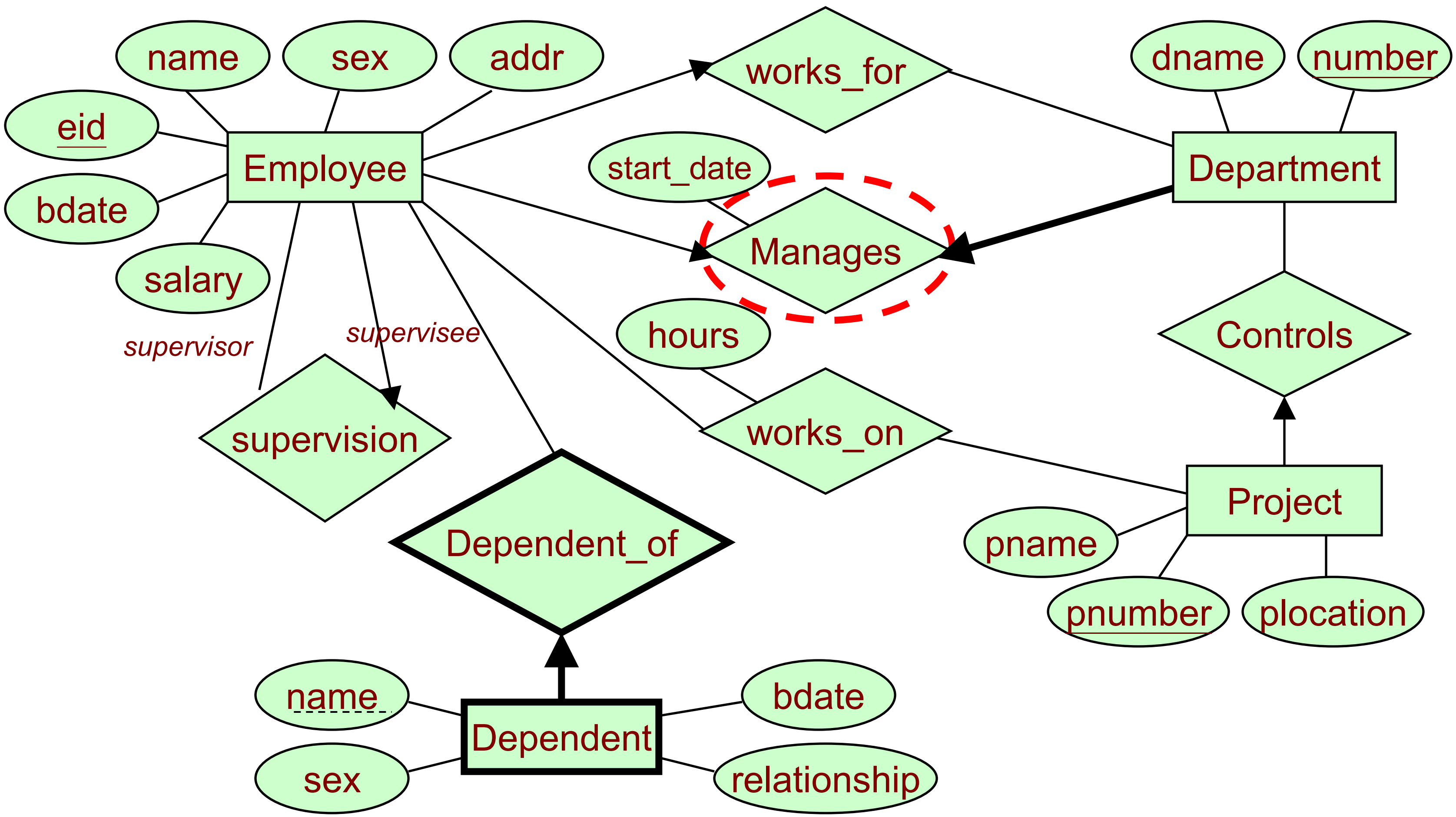


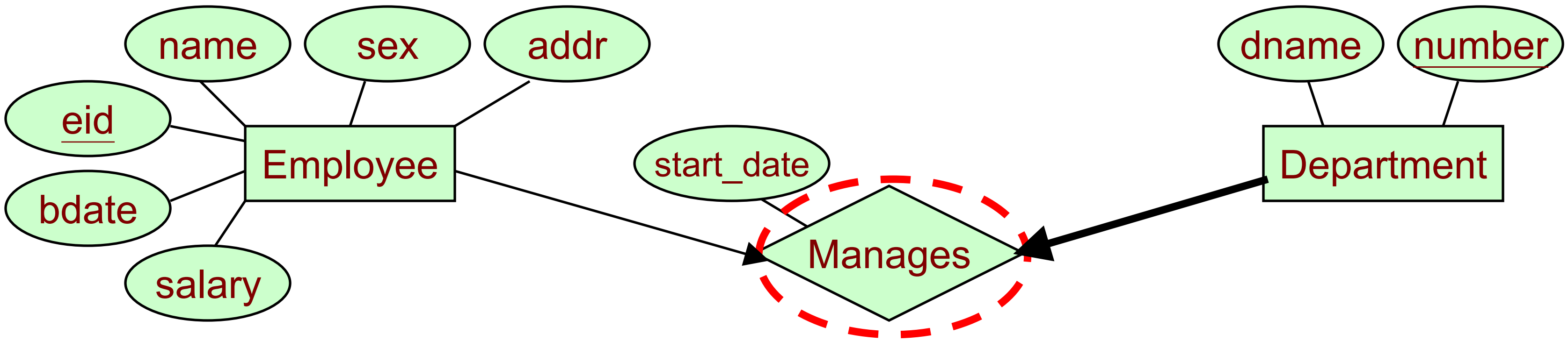
Step 3 (1-to-1 Relationship)

- For each binary one-to-one (1:1) relationship set R



  - Choose one of the 2 relation schemas, say S,
    - get primary key of T,
    - include it as foreign keys in S
  - Better if S has total participation in R
  - Include the attributes of the relationship set R as attributes of S


name
sex
addr
eid
Employee
bdate
salary
supervisor
supervisee
supervision
works_for
start_date
Manages
hours
works_on
dname
number
Department
Controls
Project
pname
pnumber
plocation
Dependent_of
name
sex
Dependent
bdate
relationship



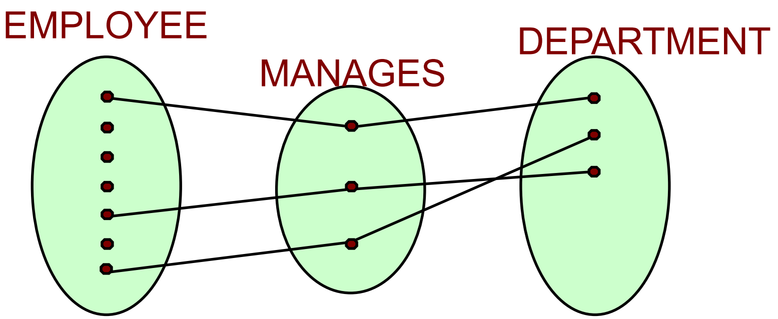
eid

- We include the **primary key** of EMPLOYEE as foreign key in DEPARTMENT and rename it **mgr_id**
- We include the attribute **startdate** of MANAGES and rename it **mgr_start_date**

DEPARTMENT

dname	dnumber	mgr_id	mgr_start_date

Compare the following two choices to include MANAGES:



Add information to EMPLOYEE

name	id	bdate	addr	sex	salary	dnum	sdate
Yeung	7	080370	…	F	20K	3	010100
Chan	3	031060	…	M	30K	4	020399
Wong	4	010280	…	F	10K	Null	Null
Cheung	8	220985	…	M	24K	Null	Null

dname	dnumber	mgr_id	mgr_start_date
Personnel	4	3	020399
Marketing	3	7	010100
…	…	…	…

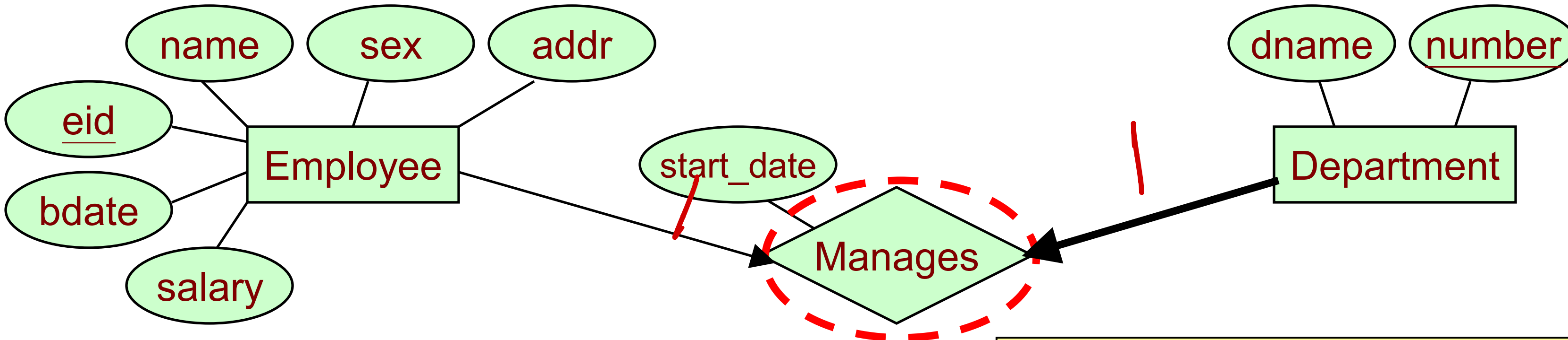
Add to DEPARTMENT

- In the above, the NULL value is a special value meaning that the value is either unknown or not applicable
- Notice that an alternative mapping of a one-to-one relationship set is possible by merging the two entity sets and the relationship into a single relation
- This is appropriate when both participations are total

Mapping 1-to-1 Relationship:

Advantage:
The total number of relations remain unchanged

Disadvantage:
It may store NULL values if there is no total participation



Can we create a new relation

Manages (eid, number, start_date)

or

Manages (eid, number, start_date)

for this relationship?

Yes.

It can be used if there are only a few relationship instances

Advantage:

It can avoid storing NULL values if there is no total participation

Disadvantage:

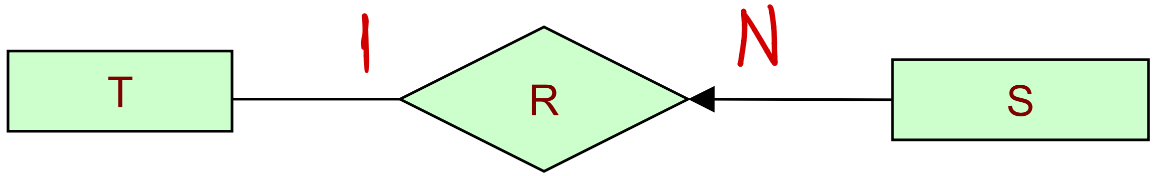
There is one additional relation

- There are three approaches for mapping 1-1 binary relationship:

  1. **Foreign Key approach:** Choose one of the relations-say S-and include a foreign key in S the primary key of T. It is better to choose an entity type with total participation in R in the role of S
  2. **Merged relation option:** An alternate mapping of a 1:1 relationship type is possible by merging the two entity types and the relationship into a single relation. This may be appropriate when both participations are total
  3. **Cross-reference or relationship relation option:** The third alternative is to set up a third relation R for the purpose of cross-referencing the primary keys of the two relations S and T representing the entity types

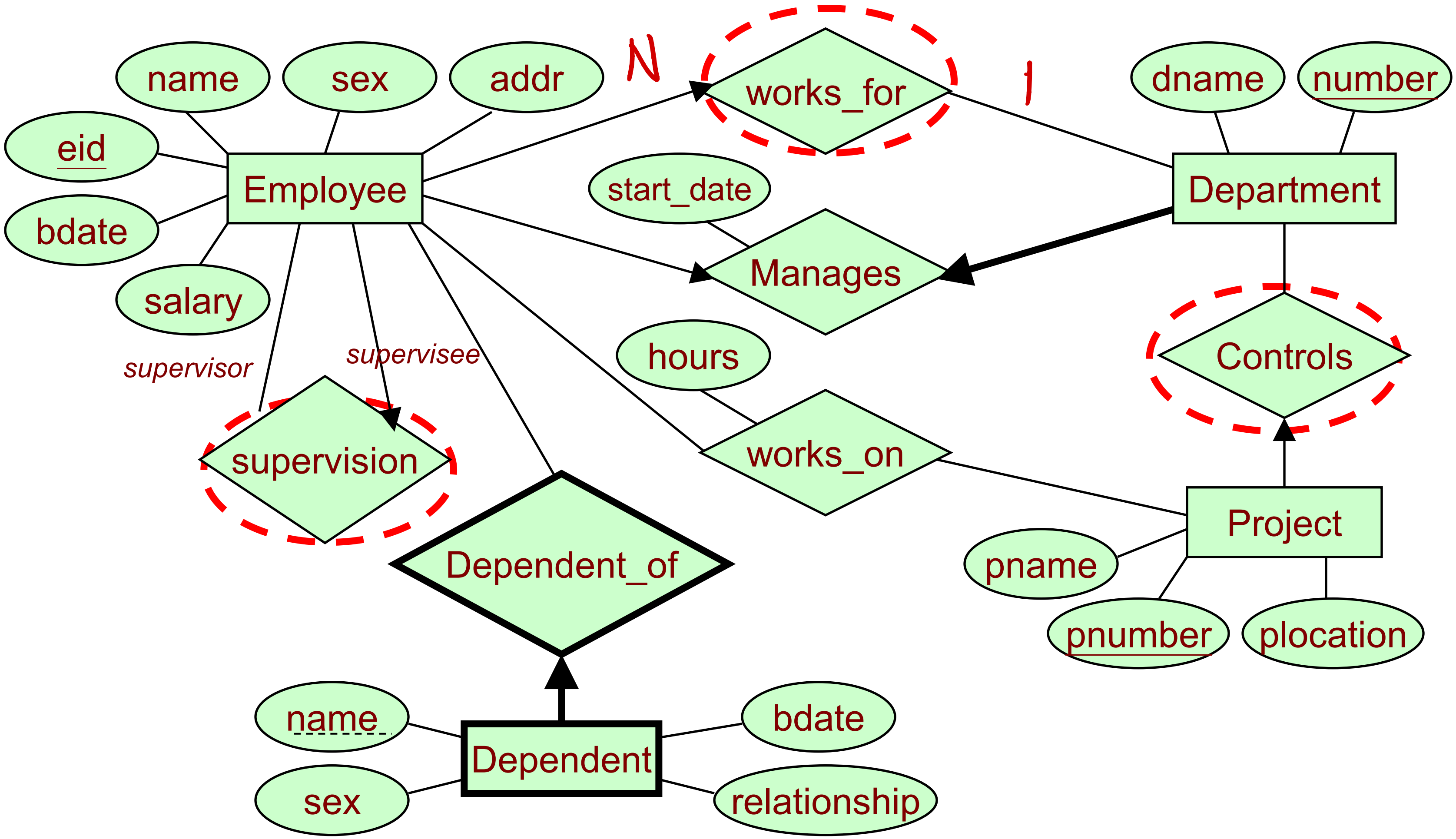
Step 4 (1-to-many Relationship)

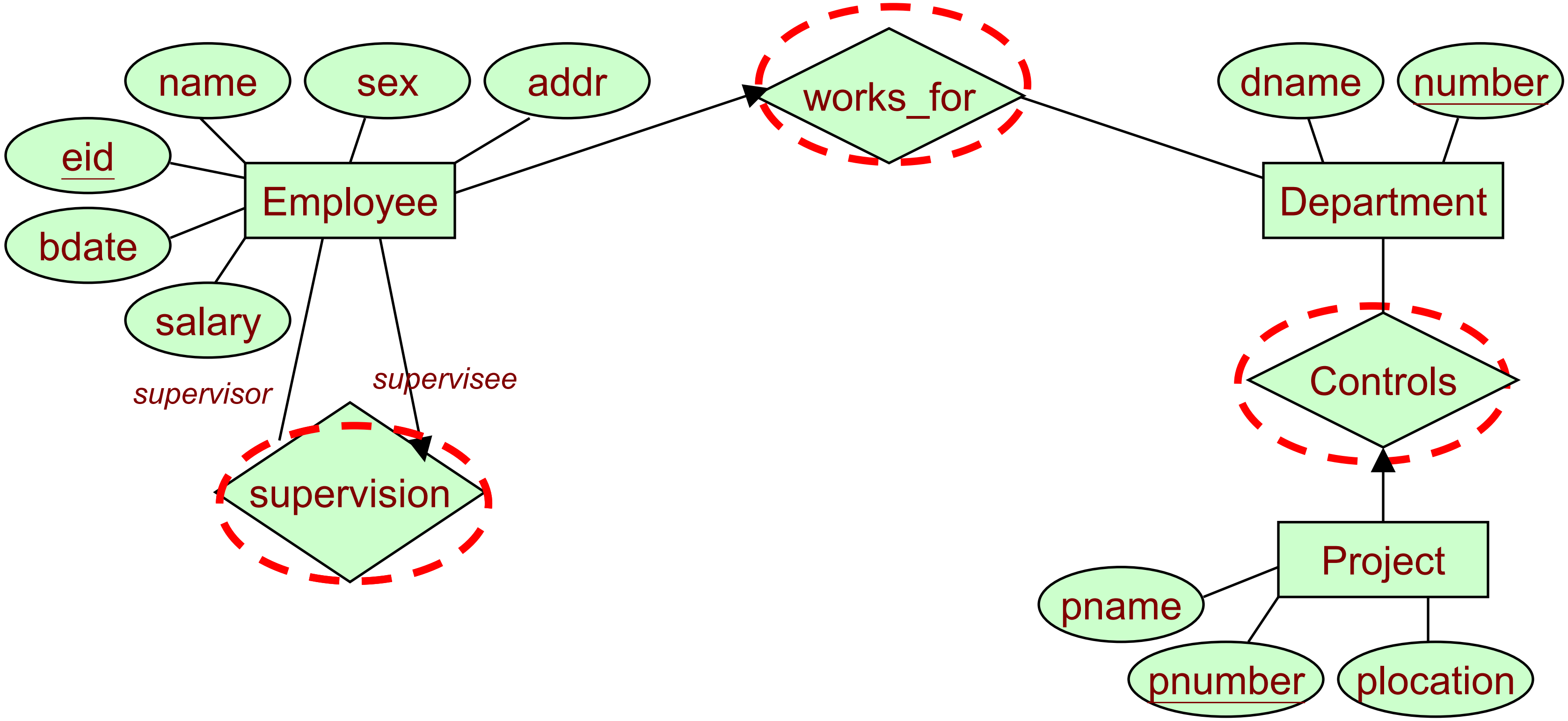
- For each **binary one-to-many** relationship set



  - Include as foreign key in S the primary key that represents the other entity set T participating in R
  - Include any attributes of the one-to-many relationship set as attributes of S
  - In other words, we always choose the one who determines the relationship as R

Add primary key of entity(1) to the one(N) as a foreign key.

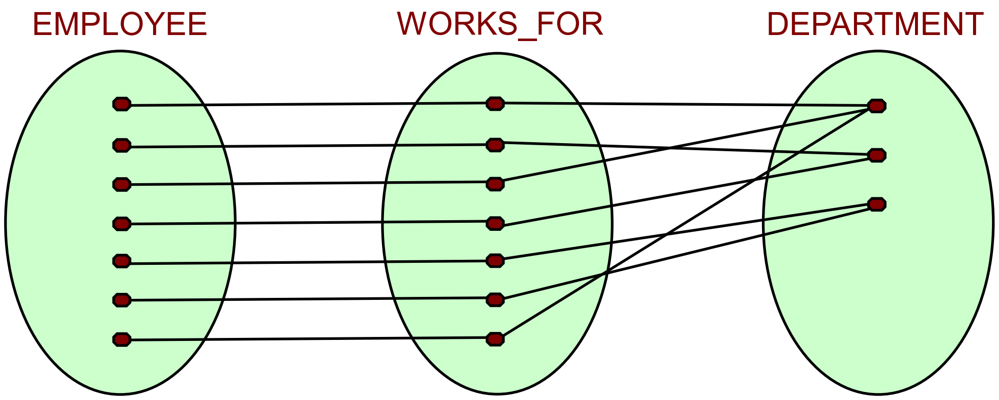

name
sex
addr
eid
Employee
bdate
salary
N
works_for
1
dname
number
Department
start_date
Manages
hours
Controls
supervisor
supervisee
supervision
works_on
Project
pname
pnumber
plocation
Dependent_of
name
Dependent
bdate
sex
relationship


name
sex
addr
eid
Employee
bdate
salary
supervisor
supervisee
supervision
works_for
dname
number
Department
Controls
Project
pname
pnumber
plocation

- The primary key dnumber of the DEPARTMENT relation schema is included as foreign key in the EMPLOYEE relation schema
- We rename it as dno (The renaming is not necessary but makes the name more meaningful.)

EMPLOYEE

name	<u>eid</u>	bdate	addr	sex	salary	dno



Compare the following 2 choices:

Add employees to department

DEPARTMENT

dname	dnumber	mgr_id	mgr_start_date	eid
Personnel	4	3	020399	3
Personnel	4	3	020399	11
Personnel	4	3	020399	12
Personnel	4	3	020399	13
Marketing	3	7	010100	21
Marketing	3	7	010100	7
Marketing	3	7	010100	22
…	…	…	…	…

EMPLOYEE

name	id	bdate	addr	sex	salary	dno
Yeung	7	080370	…	F	20K	3
Chan	3	031060	…	M	30K	4
Wong	4	010280	…	F	10K	7
Cheung	8	220985	…	M	24K	1

Add department to employee

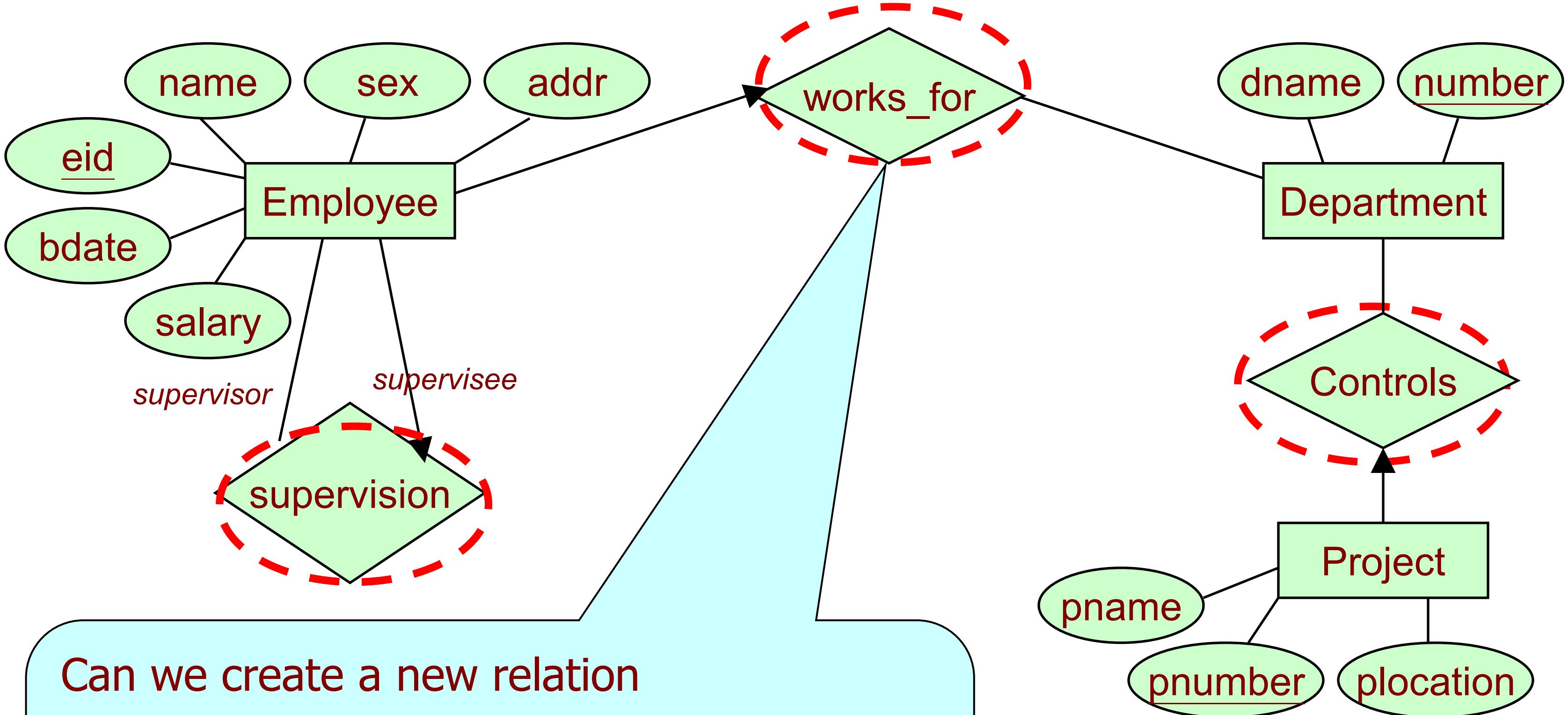
- For SUPERVISON,
  - include the primary key of the EMPLOYEE as foreign key in the EMPLOYEE, and call it super_id

Employee recursive 自己添加自己

EMPLOYEE	name	eid	bdate	addr	sex	salary	dno	super_id

- For CONTROLS relationship,
  - include dnum as foreign key in PROJECT,
  - which references the primary key dnumber of DEPARTMENT

PROJECT	pname	pnumber	plocation	dnum



Can we create a new relation

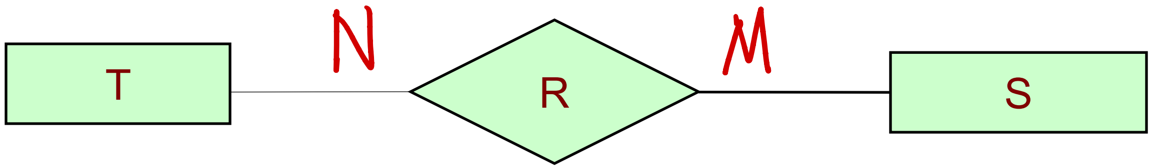
works_for (eid, number)

for this relationship?

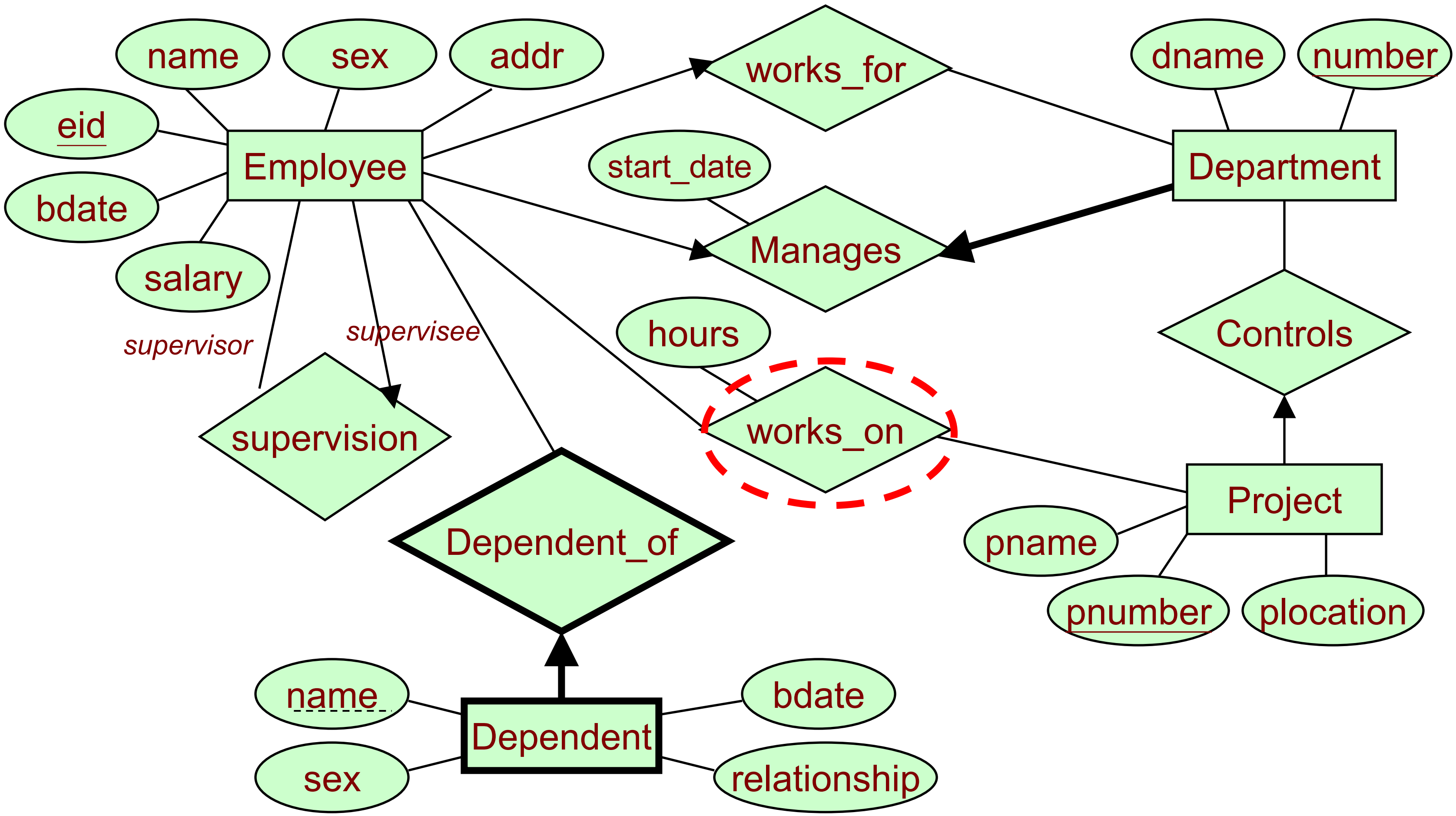
Yes.

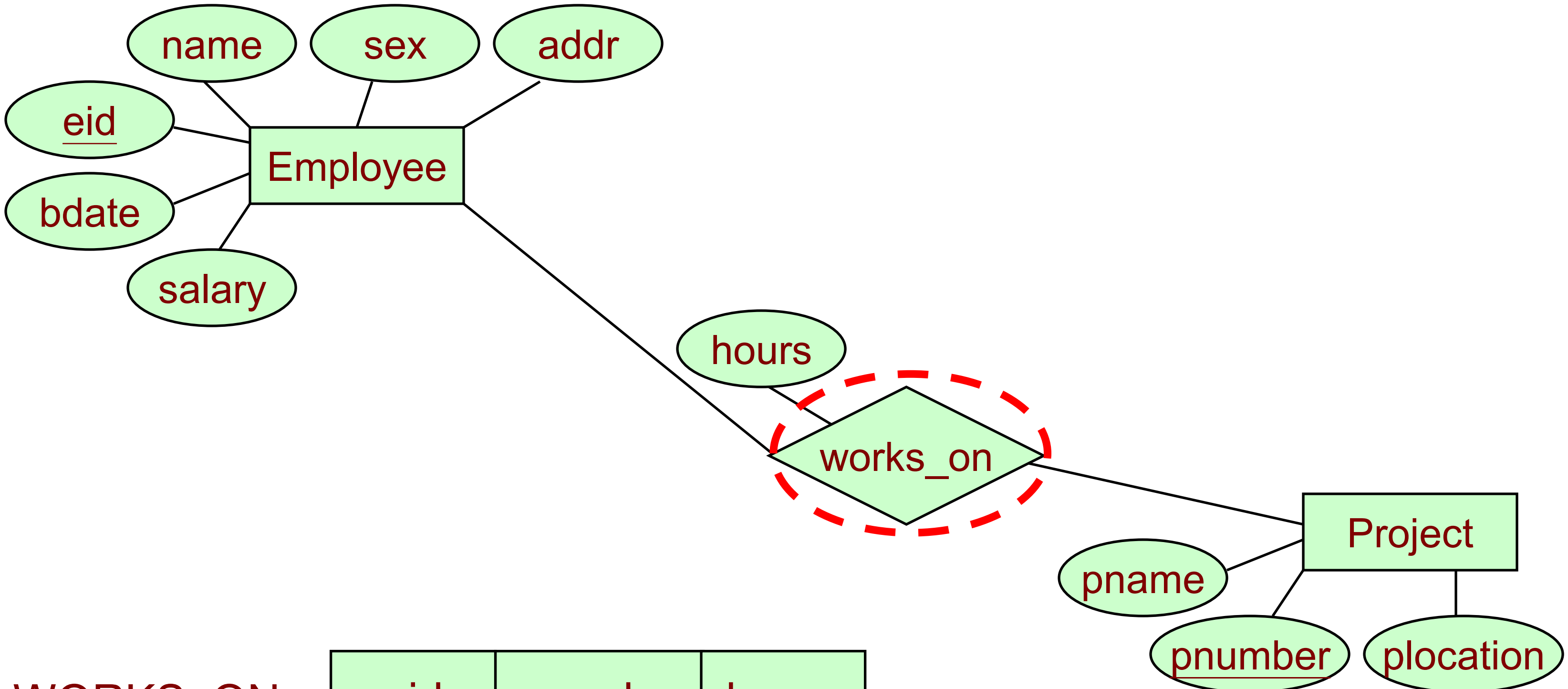
Step 5 (Many-to-many Relationship)

- For each binary many-to-many relationship set R



  - Create a new relation schema S to represent R
  - Include as foreign key attributes in S the primary keys of the relation schemas for the participating entity sets in R
  - Their combination will form the primary key of S
  - Also include attributes of the many-to-many relationship set as attributes of S


name
sex
addr
eid
Employee
bdate
salary
works_for
dname
number
start_date
Department
Manages
supervisor
supervisee
hours
Controls
supervision
works_on
Project
pname
pnumber
plocation
Dependent_of
name
bdate
Dependent
sex
relationship



WORKS_ON

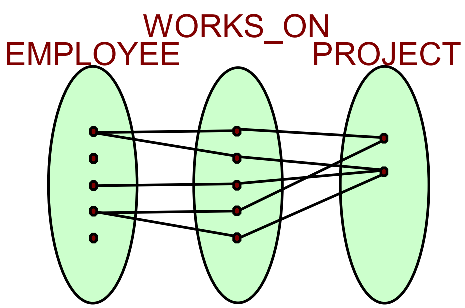
eid	pnumber	hours

Map the many-to-many relationship sets

WORKS_ON by creating the relation schema WORKS_ON

Include the primary keys of PROJECT and EMPLOYEE as foreign keys

Compare the following three choices to include WORKS_ON



Add to EMPLOYEE

name	id	bdate	addr	sex	salary	dno	pnumber	hours
Yeung	7	080370	…	F	20K	3	Null	Null
Chan	3	031060	…	M	30K	4	C77	89
Chan	3	031060	…	M	30K	4	A01	10
Wong	4	010280	…	F	10K	7	A01	101
Cheung	8	220985	…	M	24K	1	A01	22
Cheung	8	220985	…	M	24K	1	C77	57

Add to PROJECT

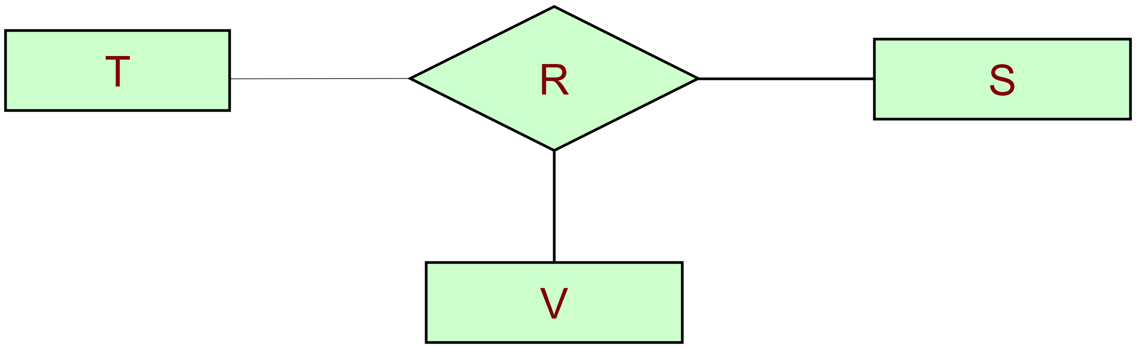
pname	pnumber	plocation	dnum	eid	hours
SmartCard	C77	C-Lab	4	2	89
SmartCard	C77	C-Lab	4	8	10
Robotics	A01	C-Lab	7	2	101
Robotics	A01	C-Lab	7	4	22
Robotics	A01	C-Lab	7	8	57

New relation WORKS_ON

eid	pnumber	hours
2	C77	89
2	A11	10
4	A11	101
8	A11	22
8	C77	57

Step 6 (Non-binary Relationship)

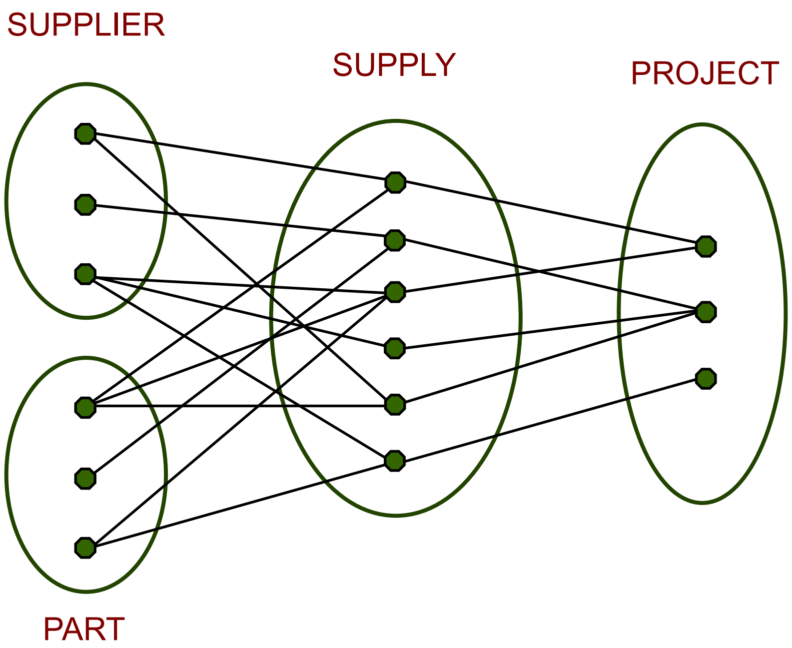
- For each non-binary relationship set

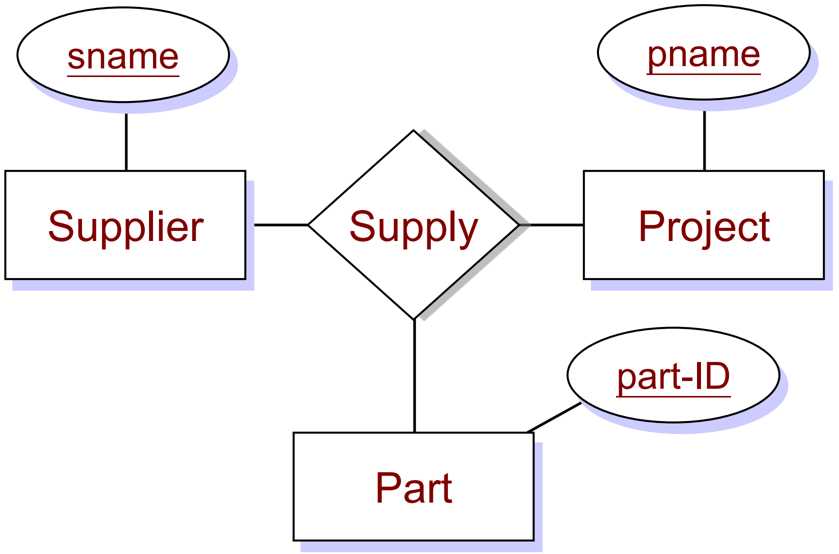


  - Create a new relation schema S to represent R
  - Include as foreign key attributes in S the primary keys of the participating entity sets
  - Also include any attributes of the non-binary relationship set as attributes of S

Ternary Relationship Set

- Create table Supply,
- Attributes are sname, pname, part-ID,
- These also form the key

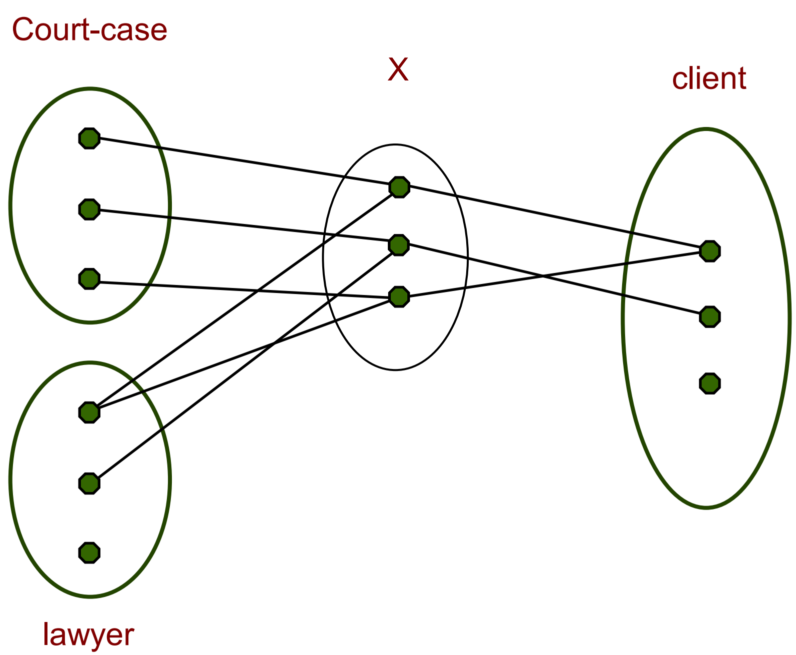


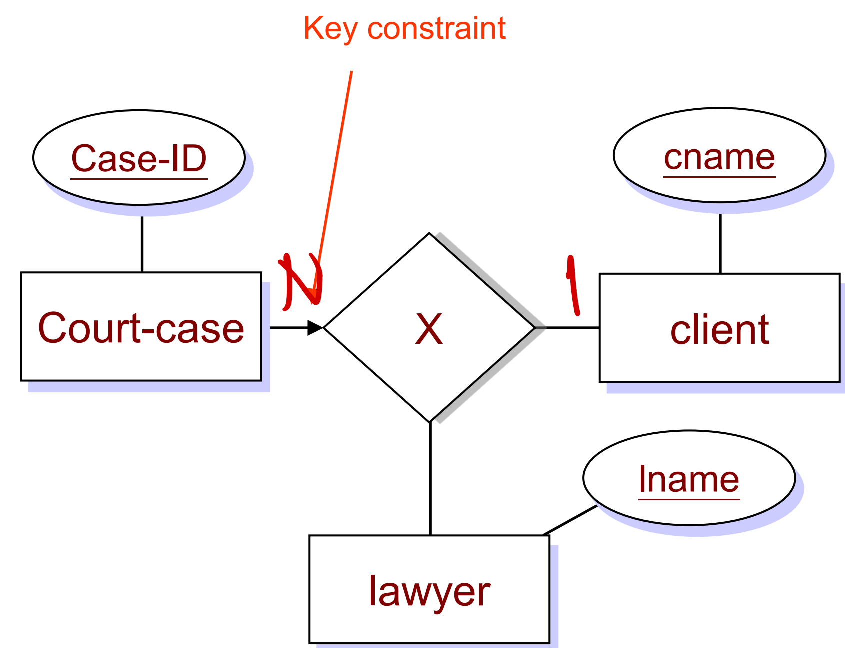


Supply (<u>sname</u>, <u>pname</u>, <u>part-ID</u>)

Ternary Relationship Set

- Create a new table X,
- Attributes are Case-ID, cname, lname,
- Case-ID is the key





X(<u>Case-ID</u>, cname, lname)

Resulting Relation Schemas:

DEPARTMENT

dname	$\underline{\text{dnumber}}$	mgr_id	mgr_start_date

EMPLOYEE

name	$\underline{\text{eid}}$	bdate	addr	sex	salary	dno	super_id

PROJECT

pname	$\underline{\text{pnumber}}$	plocation	dnum

DEPENDENT

$\underline{\text{eid}}$	$\underline{\text{dependent-name}}$	sex	bdate	relationship

WORKS_ON

$\underline{\text{eid}}$	$\underline{\text{pnumber}}$	hours

Summary of Mapping Constructs

Correspondence between ER and Relational Models

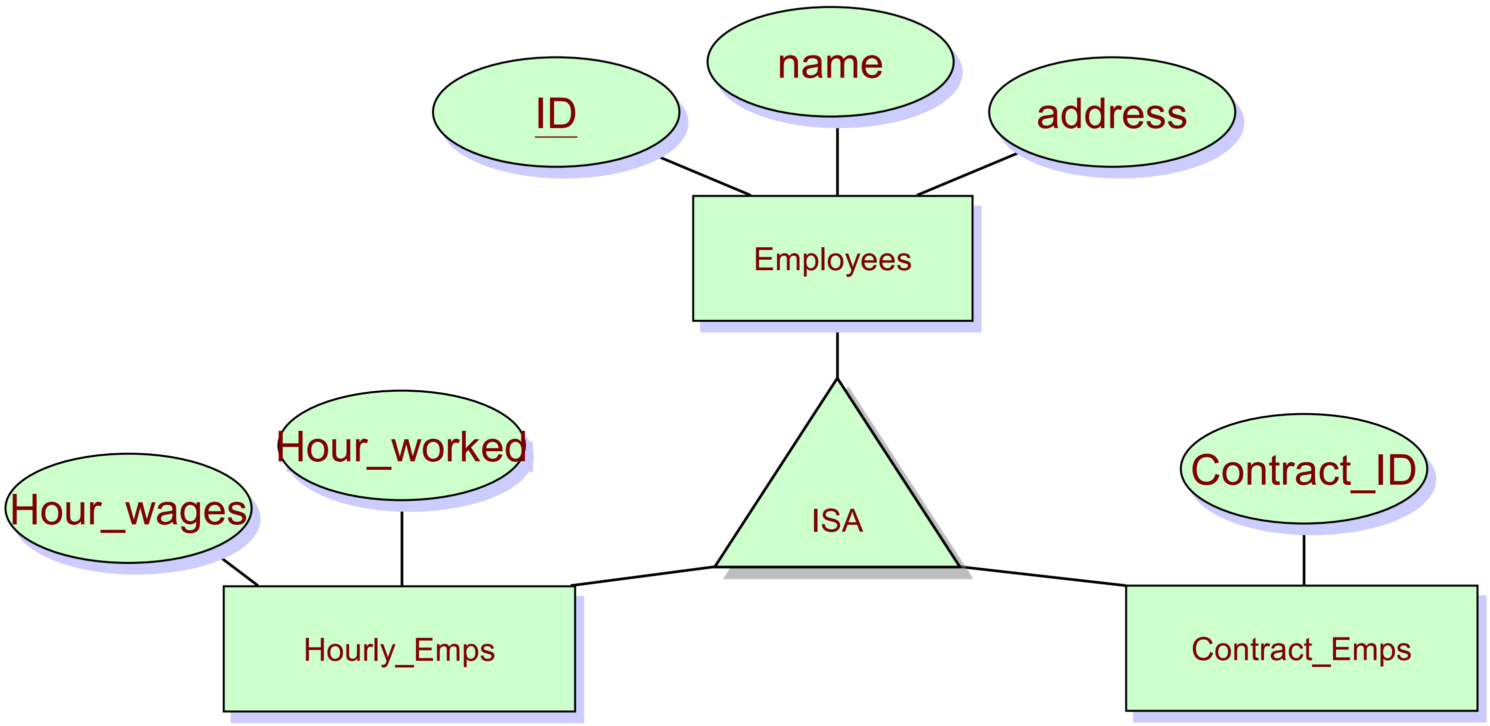
ER Model	**Relational Model**
Entity type	“Entity” relation
1:1 or 1:N relationship type	Foreign key (or “relationship” relation)
M:N relationship type	“Relationship” relation and two foreign keys
n-ary relationship type	“Relationship” relation and n foreign keys
Simple attribute	Attribute
Composite attribute	Set of simple component attributes
Multivalued attribute	Relation and foreign key
Value set	Domain
Key attribute	Primary (or secondary) key

Outline

- Relational Model
  - Relational Model Concepts
  - Relational Database Schemas
- **ER-to-Relational Mapping**
  - Translating traditional ER diagrams
  - **Translating Class Hierarchy**

Translating Class Hierarchy

- Consider the class hierarchy example



Translating Class Hierarchy

- Two possible ways:
  - 1. Map each of the entity sets Employees, Hourly-Emps, and Contract-Emps to a distinct relation

EMPLOYEE

name	id	address

HOURLY-EMP

id	hours_worked	hours_wages

CONTRACT-EMP

id	contract-id

Translating Class Hierarchy

- Two possible ways:
  - 2. Create only two relations

HOURLY-EMP

<u>id</u>	name	address	hours_worked	hours_wages

CONTRACT-EMP

<u>id</u>	name	address	contract-id

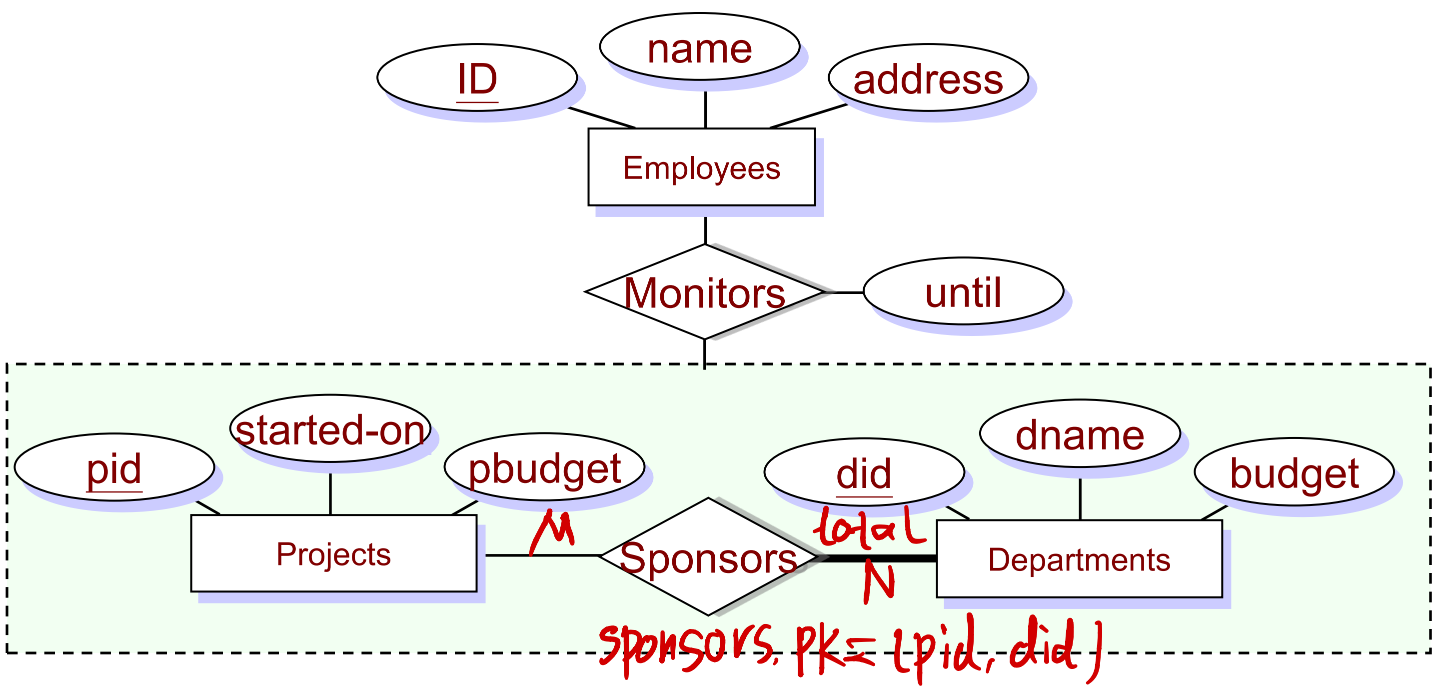
  - This requires the covering constraint to hold. (i.e. Hourly_Emp and Contract_Emp COVER Employee)

Translating Class Hierarchy

- The first method is $\boxed{\text{more general}}$.
- Disadvantage:
  - an extra relation is needed
  - more operation may be necessary when we need to get the employee information (e.g. looking up two relations)
- The second method is not always possible
- Advantage:
  - information of each employee is more easily accessible (usually only one relation look up)
  - however, if an employee can be both hourly and contract based, then information of the employee will be stored twice

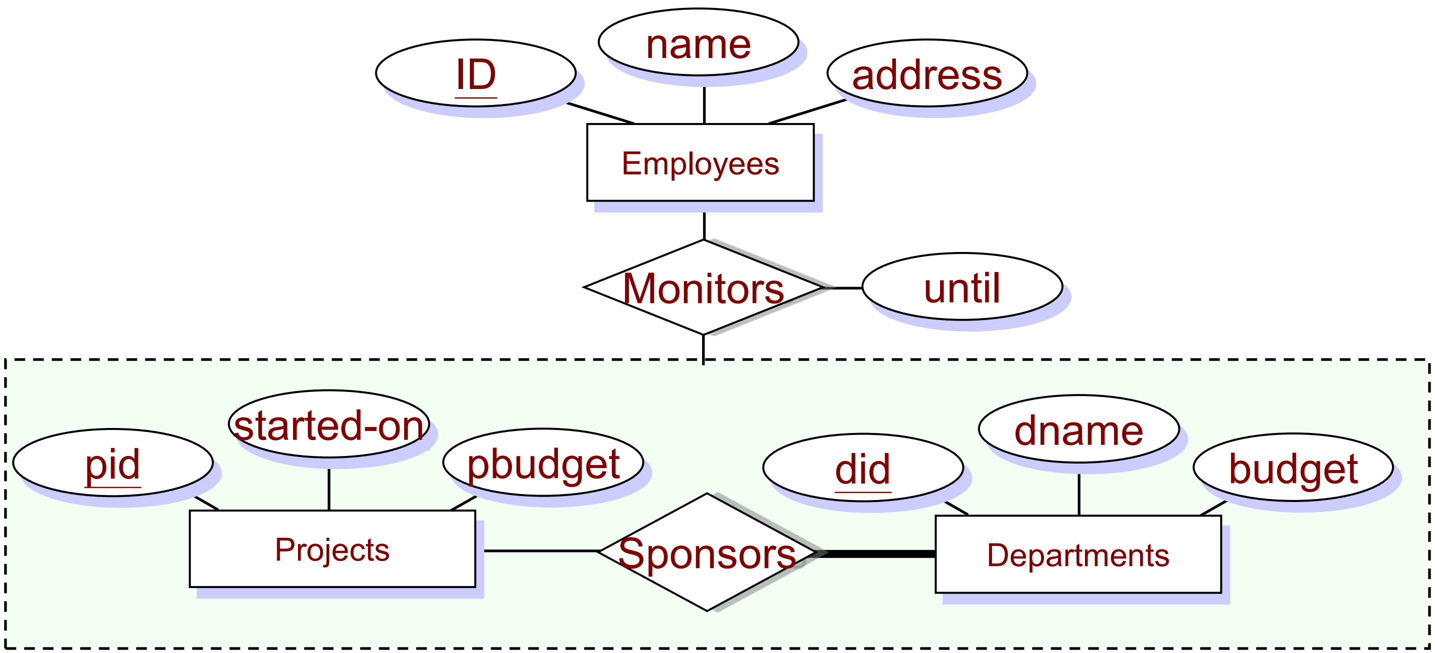
Translating Aggregation

- There is no real distinction between entities and relationships in the relational model
- Therefore the mapping of aggregation in the E-R model to the relational model is quite simple
- For example,



Translating Aggregation

- For example,
  - The key attributes of SPONSORS are: (did, pid)
  - (a sponsorship is determined by department id and project id)



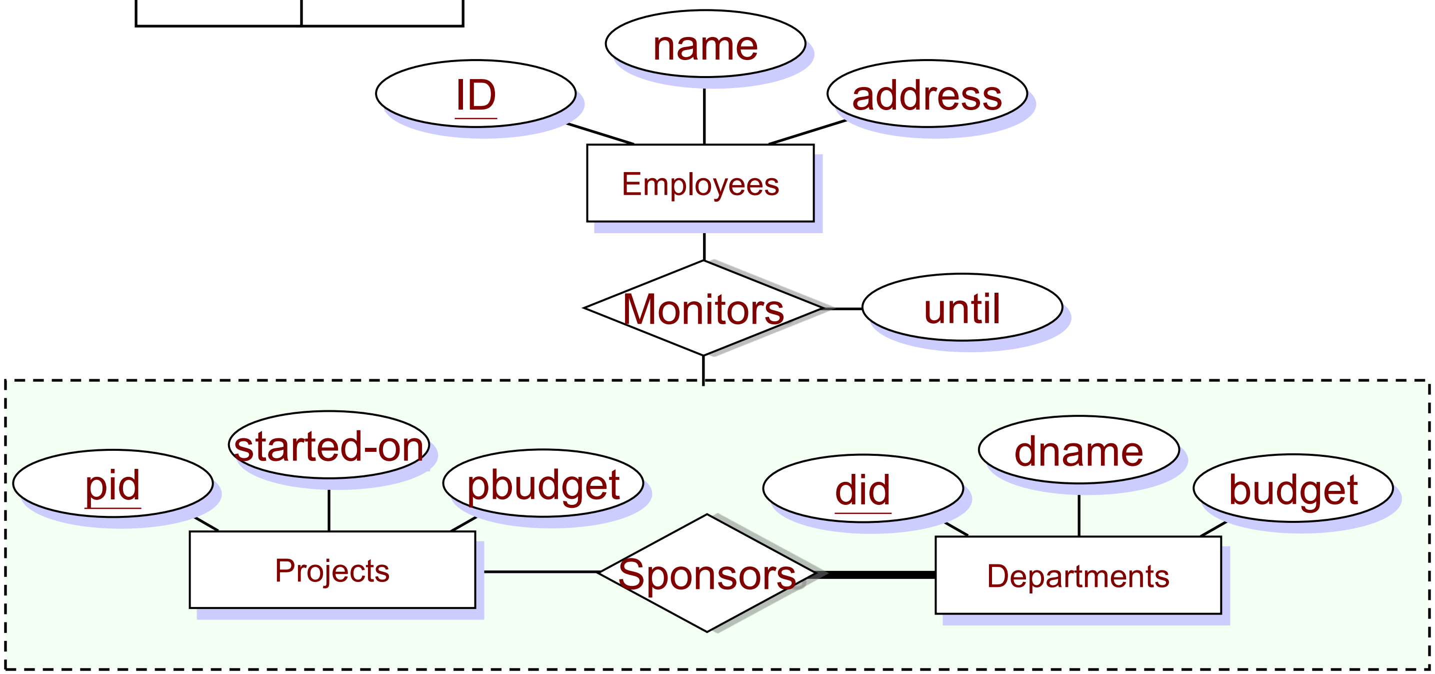
- The key attributes of SPONSORS are: (did, pid)
- For the Monitors relationship we create a relation:

MONITORS

id	did	pid	until

SPONSORS

did	pid



- Sponsors is not contained in Monitors above, if a sponsorship has no monitor, then it will not appear in Monitors

Summary

- What is a relation, relation schema
- How SQL creates tables (relations)
- Integrity Constraints - Primary Keys, Foreign Keys

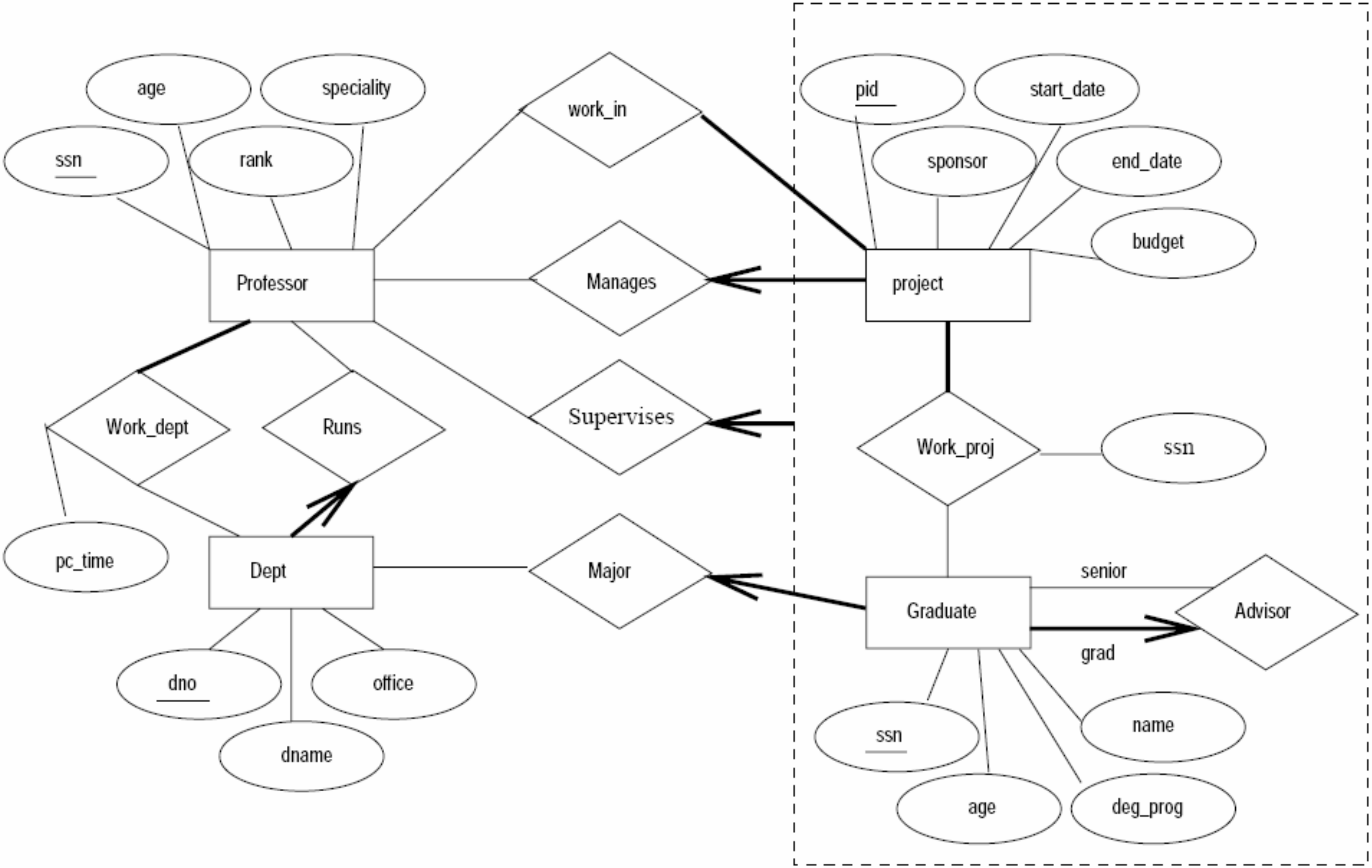
  Referential Integrity
- Translating ER to relational model
  - Translating class hierarchies, aggregations
  - Reduce data redundancy
  - Reduce the number of NULL values
  - Reduce the cost of lookup for the information

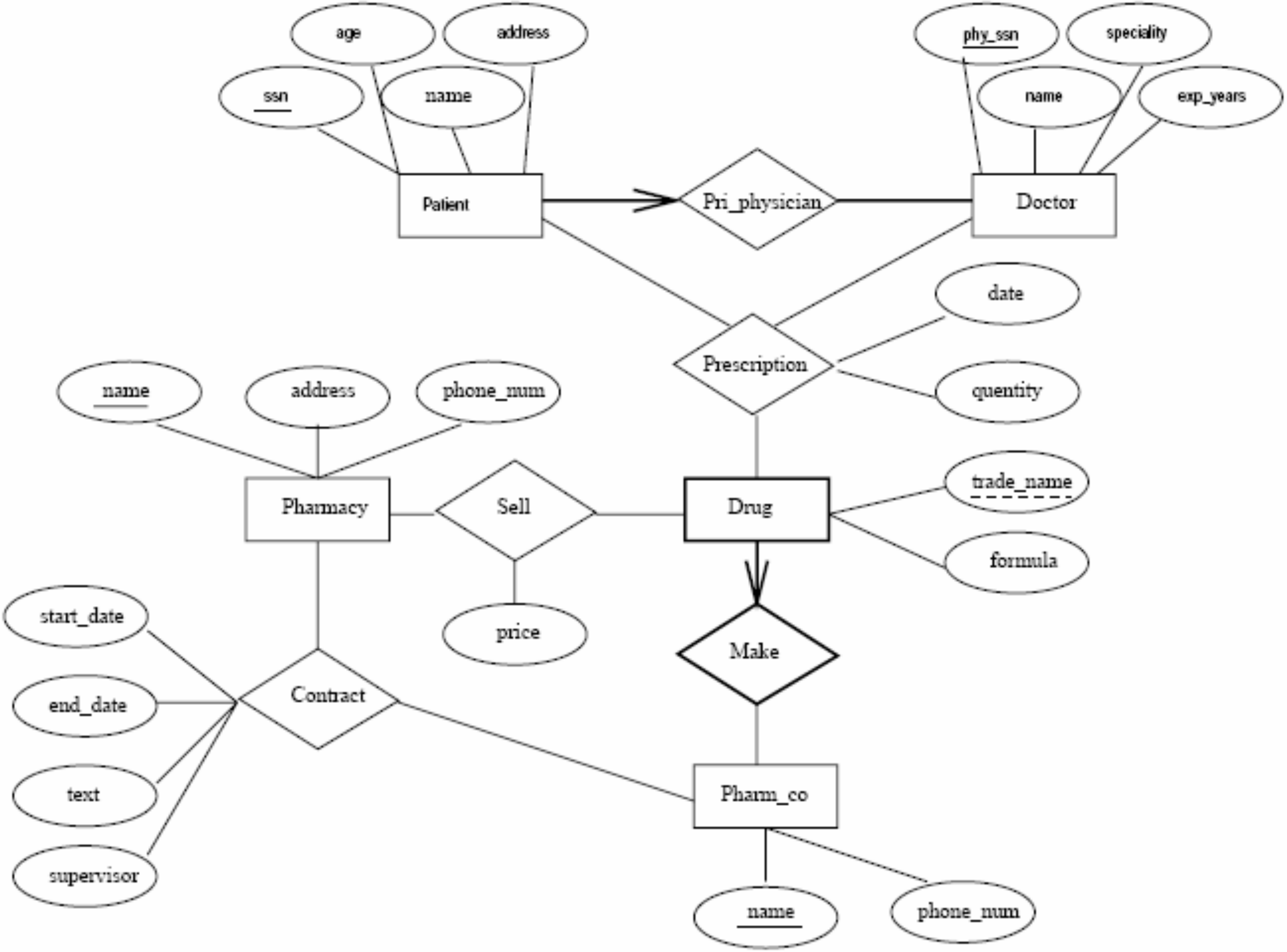
In-Class Exercise

Consider the following relations for a database that keeps track of student enrollment in courses and the books adopted for each course:

- STUDENT(SSN, Name, Major, Bdate)
- COURSE(Course#, Cname, Dept)
- ENROLL(SSN, Course#, Quarter, Grade)
- BOOK_ADOPTION(Course#, Quarter, Book_ISBN)
- TEXT(Book_ISBN, Book_Title, Publisher, Author)

Draw a relational schema diagram specifying the foreign keys for this schema.


age
speciality
ssn
rank
work_in
pid
start_date
sponsor
end_date
budget
Professor
Manages
project
Supervises
Work_dept
Runs
Work_proj
ssn
pc_time
Dept
Major
senior
Graduate
Advisor
grad
dno
office
dname
ssn
name
age
deg_prog


age
address
ssn
name
Patient
Pri_physician
phy_ssn
speciality
name
exp_years
Doctor
date
Prescription
quentity
name
address
phone_num
Pharmacy
Sell
Drug
trade_name
formula
price
Make
start_date
end_date
text
supervisor
Contract
Pharm_co
name
phone_num