

CS-GY 6083 A: Principles of Database Systems

The Relational Model Translating ER models to Relational Schemas

supplementary material:
“Database Management Systems” 3.1-3.5
class notes
GoT.sql

Prof. Julia Stoyanovich
Computer Science and Engineering at Tandon
Center for Data Science
New York University

Game of Thrones

Characters

<u>name</u>	house
Eddard	Stark
Jon Snow	Stark
Tyrion	Lannister
Daenerys	Targaryen

Episodes

<u>season</u>	<u>num</u>	title	viewers
1	1	Winter is Coming	2.2 M
1	2	The Kingsroad	2.2 M
2	1	The North Remembers	3.9 M
2	2	The Night Lands	3.8 M

Appearances

<u>name</u>	<u>season</u>	<u>num</u>
Eddard	1	1
Eddard	1	2
Jon Snow	1	2
Jon Snow	2	1
Jon Snow	2	2
Tyrion	1	1
Tyrion	1	2
Tyrion	2	2
Daenerys	1	2
Daenerys	2	1

```
select E.title, C.name, C.house
from Characters C, Episodes E, Appearances A
where C.house = 'Targaryen'
and A.name = C.name
and E.season = A.season
and E.num = A.num
```

What does this query compute?
How does it compute its results?

Relational model basics

- Introduced by Edgar F. Codd in 1970 (Turing award)
- At the heart of relational database systems
 - the basic abstraction is a *relation* (a table)
 - *tuples* are stored in rows
 - *attributes* of tuples are stored in columns
 - conceptually, a relation is a *set* of tuples
- Why this model?
 - Simple yet powerful
 - Great for processing very large data sets in bulk

Relational model basics

Episodes (season: int, num: int, *title*: string, *viewers*: long)

<u>season</u>	<u>num</u>	title	viewers
1	1	Winter is Coming	2.2 M
1	2	The Kingsroad	2.2 M
2	1	The North Remembers	3.9 M
2	2	The Night Lands	3.8 M

- **Relation**: a set or tuples - order doesn't matter, all tuples are distinct
- **Attribute**: a column in a relation (e.g., *season*)
- **Domain**: data type of an attribute (e.g., *season*: int)
- **Tuple**: a row in a relation, e.g., <1, 2, The Kingsroad, 2.2 M>

Schema vs. instances

- **Relation schema** is a description of a relation in terms of relation name, attribute names, attribute datatypes, constraints (e.g., keys)
- A schema describes **all valid instances** of a relation

schema Episodes (season: integer, num: integer, *title*: string, *viewers*: integer)

instance 1

<u>season</u>	<u>num</u>	title	viewers
1	1	Winter is Coming	2.2 M
1	2	The Kingsroad	2.2 M
2	1	The North Remembers	3.9 M

instance 2

<u>season</u>	<u>num</u>	title	viewers
1	20	Blah, Blah and Blah	0
4	7	Yet Another Title	10 B

instance 3

<u>season</u>	<u>num</u>	title	viewers
---------------	------------	-------	---------

Equivalent representations of a relation

- Relation schemas are **sets** of attributes
 - The order in which attributes appear does not matter
 - Each attribute appears (at most) once
- Relation instances are **sets** of tuples
 - The order in which tuples appear does not matter
 - Each valid (possible) tuple appears in an instance at most once

These are all equivalent representations!

Episodes

<u>season</u>	<u>num</u>	title	viewers
1	1	Winter ...	2.2 M
1	2	The King....	2.2 M
2	1	The North...	3.9 M

Episodes

<u>season</u>	<u>num</u>	viewers	title
1	1	2.2 M	Winter...
1	2	2.2M	The King....
2	1	3.9M	The North...

Episodes

<u>season</u>	<u>num</u>	viewers	title
1	2	2.2M	The King....
2	1	3.9M	The North...
1	1	2.2 M	Winter...

Integrity constraints

- Ensure that data adheres to the rules of the application
 - Specified when schema is defined
 - Checked and enforced by the DBMS when relations are modified (tuples added / removed / updated)
 - **Must hold on every valid instance of the database**
1. *Domain constraints* - specify valid data types for each attribute, e.g., Students (*sid*: integer, *name*: string, *gpa*: decimal)
 2. *Key constraints* - define a unique identifier for each tuple
 3. *Referential integrity constraints* - specify links between tuples

Key constraints

Definition: A set of attributes is a *candidate key* for a relation if:

1. No two distinct tuples can have the same values for all key attributes (*candidate key identifies a tuple*)
2. This is not true for any subset of the key attributes (*candidate key is minimal*)

- If #2 is false (the set of attributes is not minimal) we have a *superkey*
- If there are 2 or more candidate keys
 - one of them is chosen to be the *primary key*
 - other candidate keys are marked as such using the **UNIQUE** keyword

Keys: an example

Students (*sid*: integer, *login*: string, *name*: string, *dob*: date)

1. What could be the candidate keys?
2. For each candidate key, what are all the super keys?

Option 1 candidate key = {*sid*}

super keys = { {*sid* *login*}, {*sid* *name*}, {*sid* *dob*}, {*sid* *login* *name*},
{*sid* *login* *dob*}, {*sid* *name* *dob*}, {*sid* *login* *name* *dob*} }

Option 2 candidate key = {*login*}

super keys = { {*login* *sid*}, {*login* *name*}, {*login* *dob*}, {*login* *sid* *name*},
{*login* *sid* *dob*}, {*login* *name* *dob*}, {*login* *sid* *name* *dob*} }

Option 3 candidate key = {*name* *dob*}

super keys = { {*name* *dob* *sid*}, {*name* *dob* *login*}, {*name* *dob* *sid* *login*} }

Keys: more examples

Movies (*title*: string, *genre*: string, *year*: integer, *length*: integer)

1. What could be the candidate keys?
2. For each candidate key, what are all the super keys?

Movie_Stars (*name*: string, *dob*: date, *active_start*: date, *active_end*: date)

1. What could be the candidate keys?
2. For each candidate key, what are all the super keys?

Creating and modifying relations

1. Translate domain constraints to vendor-specific data types
2. Implement key constraints

Departments (*did*: integer, *name*: string)

```
create table Departments (  
  did      integer      primary key,  
  name     varchar(128) not null  
);
```

Every relation must have a primary key!

This relation has 1 candidate key, it becomes the primary key.

PostgreSQL datatypes

integer, decimal, numeric, real

char(x) - stores a string of length exactly x

varchar(x) - stores a string of length *at most* x

date - stores date but no time (1 day resolution)

timestamp - stores date and time (up to microsecond)

time - stores time of day but no date (up to microsecond)

This is a very basic list, for complete information see:

<https://www.postgresql.org/docs/12/datatype.html>

Specifying required attributes

To specify that each department must have a name, we use the construct **not null**

```
create table Departments (  
  did integer primary key,  
  name varchar(128) not null  
);
```

Departments

did	name
1	CS
2	Italian

Departments

did	name
1	CS
2	<i>null</i>

Departments

did	name
1	CS
2	Italian
3	CS
1	English

Which of these instances are illegal and why?

Creating relations

1. Translate domain constraints to vendor-specific data types
2. Implement key constraints

Students (*sid*: integer, *login*: string, *name*: string, *dob*: date, *gpa*: decimal)

```
create table Students (  
  sid    integer      primary key,  
  login  varchar(128) unique,  
  name   varchar(128),  
  dob    date,  
  gpa    decimal,  
  unique (name, dob)  
);
```

Every relation must have exactly one primary key!

This relation has 3 candidate keys

1. *sid* becomes the primary key
2. *login* is designated unique
3. the combination of *name* and *dob* is designated unique

Primary keys may be composite

1. Translate domain constraints to vendor-specific data types
2. Implement key constraints

Students (*sid*: integer, *login*: string, *name*: string, *dob*: date, *gpa*: decimal)

```
create table Students (  
    sid    integer        unique,  
    login  varchar(128)   unique,  
    name   varchar(128),  
    dob    date,  
    gpa    decimal,  
    primary key (name, dob)  
);
```

Every relation must have exactly one primary key!

This relation has 3 candidate keys

1. *sid* is designated unique
2. *login* is designated unique
3. the combination of *name* and *dob* becomes the primary key

How do we pick a primary key?

- If there is only one candidate key - that's our primary key, no choice
- If multiple candidate keys are available, pick one that
 - does not contain any attributes that may be null
 - is simple (as opposed to composite), or made up of fewer columns if composite
 - is compact (integer better than varchar(128))
 - is made up of attributes whose values do not change, or do not change often
- It is sometimes practical to add a simple, compact column to the relation schema, to use as a primary key, e.g., add a “synthetic” id column

Primary key vs. UNIQUE

```
create table Employees (  
    id      integer      primary key,  
    login   varchar(128)  unique,  
    name    varchar(128)  not null  
);
```

- Primary key
 - No two tuples can agree on values for (all attributes) in a primary key
 - All attributes that make part of a primary key must have assigned values (i.e., they **may not be null**)
- UNIQUE
 - No two tuples can agree on values for unique attributes (or combinations of attributes designated unique)
 - However, unique attributes **may be null**

Primary key vs. UNIQUE

```
create table Employees (  
    id      integer      primary key,  
    login   varchar(128) unique,  
    name    varchar(128) not null  
);
```

Instance 1

id	login	name
1	jim	Jim Morrison
2	amy	Amy Winehouse
3	<i>null</i>	Ethan Coen
4	raj	Raj Kapoor

Instance 3

id	login	name
1	raj	Raj Kapoor
2	amy	Amy Winehouse
3	<i>null</i>	Ethan Coen
4	raj	Raj Kapoor

Instance 2

id	login	name
1	<i>null</i>	Jim Morrison
2	amy	Amy Winehouse
3	<i>null</i>	Ethan Coen
4	raj	Raj Kapoor

Instance 4

id	login	name
1	jim	Jim Morrison
2	amy	<i>null</i>
3	<i>null</i>	Ethan Coen
4	raj	Raj Kapoor

Which of these instances are illegal and why?

Creating relations with SQL

Consider relation schemas and business rules below.

Write create table statements that encode these relation schemas.

Give an example of a valid relation instance.

Famous_Scientists (*name*: string, *dob*: date, *field*: string, *employer*: string)

No two scientists have the same *name*.

No two scientists have the same combination of *dob* and *field* of study.

Each scientist has a *name*, a *dob* and a *field* of study.

Not all scientists are *employed*.

Creating relations with SQL (solution)

Consider relation schemas and business rules below.

Write create table statements that encode these relation schemas.

Give an example of a valid relation instance.

Famous_Scientists (*name*: string, *dob*: date, *field*: string, *employer*: string)

No two scientists have the same *name*.

No two scientists have the same combination of *dob* and *field* of study.

Each scientist has a *name*, a *dob* and a *field* of study.

Not all scientists are *employed*.

Creating relations with SQL

Consider relation schemas and business rules below.

Write create table statements that encode these relation schemas.

Give an example of a valid relation instance.

Space_Ships (*name*: string, *seq_num*: integer, *country*: string, *captain*: string, *flight*: date)

No two space ships have the same combination of name and sequence number (*seq_num*).

No two space ships have the same combination of *captain* and *flight* date.

Each space ship has a name, a seq_num, and a country.

Not all space ships have been flown or have a captain.

Creating relations with SQL (solution)

Consider relation schemas and business rules below.

Write create table statements that encode these relation schemas.

Give an example of a valid relation instance.

Serial Types

serial and bigserial are a convenient way to specify auto incrementing columns in PostgreSQL

```
create table Employees (  
    id      serial          primary key,  
    name    varchar(128)  
);
```

this is equivalent to specifying the following

```
create sequence Employees_id_seq;
```

```
create table Employees (  
    id      integer primary key default nextval ('Employees_id_seq,')  
    name    varchar(128)  
);
```

```
alter sequence Employees_id_seq owned by Employees.id;
```

Modifying relation schemas with SQL

- Deleting a relation (removes both the schema and the data)

```
drop table Students;
```

- Changing the schema of a relation

```
alter table Students add class char(4);
```

```
alter table Students add class char(4) default 2017;
```

```
alter table Students drop column gpa;
```


Where do business rules come from?

- **Business rules are given:** by the client, by the application designer, by your boss
- A relation schema encodes business rules
- **We can never-ever-ever deduce business rules by looking at the relation instance!**
- We can sometimes know which rules do not hold, but we cannot be sure which rules do hold

Employees

id	login	name
1	jim	Jim Morrison
2	amy	Amy Winehouse
3	amy	Amy Pohler
4	raj	Raj Kapoor

1. Which column **is not** a candidate key?
2. Which column(s) **may be** a candidate key?
3. Give 2 create table statements for which this instance is valid.

Foreign key constraints

- A foreign key defines a (directed) link between tuples in different relations
- Foreign keys enforce *referential integrity*: a requirement that a value in an attribute or attributes of a tuple in one relation must also appear as a value in another relation
- Example
 - *Students*(sid), *Courses*(cid), *Enrollment*(sid, cid, grade)
 - *Enrollment*.sid must refer to an existing student, i.e., must match *Students*.sid for some tuple in *Students*
 - *Enrollment*.cid must refer to an existing course, i.e., must match *Courses*.cid for some tuple in *Courses*

Foreign keys: example

```
create table Students (  
  sid integer primary key,  
  login varchar(128) unique,  
  name varchar(128) not null,  
  dob date not null,  
  gpa decimal,  
  unique (name, dob)  
);
```

```
create table Enrollment (  
  sid integer,  
  cid integer,  
  grade decimal,  
  primary key (sid, cid),  
  foreign key (sid) references Students(sid),  
  foreign key (cid) references Courses(cid)  
);
```

Students

sid	login	name	dob	gpa
123	Jane	Jane S.	1/1/90	4
456	Ann	Ann M.	5/25/92	3.8

Enrollment

sid	cid	grade
123	210	4
456	210	4



Declaring foreign key constraints

- A foreign key in a table must reference (point to) a candidate key in another table.
- That is, the target column(s) are either a primary key or are designated UNIQUE.
- Some relational systems limit this further: a foreign key must point to a primary key. For efficiency reasons, this is usually a better choice.

```
create table Students (  
    sid    integer      primary key,  
    login  varchar(128) unique,  
    name   varchar(128),  
    dob    date,  
    gpa    decimal,  
    unique (name, dob)  
);
```

```
create table Sibling_Pairs (  
    thing_one    integer,  
    thing_two    integer,  
    primary key (thing_one, thing_two),  
    foreign key (thing_one)  
                references Students(sid),  
    foreign key (thing_two)  
                references Students(sid)  
);
```

candidate keys: **sid**, login, (name, dob)

Declaring foreign key constraints

- A foreign key in a table must reference (point to) a candidate key in another table.
- That is, the target column(s) are either a primary key or are designated UNIQUE.
- Some relational systems limit this further: a foreign key must point to a primary key. For efficiency reasons, this is usually a better choice.

```
create table Students (  
    sid    integer      primary key,  
    login  varchar(128) unique,  
    name   varchar(128),  
    dob    date,  
    gpa    decimal,  
    unique (name, dob)  
);
```


```
create table Sibling_Pairs (  
    thing_one  varchar(128),  
    thing_two  varchar(128),  
    primary key (thing_one, thing_two),  
    foreign key (thing_one)  
        references Students(login),  
    foreign key (thing_two)  
        references Students(login)  
);
```


candidate keys: **sid**, login, (name, dob)

Enforcing referential integrity

- Back to *Students* and *Enrollment*; *Enrollment.sid* references *Students.sid*
- What happens if *Enrollment* tuple with a non-existent sid is inserted? (**Reject!**)
- What should be done if *Students* tuple is deleted, with corresponding *Enrollment* tuples?
 1. Disallow deletion - **default behavior**
 2. Delete all corresponding *Enrollment* tuples - **cascading delete**
 3. Set *Enrollment.sid* to a default sid - does not always make sense
 4. Set *Enrollment.sid* to *null* - **allowed if not part of primary key in Enrollment**, but does not always make sense
- Similar question if *Students.sid* is updated in some tuple

```
create table Enrollment (  
    sid      integer,  
    cid      integer,  
    grade    decimal,  
    primary key (sid, cid),  
    foreign key (sid) references Students(sid) on delete cascade,  
    foreign key (cid) references Courses(cid) on delete set default  
);
```

cascading delete 

set default to what? 

Nulls in foreign key columns

```
create table Employees (  
    ssn      char(12)      primary key,  
    name     varchar(128) not null,  
    title    varchar(128)  
);  
insert into Employees (ssn, name, title) values ('111-11-1111', 'Ann', 'CEO');  
insert into Employees (ssn, name, title) values ('222-22-2222', 'Bob', 'clerk');
```

1. an employee has at most one manager, who is also an employee
2. not every employee has a manager

```
create table Managers (  
    emp_ssn    char(12)      primary key,  
    mgr_ssn    char(12),  
    foreign key (emp_ssn) references Employee(ssn),  
    foreign key (mgr_ssn) references Employee(ssn)  
);  
insert into Managers (emp_ssn, mgr_ssn) values ('222-22-2222', '111-11-1111');  
insert into Managers (emp_ssn, mgr_ssn) values ('111-11-1111', null);
```

Foreign key examples

Consider relation schemas and business rules below. Primary keys are given, underlined in the relation schemas.

Write create table statements that encode these relation schemas, with the right foreign key constraints. Give an example of valid relation instances.

Scientists (*name*: string, *field*: string, *lab*: string)

Labs (*name*: string, *location*: string)

Each scientist works at some lab.

If a lab closes, the corresponding scientist tuples are deleted.

Players (*name*: string, *level*: integer, *country*: string, *favorite_game*: string)

Games (*name*: string, *maker*: string)

All players always play their one favorite game.

If a game is discontinued, its players are made to play Angry Birds.

Foreign key examples (solution)

Consider relation schemas and business rules below. Primary keys are given, underlined in the relation schemas.

Write create table statements that encode these relation schemas, with the right foreign key constraints. Give an example of valid relation instances.

Scientists (*name*: string, *field*: string, *lab*: string)

Labs (*name*: string, *location*: string)

Each scientist works at some lab.

If a lab closes, the corresponding scientist tuples are deleted.

Foreign key examples (solution)

Consider relation schemas and business rules below. Primary keys are given, underlined in the relation schemas.

Write create table statements that encode these relation schemas, with the right foreign key constraints. Give an example of valid relation instances.

Players (name: string, level: integer, country: string, favorite_game: string)

Games (name: string, maker: string)

All players always play their one favorite game.

If a game is discontinued, its players are made to play Angry Birds.

Constraints continued

- So far, we talked about enforcing:
 - **domain constraints**, by specifying data types for attributes
 - **not null constraints**, by specifying not null for attributes
 - **key constraints**, by defining primary keys and designating attributes / combinations of attributes as UNIQUE
 - **referential integrity constraints**, by defining foreign keys
- Another useful type of a constraint is a **CHECK constraint**

Specifying CHECK constraints

- We already saw an example: **not null constraints**
- Another example

```
create table People (  
    ssn      char(11) primary key,  
    name     varchar(128),  
    age      integer not null,  
    gender   char(1),  
    country  varchar(64),  
    check (gender in ('M', 'F')),  
    check (age > 0),  
    check (country in (select name from Countries))  
);
```

not supported in
PostgreSQL 12



ERROR: cannot use subquery in check constraint

Naming constraints

```
create table People (  
    ssn      char(11) primary key,  
    name     varchar(128),  
    age      integer not null,  
    gender   char(1),  
    country  varchar(64),  
    constraint Gender_Constraint  
                check (gender in ('M', 'F')),  
    constraint Age_Constraint  
                check (age > 0),  
    constraint Country_Constraint  
                check (country in (select name from Countries))  
);
```

- Q: Why do we name constraints?
- A: For readability.
- A: Also because we can refer to them by name, so as to drop them.

Modifying constraints

```
create table People (  
    ssn      char(11) primary key,  
    name     varchar(128),  
    age      integer not null,  
    gender   char(1),  
    country  varchar(64),  
    constraint Gender_Constraint  
                check (gender in ('M', 'F')),  
    constraint Age_Constraint  
                check (age > 0),  
    constraint Country_Constraint  
                check (country in (select name from Countries))  
);
```

```
alter table People drop constraint Country_Constraint;
```

```
alter table People add constraint SSN_Constraint  
                check (ssn like '%-%-%')
```

Translating from ER to relational

- Recall Entity Relationship modeling from Lecture 1
- The ER model is good for representing high-level database design
- As the next step, we must translate the ER diagram into a collection of database tables with associated integrity constraints
- As we will see, that not all ER constructs can be implemented by the relational model

ER refresher

Draw ER diagrams representing entity sets and relationship sets described below.

Movie stars play in movies. A movie star may only play **one part** in a movie.

Movie stars play in movies. A movie star may play **multiple parts** in a movie, and **all must be recorded**.

Movie stars play in movies. A movie star may play **multiple parts** in a movie, but only the most significant **one must be recorded**.

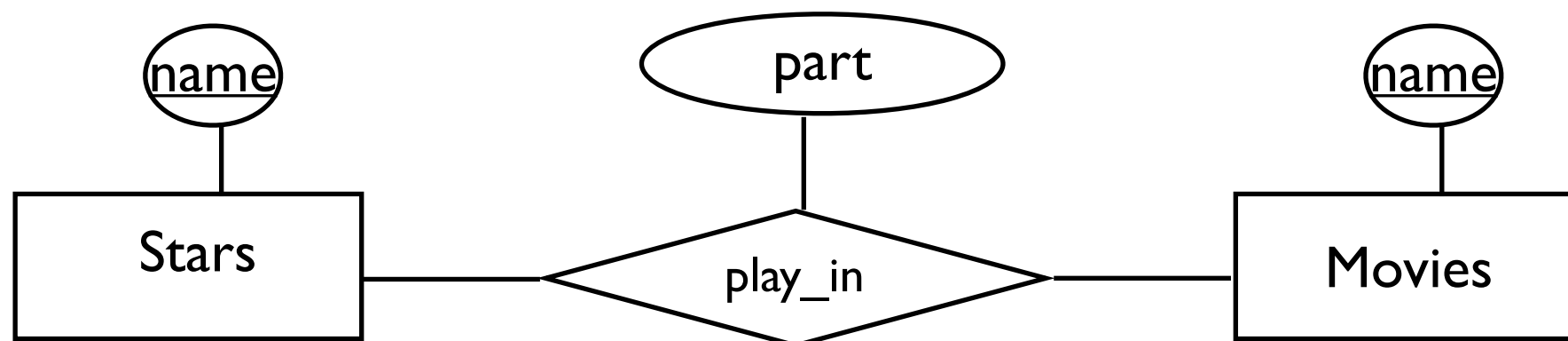
ER refresher (solution)

Draw ER diagrams representing entity sets and relationship sets described below.

Movie stars play in movies. A movie star may only play **one part** in a movie.

Movie stars play in movies. A movie star may play **multiple parts** in a movie, but only the most significant **one must be recorded**.

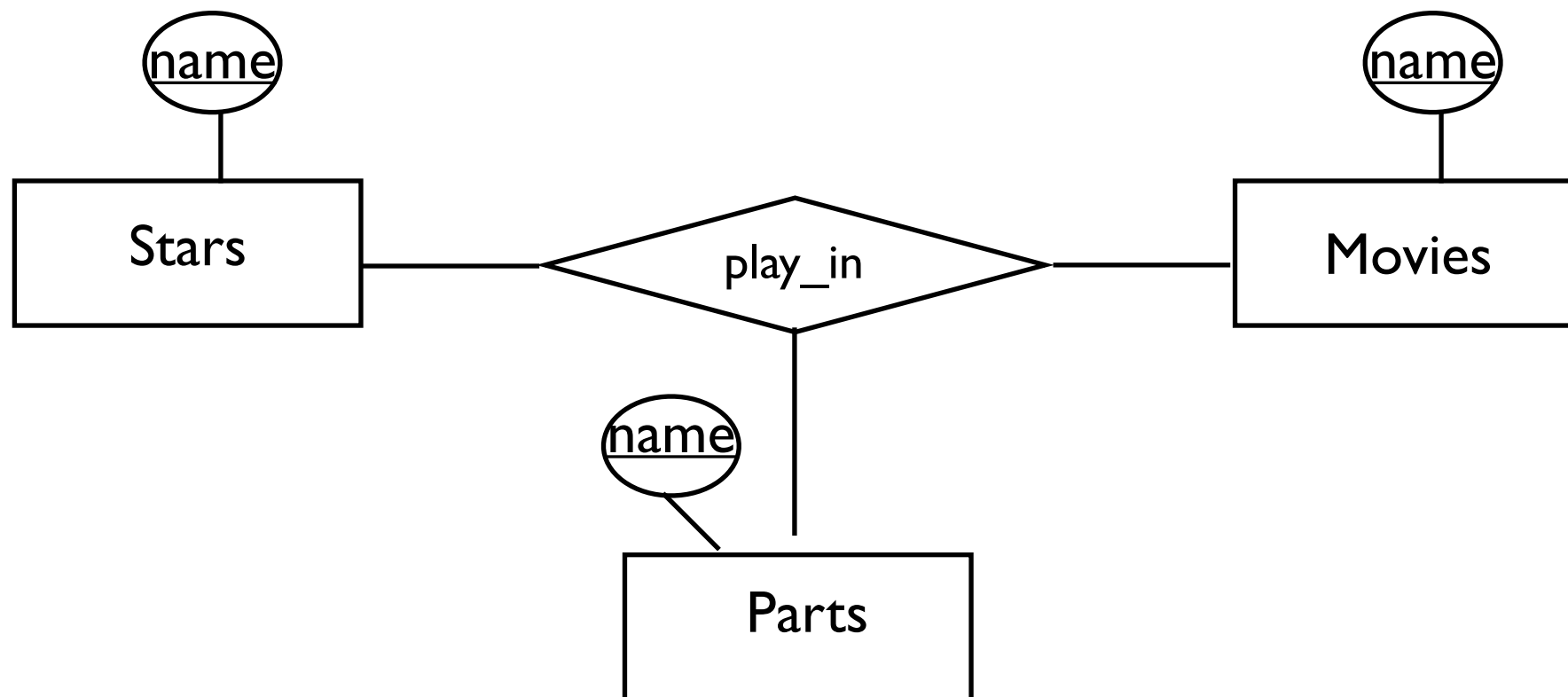
same answer for both!



ER refresher (solution)

Draw ER diagrams representing entity sets and relationship sets described below.

Movie stars play in movies. A movie star may play **multiple parts** in a movie, and **all must be recorded**.



ER refresher

Draw ER diagrams representing entity sets and relationship sets described below. Clearly mark all key and participation constraints.

A dish is made of ingredients. There are between 1 and 10 ingredients per dish. An ingredient may be used in any number of dishes.

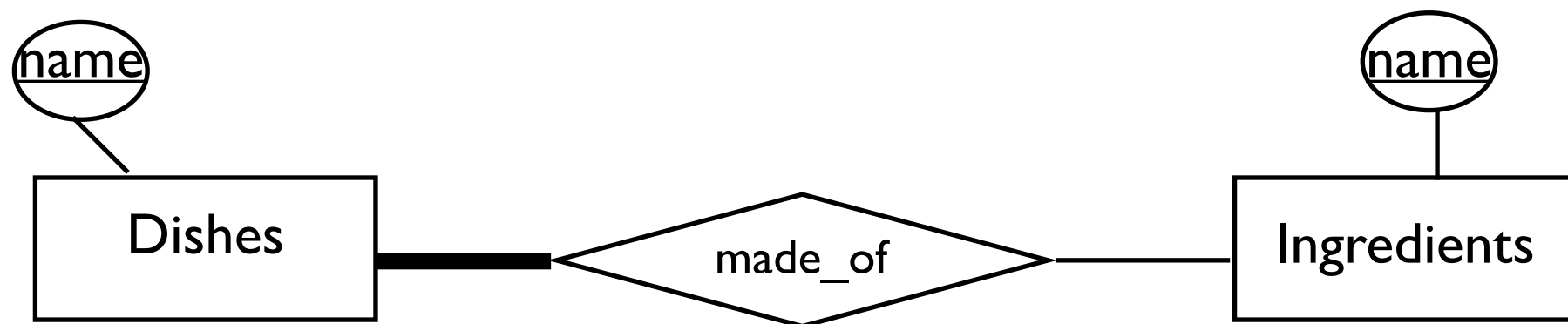
A taxi cab has exactly one driver. A driver is assigned to at most one taxi cab.

A taxi cab has at most one driver. A driver is assigned to at least one taxi cab.

ER refresher (solution)

Draw ER diagrams representing entity sets and relationship sets described below. Clearly mark all key and participation constraints.

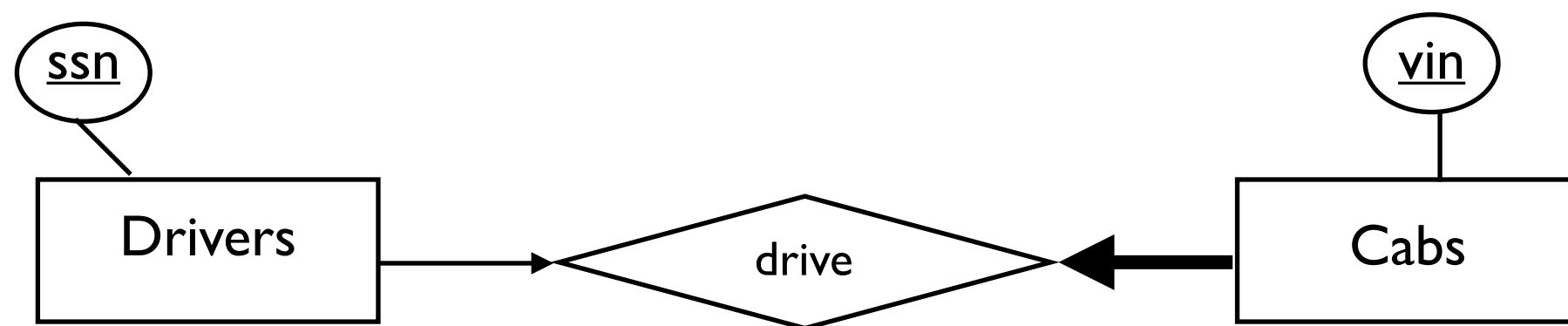
A dish is made of ingredients. There are between 1 and 10 ingredients per dish. An ingredient may be used in any number of dishes.



ER refresher (solution)

Draw ER diagrams representing entity sets and relationship sets described below. Clearly mark all key and participation constraints.

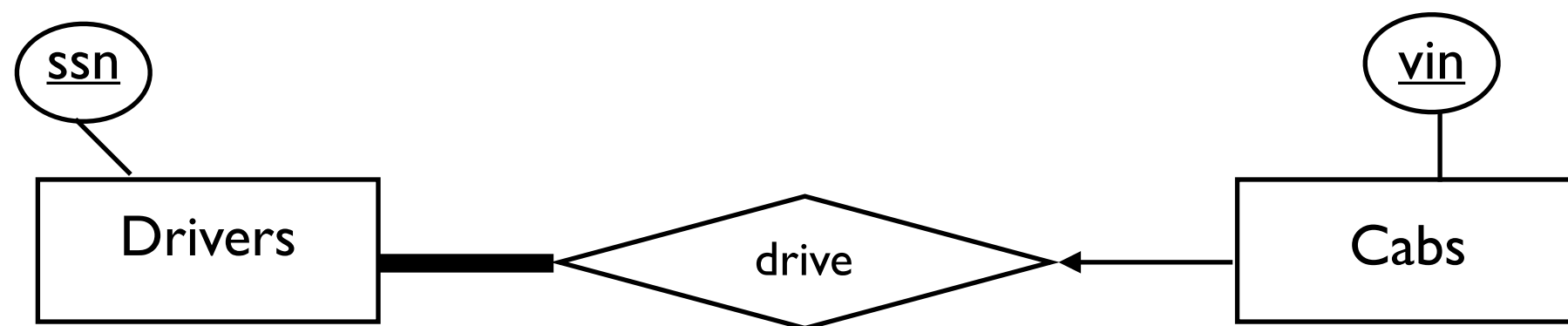
A taxi cab has exactly one driver. A driver is assigned to at most one taxi cab.



ER refresher (solution)

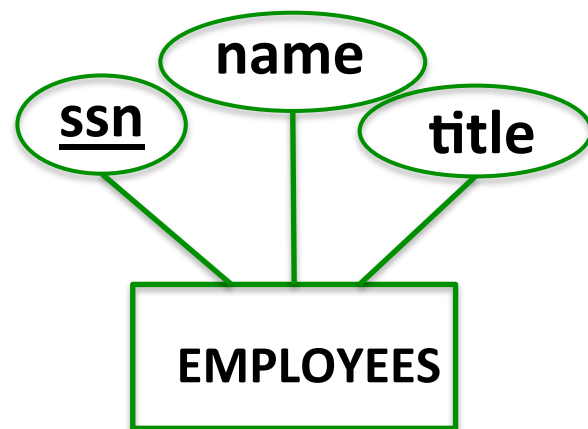
Draw ER diagrams representing entity sets and relationship sets described below. Clearly mark all key and participation constraints.

A taxi cab has at most one driver. A driver is assigned to at least one taxi cab.



Entity sets

An entity set is mapped to a table



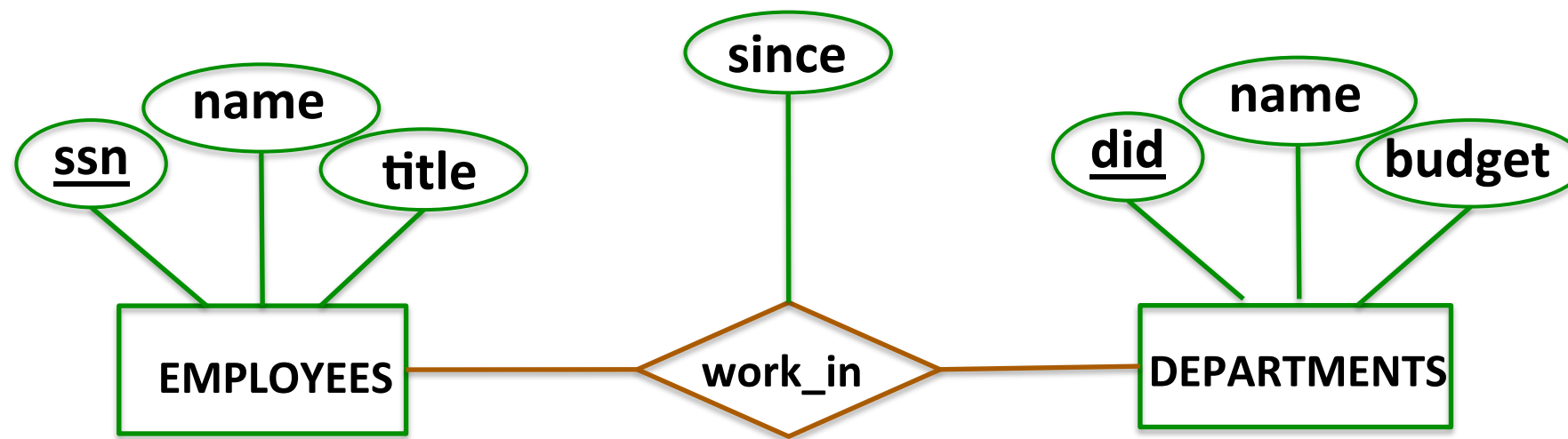
```
create table Employees (  
    ssn    char(11)    primary key,  
    name   varchar(128) not null,  
    title  varchar(128)  
);
```

Employees

ssn	name	title
123-12-3333	Joe	manager
333-56-1235	Ann	CTO
737-11-2345	Jane	clerk

Relationship sets without constraints

A relationship set without key / participation constraints is mapped to a table



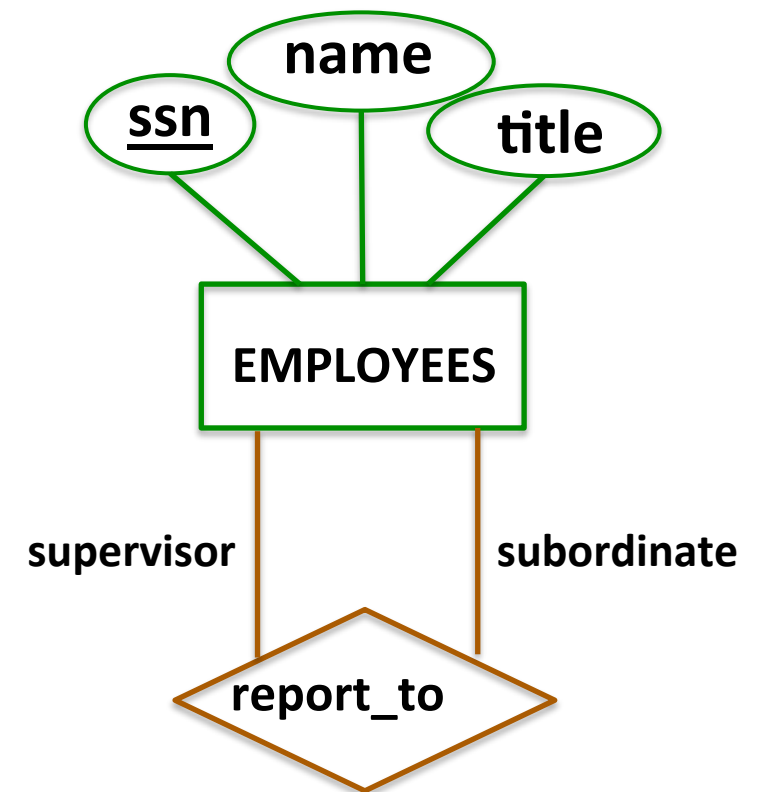
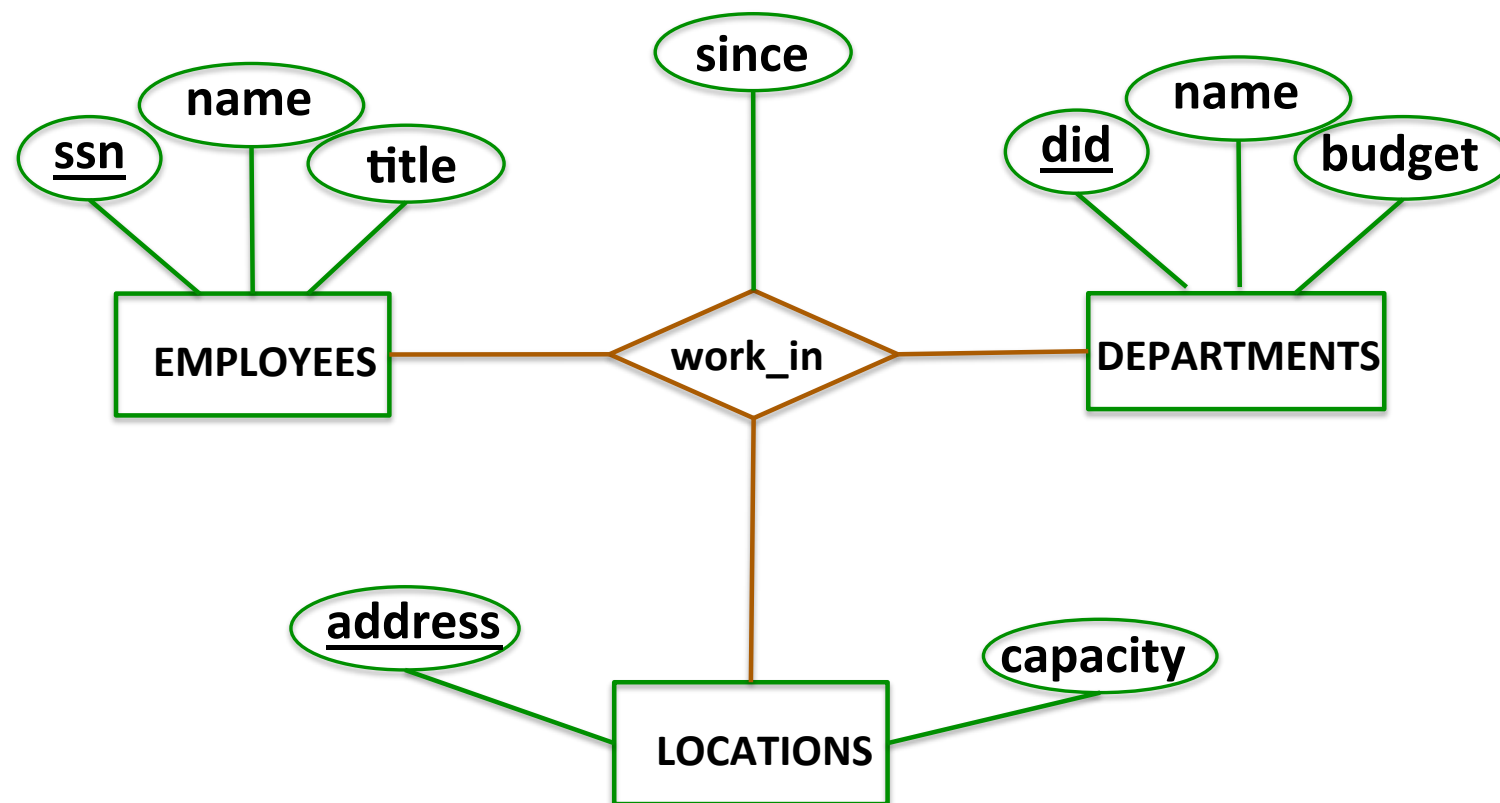
```
create table Employees (  
    ssn      char(11)      primary key,  
    name     varchar(128) not null,  
    title    varchar(128)  
);
```

```
create table Departments (  
    did      integer      primary key,  
    name     varchar(128) not null,  
    budget   integer  
);
```

```
create table Work_In (  
    ssn      char(11),  
    did      integer,  
    since    date,  
    primary key (ssn, did),  
    foreign key (ssn) references Employee(ssn),  
    foreign key (did) references Department(did)  
);
```

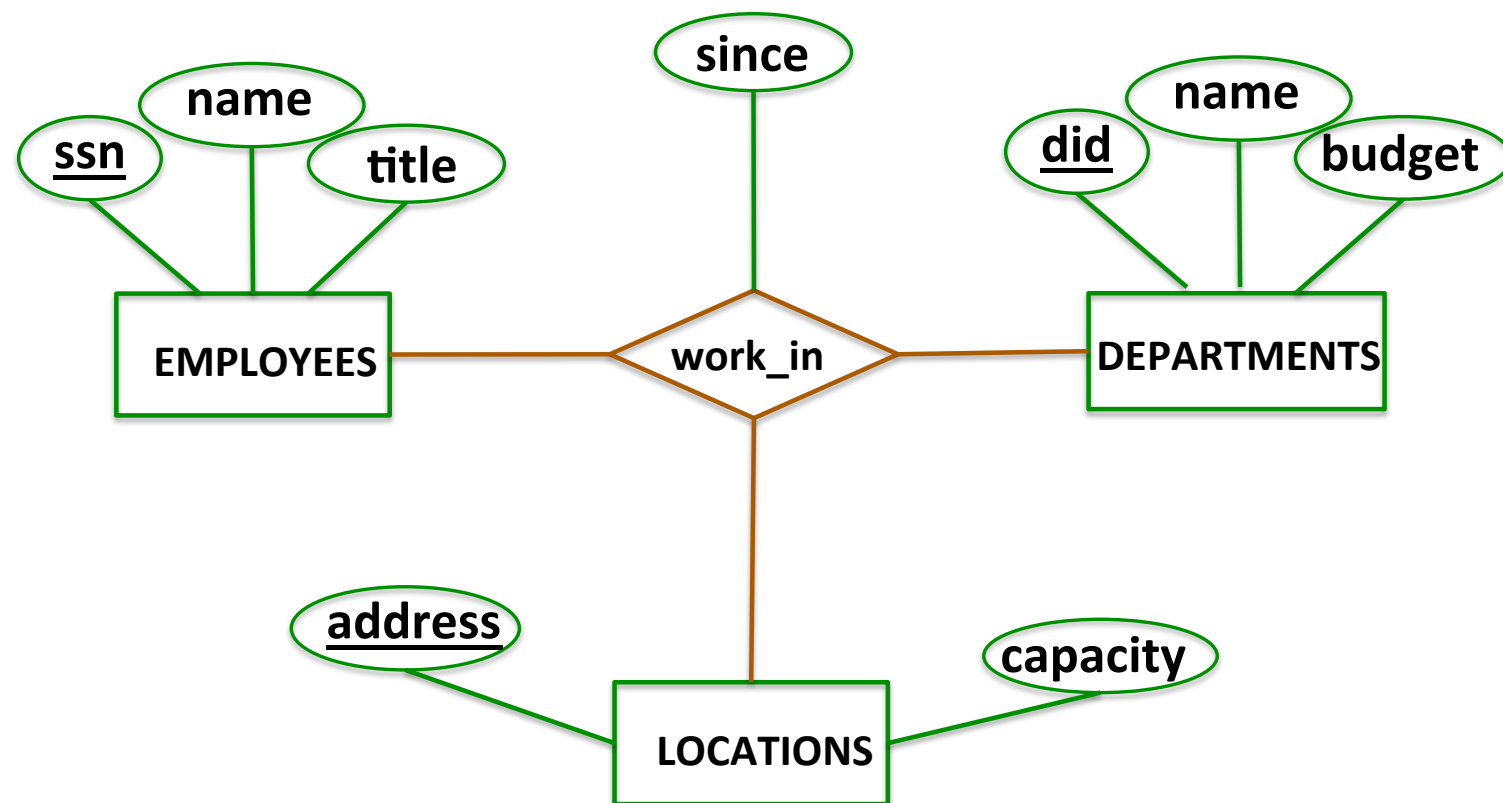

Exercise

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.



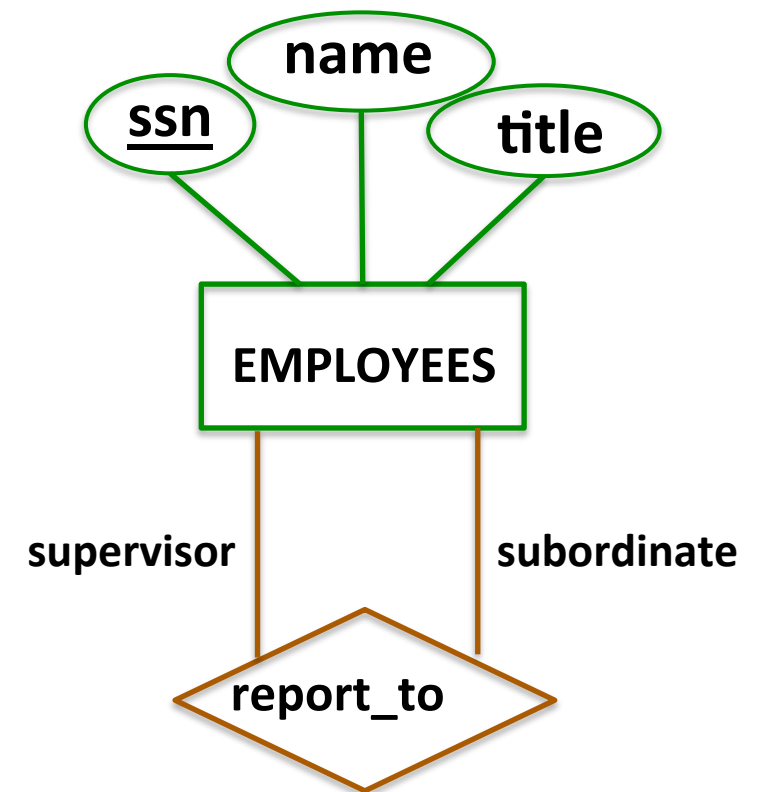
Exercise (solution)

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.

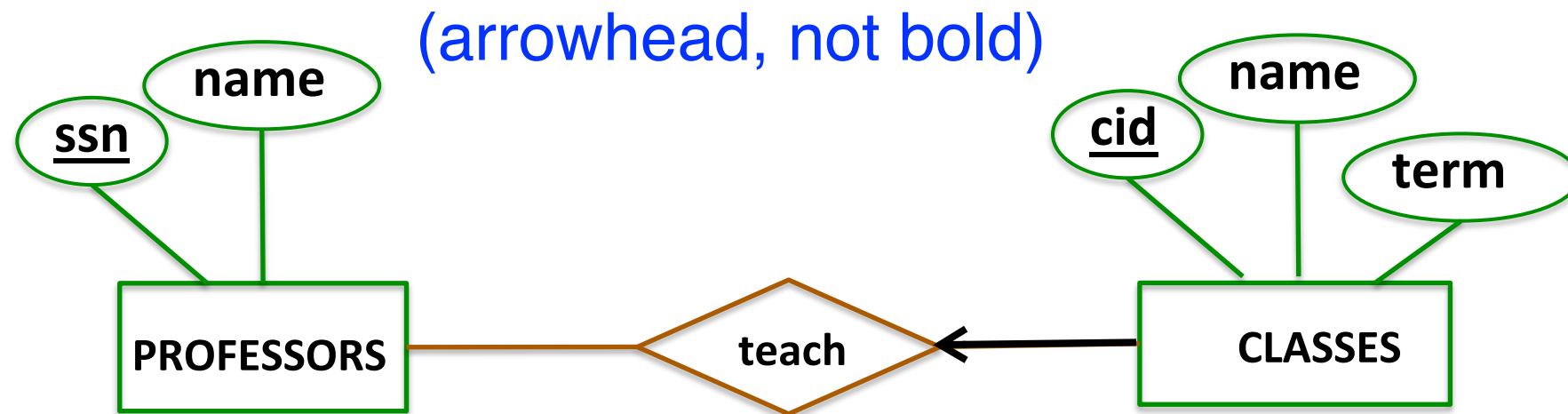


Exercise (solution)

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.



Relationship sets with key constraints (I)



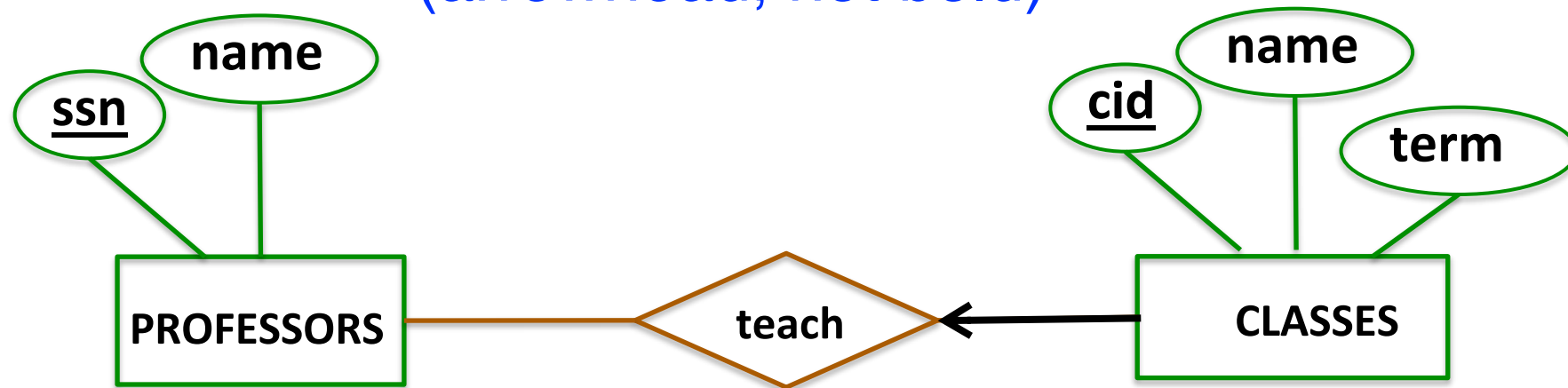
A relationship set with **1 or several key constraints** is modeled using a relation. Keys of entity sets linked via key constraints (arrows) are candidate keys.

```
create table Professors (  
    ssn    char(11) primary key,  
    name   varchar(128)  
);  
  
create table Classes (  
    cid    integer primary key,  
    name   varchar(128),  
    term   varchar(16)  
);
```

```
create table Teach (  
    cid    integer primary key,  
    ssn    char(11),  
    foreign key (ssn) references Professors(ssn),  
    foreign key (cid) references Classes(cid)  
);
```

Relationship sets with key constraints (II)

(arrowhead, not bold)



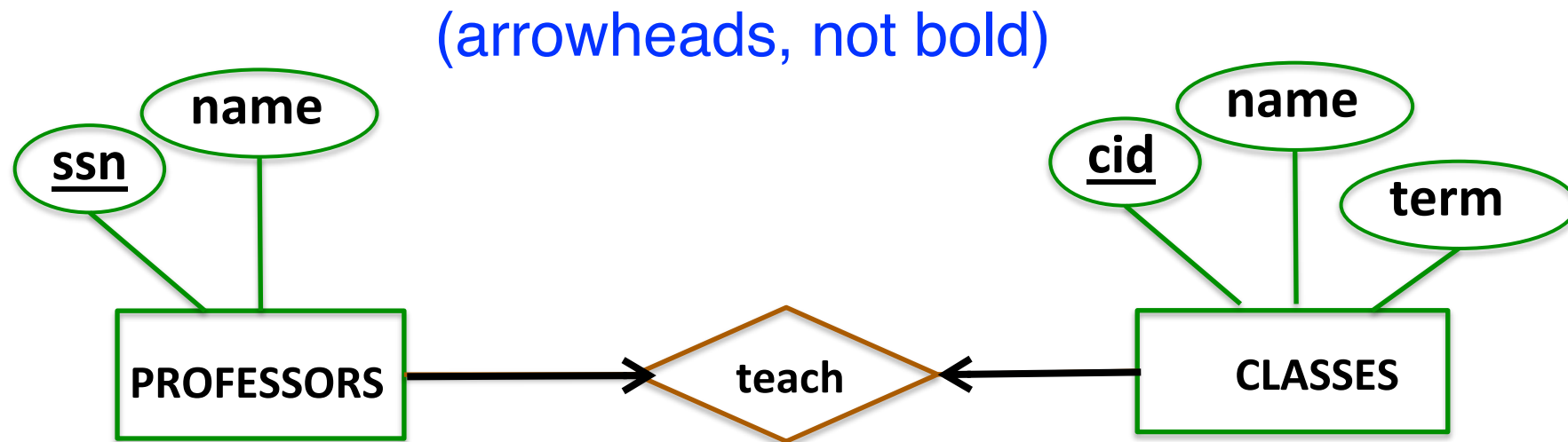
Use one relation to represent the entity set linked by an arrow and the relationship set to which the arrow points. **This is arguably a better option than on the previous slide!**

```
create table Professors (
  ssn    char(11) primary key,
  name   varchar(128)
);
```

```
create table Classes_Teach (
  cid    integer primary key,
  name    varchar(128),
  term    varchar(16),
  ssn     char(11),
  foreign key (ssn) references Professors(ssn)
);
```

does this model classes that are not taught by any professor?

Relationship sets with key constraints (III)



A relationship set with **1 or several key constraints** is modeled using a relation. Keys of entity sets linked via key constraints (arrows) are candidate keys. We pick one as primary key, designate others as unique.

```
create table Professors (  
    ssn    char(11) primary key,  
    name   varchar(128)  
);
```

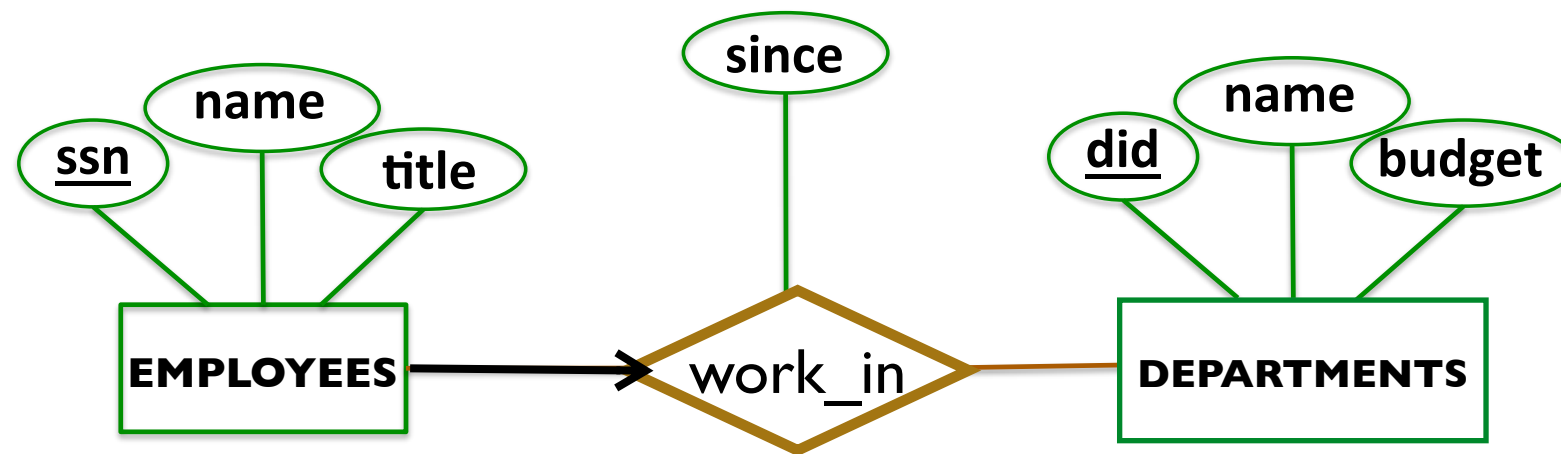
why not make (cid, ssn) primary key?

```
create table Classes_Teach (  
    cid     integer primary key,  
    name    varchar(128),  
    term    varchar(16),  
    ssn     char(11) unique,  
    foreign key (ssn) references Professors(ssn)  
);
```

what's another valid translation?

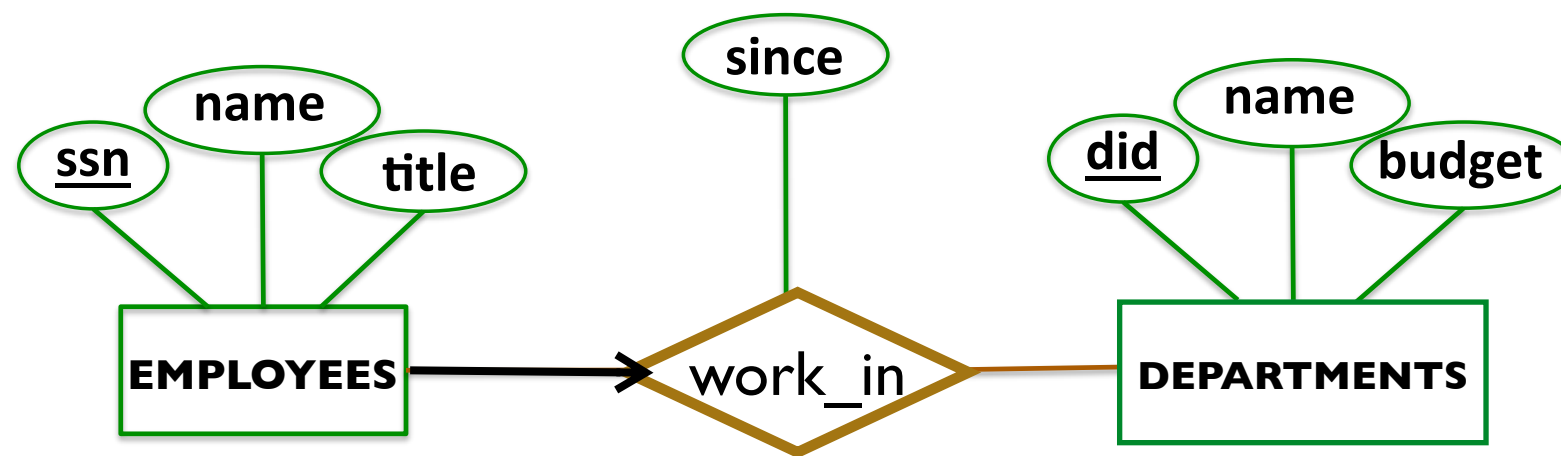
Exercise

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.



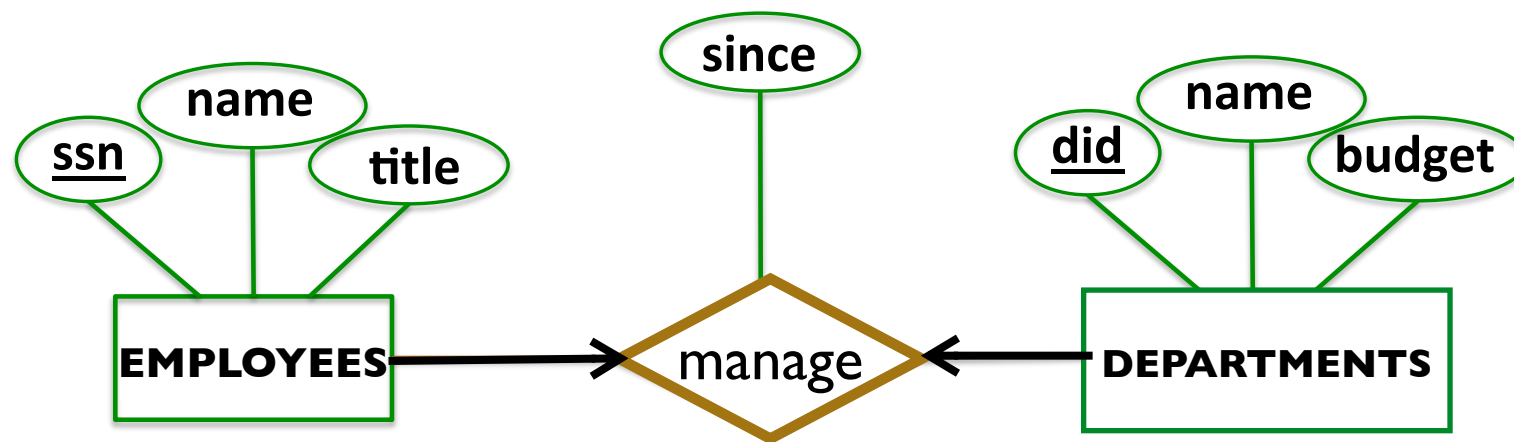
Exercise (solution)

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.



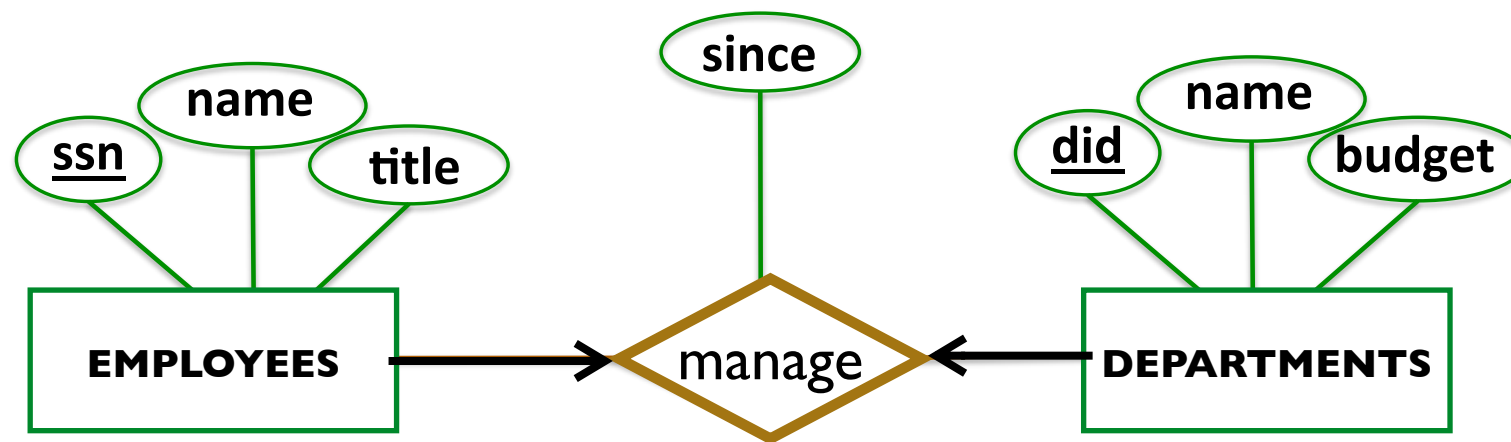
Exercise

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.

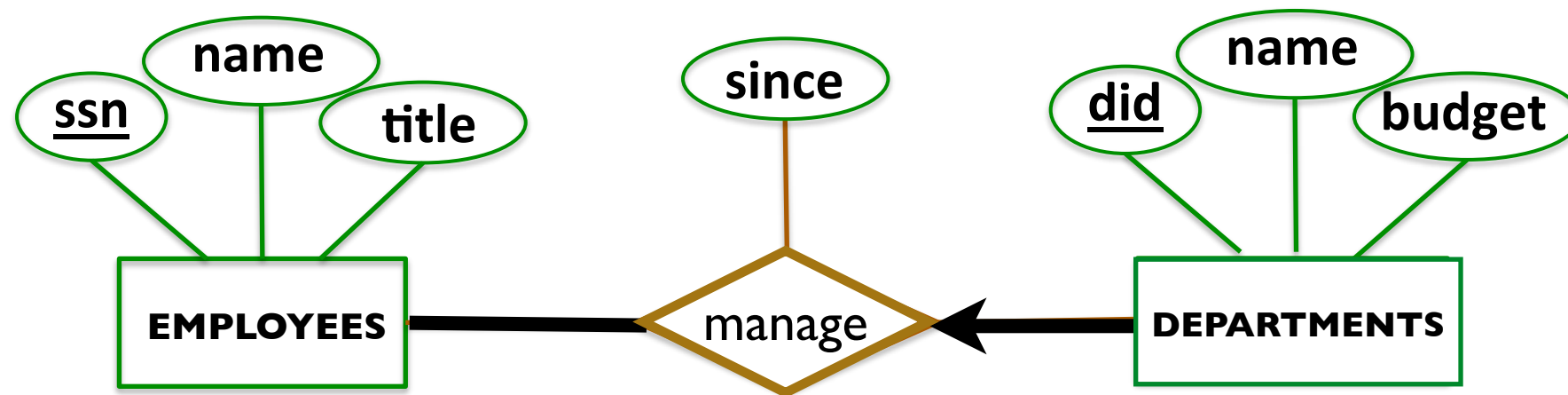


Exercise (solution)

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.



Relationship sets with participation constraints



```
create table Employees (
  ssn char(11) primary key,
  name varchar(128),
  title varchar(64)
);
```

```
create table Departments_Manage (
  did integer primary key,
  name varchar(128),
  budget integer,
  ssn char(11) not null,
  since date,
  foreign key (ssn) references Employees(ssn)
);
```

participation of Department:
“every department must have a manager”

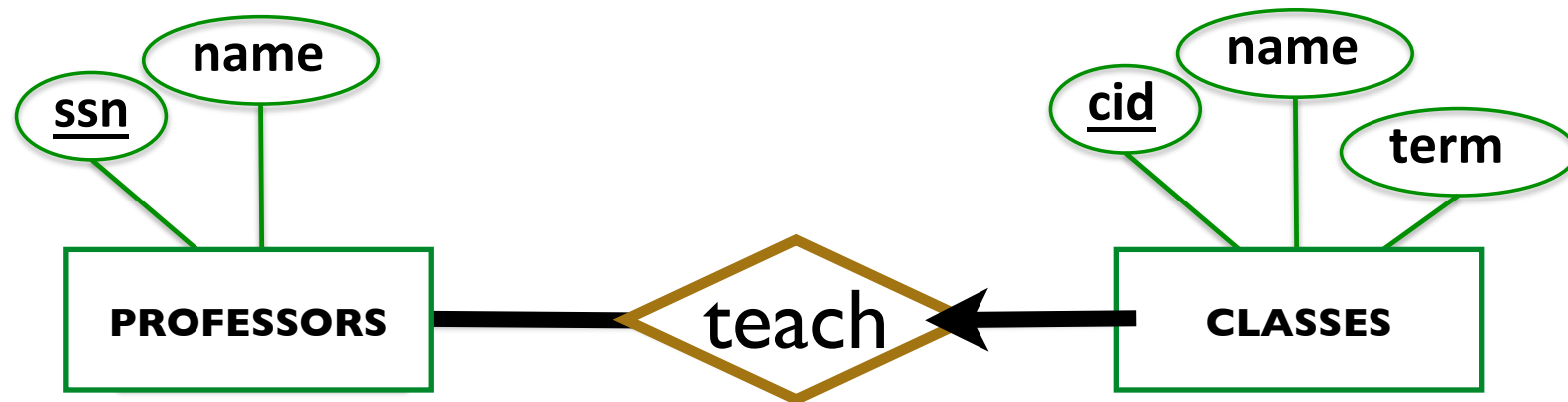
cannot capture participation of Employee without
resorting to table constraints / assertions

Relationship sets with participation constraints

- Without assertions, we can only capture key / participation constraints from **one** entity set. This is done by representing the entity set together with the relationship set as one table.
- A special case we can represent is if all entity sets in the relationship set have both key and participation constraints. The best translation is to map everything into a single table.

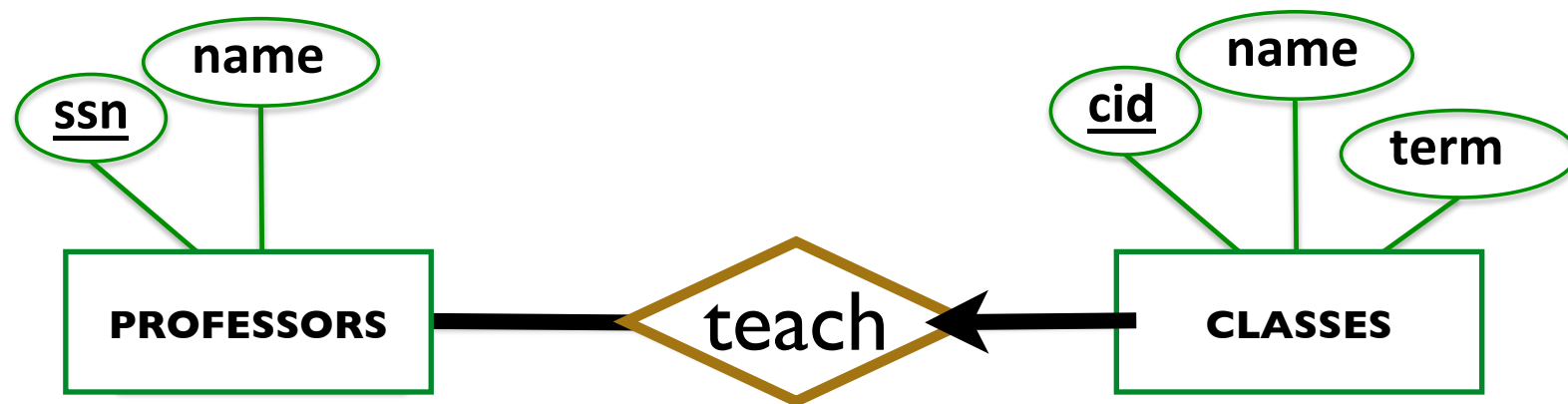
Exercise

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.



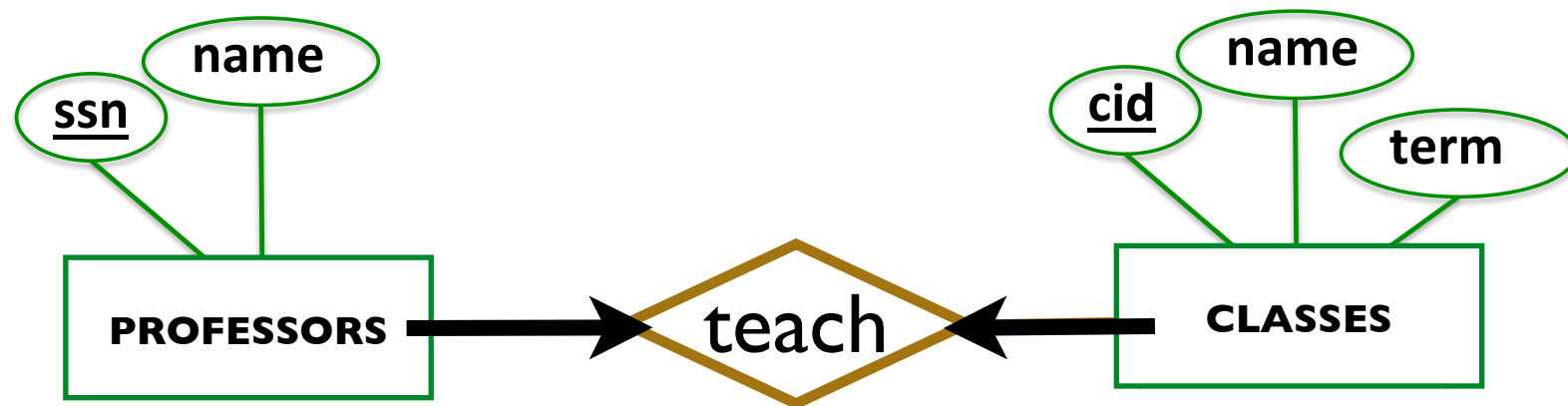
Exercise (solution)

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.



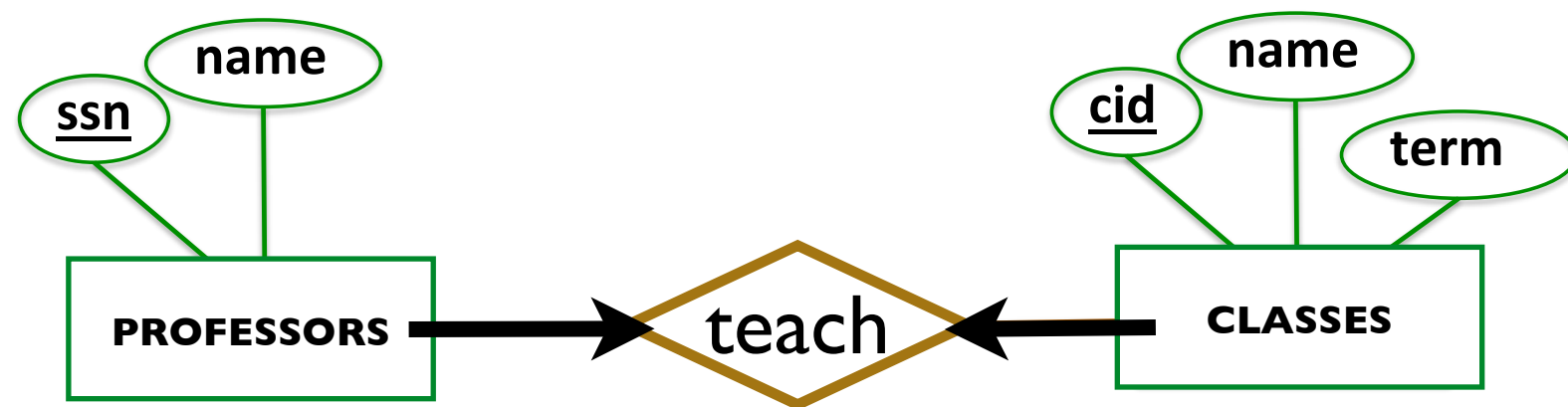
Exercise

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.



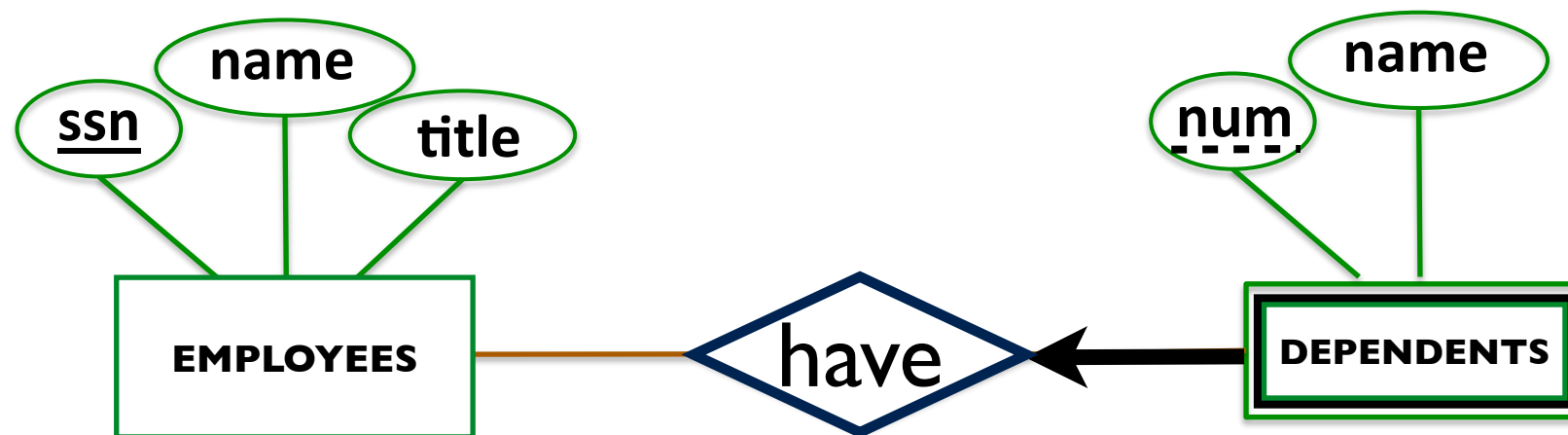
Exercise (solution)

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.



Weak entity sets

- The identifying owner entity and the weak entity must participate in a **one-to-many** relationship set. This relationship set is called the *identifying relationship set* of the weak entity.
- The weak entity must have **total participation** in the identifying relationship set.



```
create table Employees (  
    ssn      char(11) primary key,  
    name     varchar(128),  
    title    varchar(128)  
);
```

```
create table Have_Dependents (  
    ssn      char(11),  
    num      integer,  
    name     varchar(128),  
    primary key (ssn, num),  
    foreign key (ssn) references Employee(ssn)  
    on delete cascade  
);
```

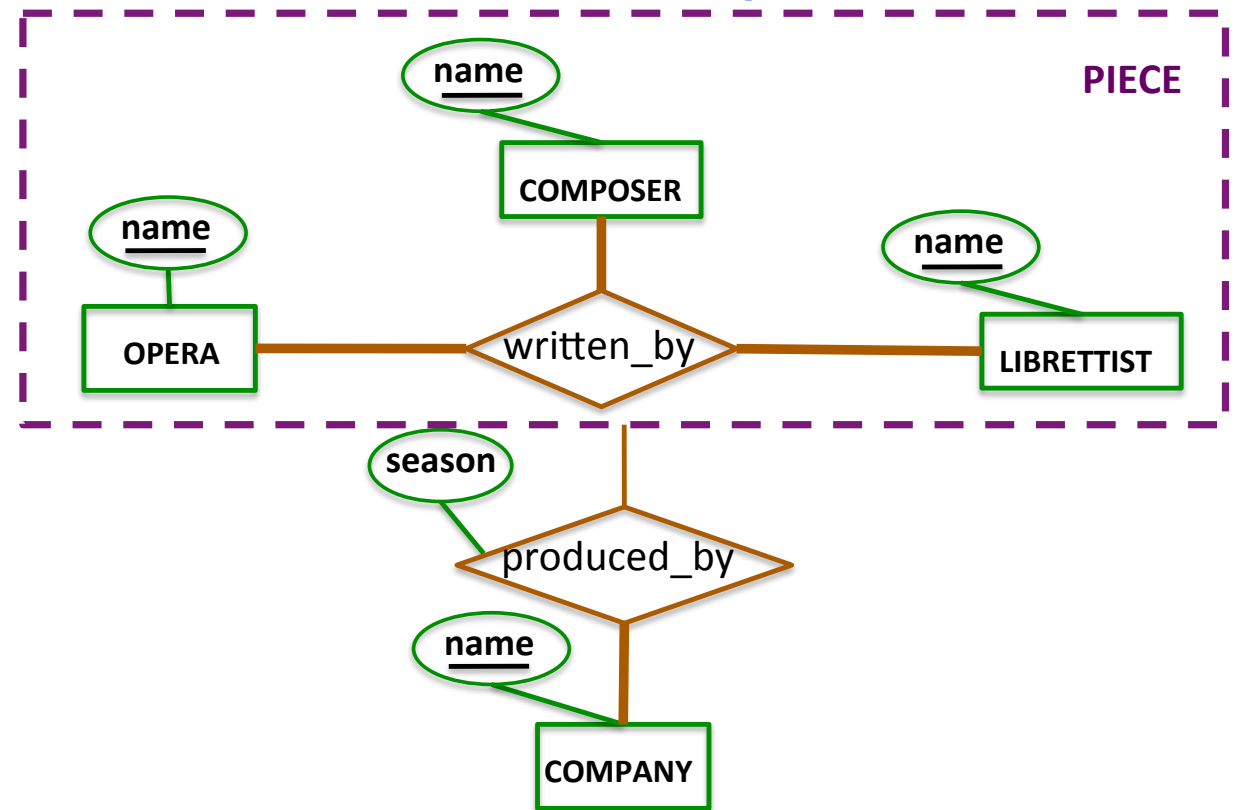
Translating aggregation

An entity cluster (a grouping of entity sets and a relationship set) is a kind of an entity set.

```
create table Opera (
  name      varchar(128) primary key
);
create table Composer (
  name      varchar(128) primary key
);
create table Librettist (
  name      varchar(128) primary key
);
create table Piece (
  opera_name      varchar(64),
  composer_name   varchar(128),
  librettist_name varchar(128),
  primary key (opera_name, composer_name, librettist_name),
  foreign key (opera_name) references Opera(name),
  foreign key (composer_name) references Composer(name),
  foreign key (librettist_name) references Librettist(name)
);
```

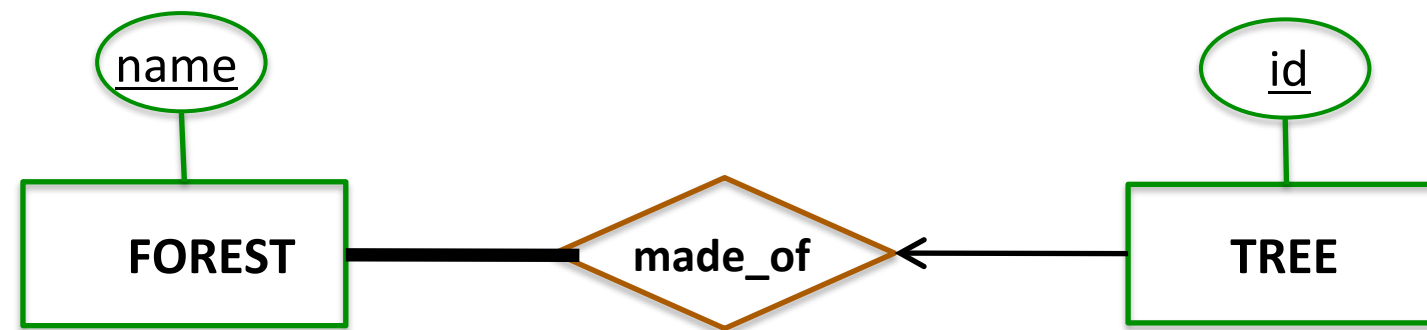
```
create table Company (
  name      varchar(64) primary key
);
```

```
create table Company_Produced_Piece (
  company_name      varchar(64),
  opera_name        varchar(64),
  composer_name     varchar(128),
  librettist_name   varchar(128),
  season            date,
  primary key (company_name, opera_name, composer_name, librettist_name),
  foreign key (opera_name, composer_name, librettist_name)
    references Piece (opera_name, composer_name, librettist_name),
  foreign key (company_name) references Company (name)
);
```



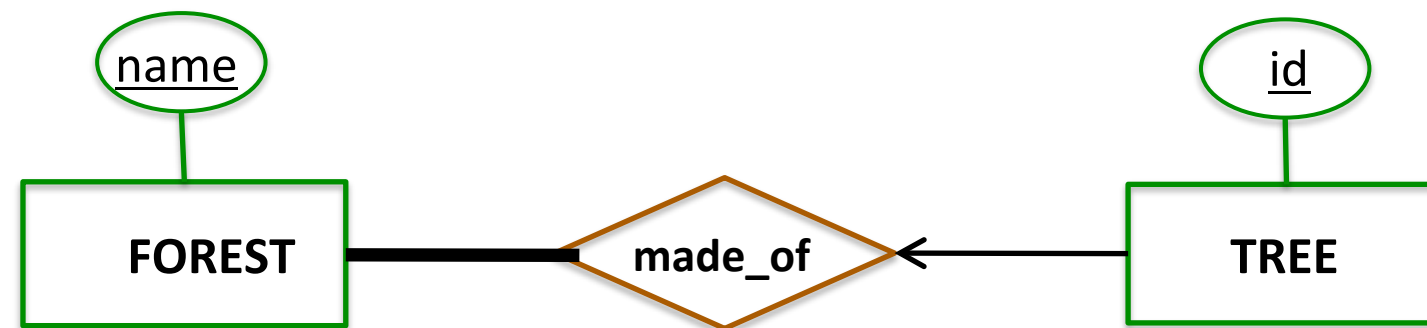
Exercise

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.



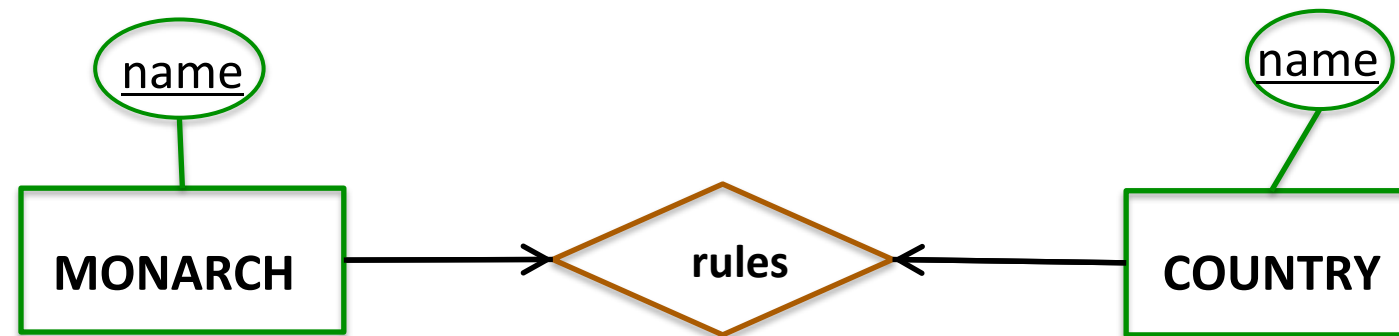
Exercise (solution)

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.



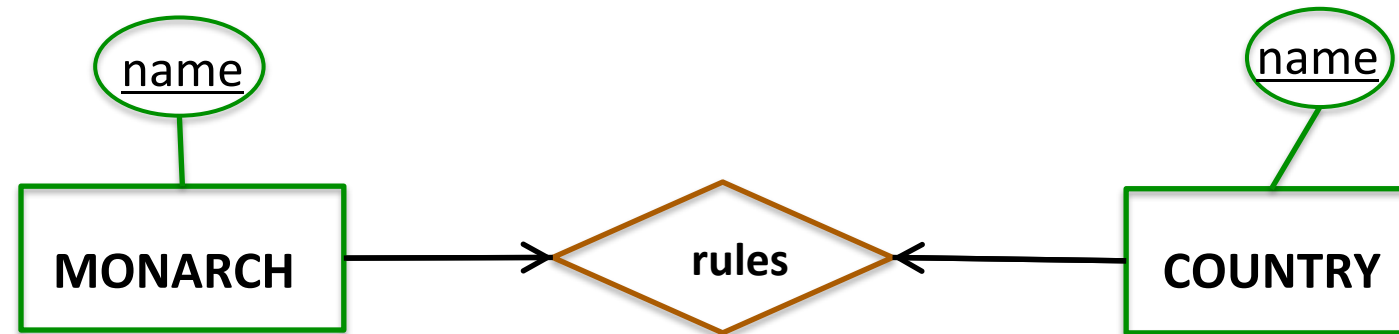
Exercise

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.



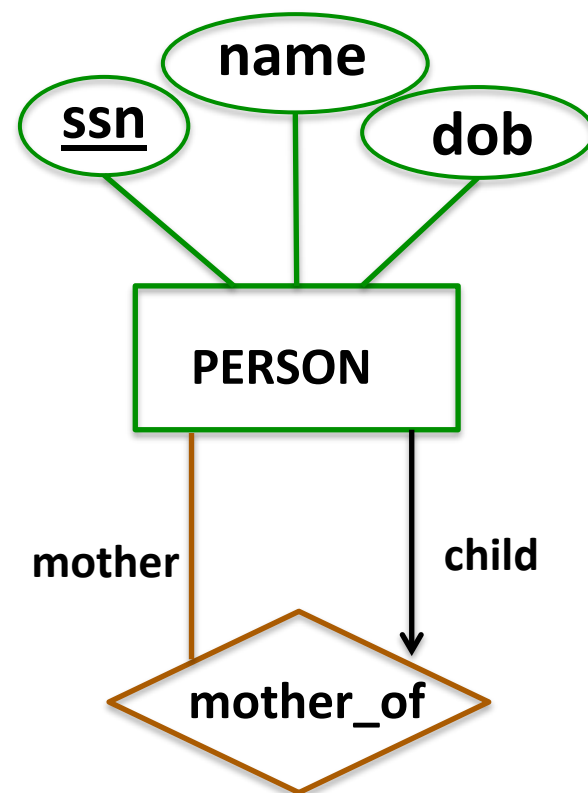
Exercise (solution)

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.



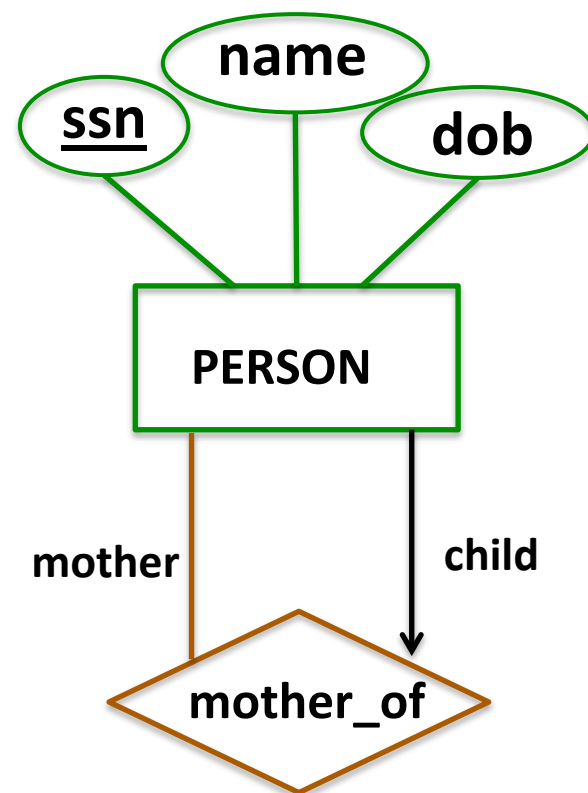
Exercise

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.



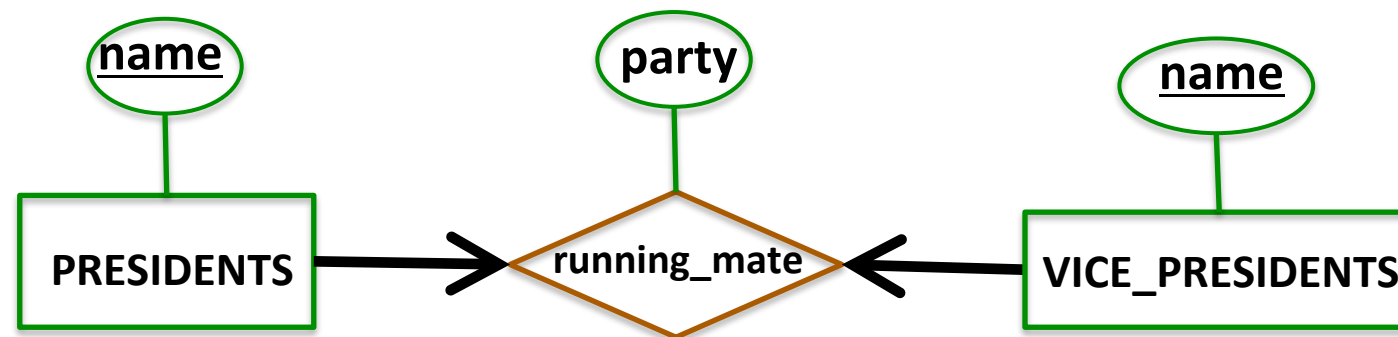
Exercise (solution)

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.



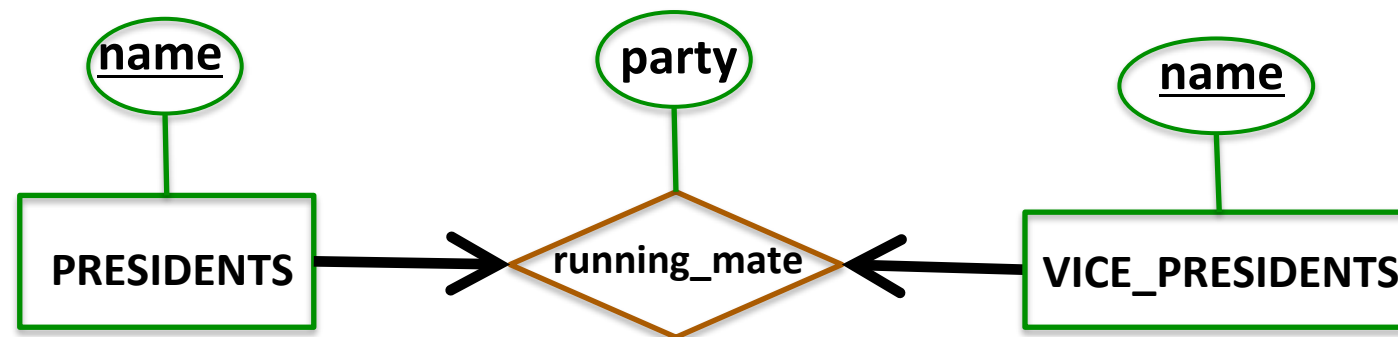
Exercises

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.



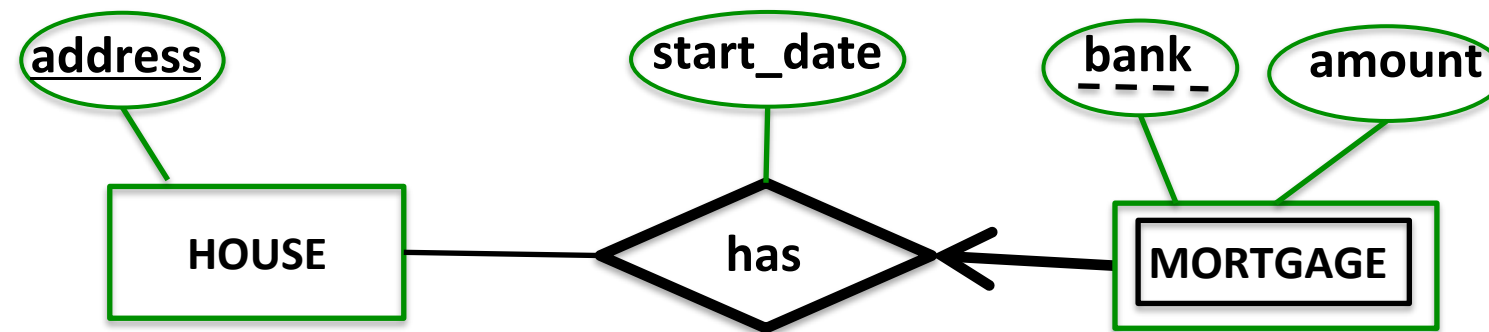
Exercise (solution)

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.



Exercise

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.



Exercise (solution)

Give a relational translation of the following ER diagrams.
Translate the entity sets and the relationship sets as appropriate.

