

# Mapping ER to SQL

---

- Mapping ER to SQL
- Reminder: SQL/Relational Model vs ER Model
- Mapping ER to SQL
- Mapping Strong Entities
- Mapping Weak Entities
- Mapping N:M Relationships
- Mapping 1:N Relationships
- Mapping 1:1 Relationships
- Mapping n-way Relationships
- Mapping Composite Attributes
- Mapping Multi-valued Attributes (MVAs)
- Mapping Subclasses

## ❖ Mapping ER to SQL

---

We have explored mapping ER designs to relational schemas

SQL schemas are essentially more detailed versions of relational schemas

The mapping is much the same, except that

- you need to provide more details on allowed values
- you can map some ideas from ER that are not in relational schemas

There are also some ideas from ER that do not map to an SQL schema

## ◆ Reminder: SQL/Relational Model vs ER Model

---

Correspondences between SQL/relational and ER data models:

- $\text{attribute(ER)} \cong \text{attribute(Rel)}$ ,  $\text{entity(ER)} \cong \text{row/tuple(Rel)}$
- $\text{entity set(ER)} \cong \text{table/relation(Rel)}$ ,  $\text{relationship(ER)} \cong \text{table/relation(Rel)}$

Differences between SQL and ER models:

- SQL uses tables to model entities *and* relationships
- SQL has no composite or multi-valued attributes (only atomic)
- SQL has no object-oriented notions (e.g. subclasses, inheritance)

Note that ...

- not all aspects of ER can be represented exactly in an SQL schema
- some aspects of SQL schemas (e.g. domains) do not appear in ER

## ❖ Mapping ER to SQL

---

Some conventions that we use in mapping ER to SQL

- stop using upper-case for SQL keywords (use **table** vs **TABLE**)
- all tables based on entities are given plural names
- attributes in entities are given the same name in ER and SQL
- attributes in relationships are given the same name in ER and SQL
- ER key attributes are defined using **primary key**
- text-based attributes are defined with type **text**, unless there is a size which is obvious from the context
- attribute domains can be PostgreSQL-specific types where useful
- foreign keys within entity tables are named after the relationship
- foreign keys in relationship tables are named **table\_id**

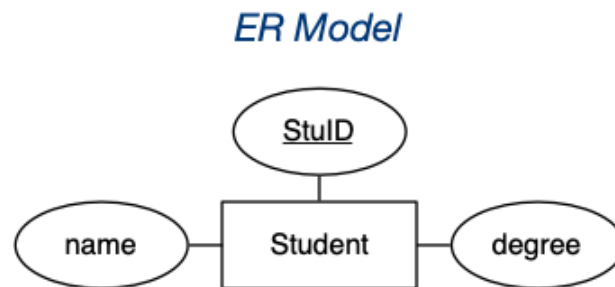
## ❖ Mapping Strong Entities

An entity set  $E$  with atomic attributes  $a_1, a_2, \dots, a_n$

maps to

A table  $R$  with attributes (columns)  $a_1, a_2, \dots, a_n$

Example:



*SQL Version*

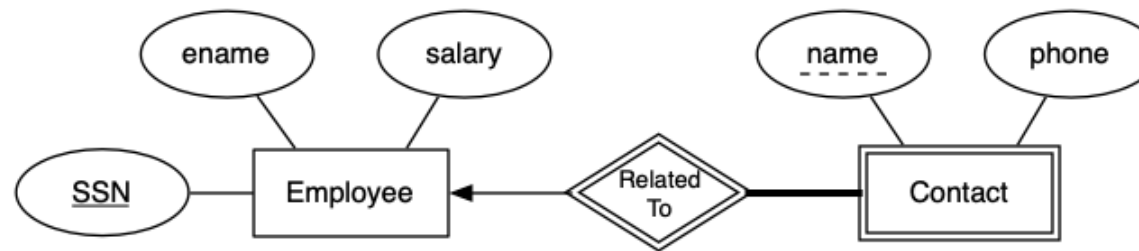
```
create table Students (  
    stuID integer primary key,  
    name text not null,  
    degree char(4)  
);
```

Note: the key is preserved in the mapping.

## ❖ Mapping Weak Entities

Example:

*ER Model*



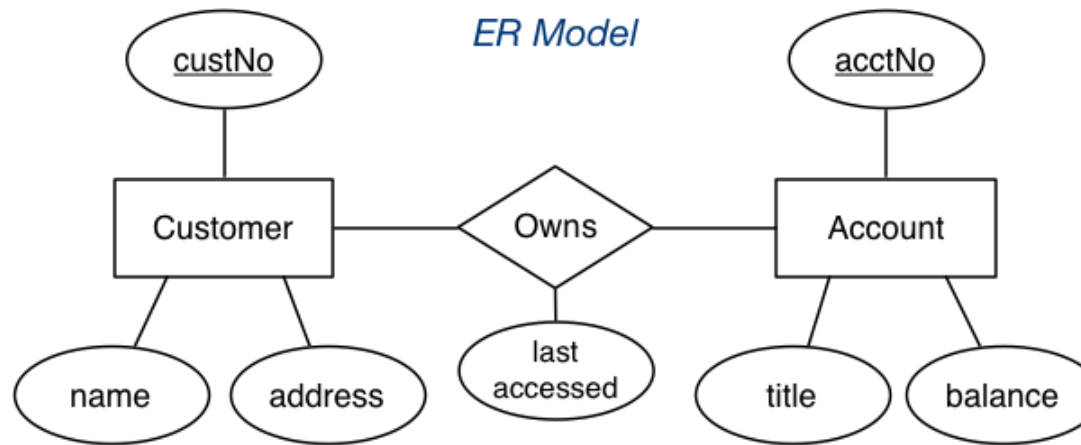
*SQL Version*

```
create table Employees (
    SSN    text primary key,
    ename  text,
    salary currency
);
```

```
create table Contacts (
    relatedTo text not null, -- total participation
    name     text,          -- not null implied by PK
    phone    text not null,
    primary key (relatedTo, name),
    foreign key (relatedTo) references Employees (ssn)
);
```

## ❖ Mapping N:M Relationships

Example:



*Relational Version*

Customer	<b>custNo</b>	name	address
Account	<b>acctNo</b>	title	balance
Owns	<b>acctNo</b>	<b>custNo</b>	lastAccessed

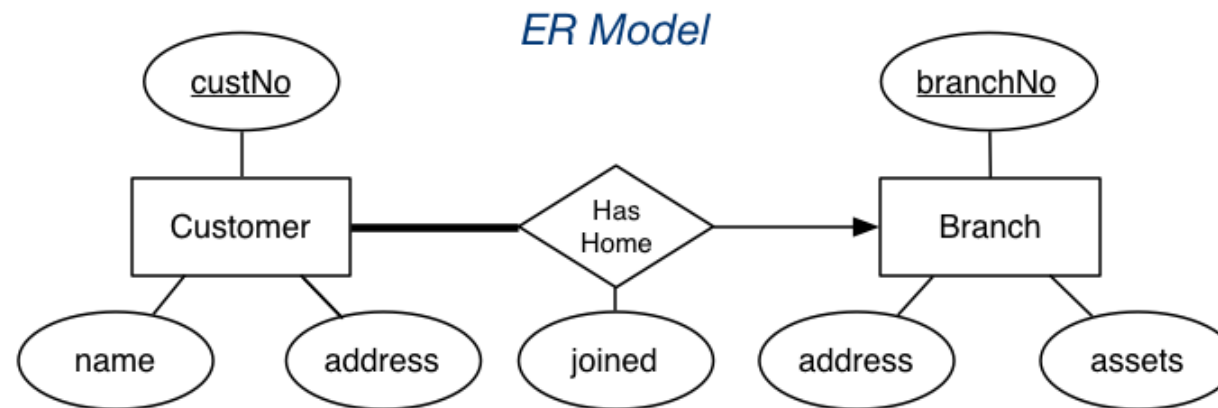
## ❖ Mapping N:M Relationships (cont)

```
create table Customers (  
    custNo  serial primary key,  
    name    text not null,  
    address text -- don't need to know customer's address  
);  
create table Accounts (  
    acctNo  char(5) check (acctNo ~ '[A-Z]-[0-9]{3}'),  
    title   text not null, -- acctNos are like 'A-123'  
    balance float default 0.0,  
    primary key (acctNo)  
);  
create table Owns (  
    customer_id integer references Customers(custNo),  
    account_id   char(5) references Accounts(acctNo),  
    last_accessed timestamp,  
    primary key (customer_id, account_id)  
);
```



## ❖ Mapping 1:N Relationships

Example:



*Relational Version*

Customer	<b>custNo</b>	name	address	branchNo	joined
----------	---------------	------	---------	----------	--------

Branch	<b>branchNo</b>	address	assets
--------	-----------------	---------	--------

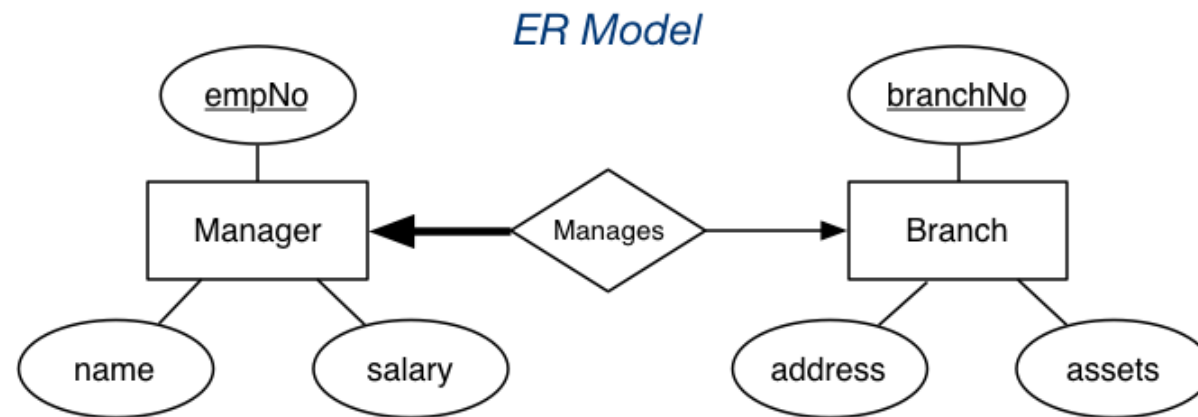
## ❖ Mapping 1:N Relationships (cont)

```
create table Branches (  
    branchNo serial primary key,  
    address  text not null,  
    assets   currency  
);  
create table Customers (  
    custNo  serial primary key,  
    name    text not null,  
    address text,  
    hasHome integer not null, -- total participation  
    joined  date not null,  
    foreign key (hasHome) references Branches(branchNo)  
);
```

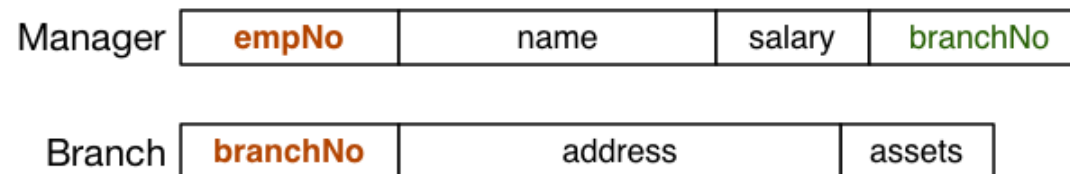
**hasHome** implements the 1:n relationship; **not null** implements total participation

## ❖ Mapping 1:1 Relationships

Example:



*Relational Version*



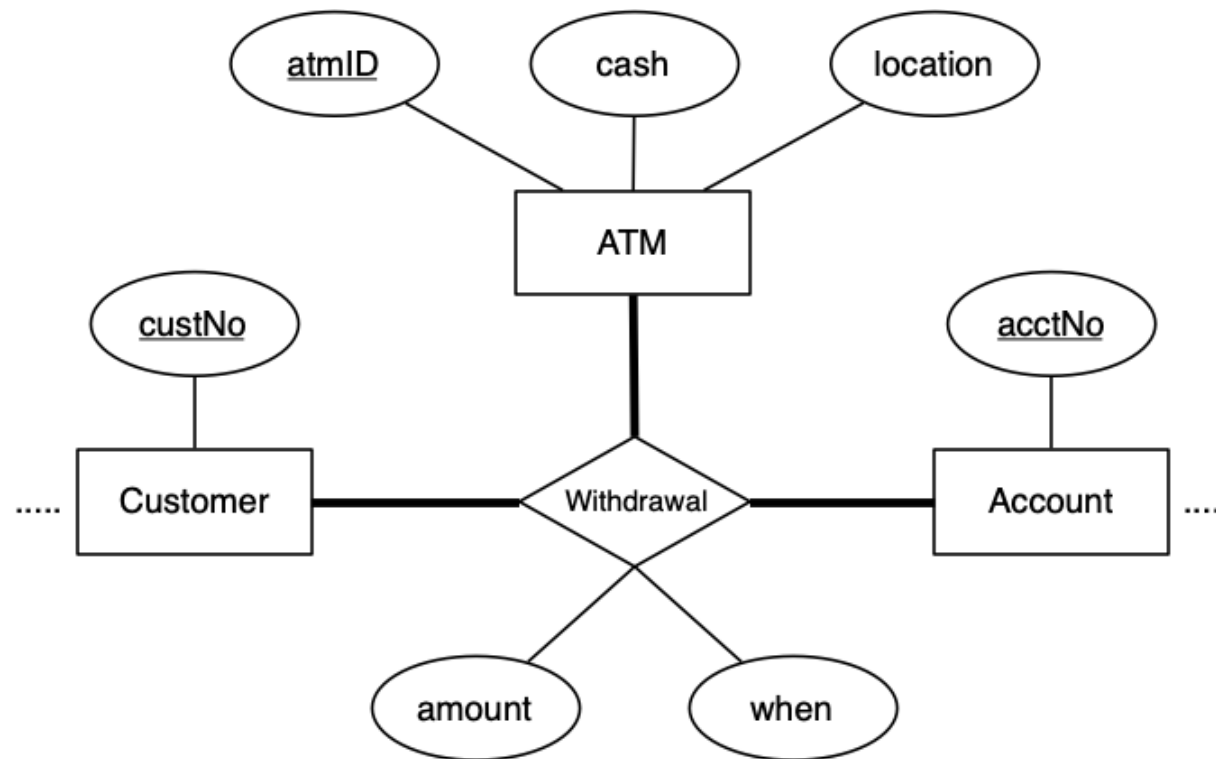
## ❖ Mapping 1:1 Relationships (cont)

```
create table Branches (  
    branchNo serial primary key,  
    address  text not null,  
    assets   currency          -- a new branch  
);                                     -- may have no accounts  
create table Managers (  
    empNo    serial primary key,  
    name     text not null,  
    salary   currency not null, -- when first employed,  
                                   -- must have a salary  
    manages  integer not null, -- total participation  
    foreign key (manages) references Branches(branchNo)  
);
```

If both entities have total participation, cannot express this in SQL except by putting a (redundant) **not null** foreign key in one table

## ❖ Mapping n-way Relationships

Example:



A customer accesses one of their accounts at a specific ATM

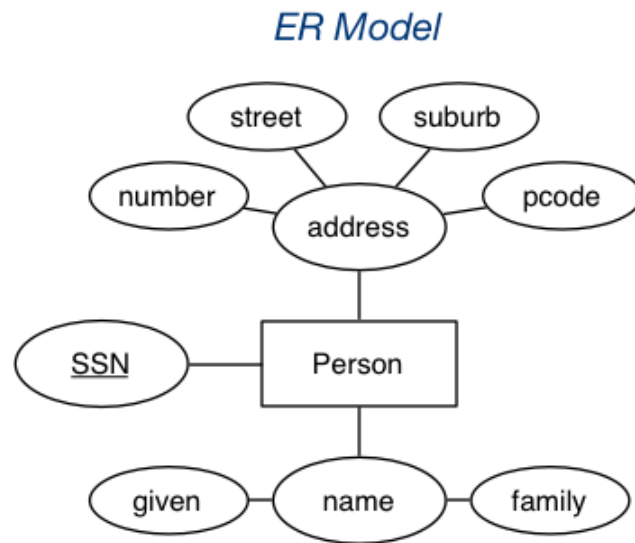
## ❖ Mapping n-way Relationships (cont)

```
create table Customers (  
    custNo    serial primary key, ...  
);  
create table Accounts (  
    acctNo    char(5) ... primary key, ...  
);  
create table ATMs (  
    atmID     serial primary key,  
    cash      currency check (cash >= 0),  
    location  text not null  
);  
create table Withdrawal (  
    customer_id integer references Customers(custNo),  
    account_id   char(5) references Accounts(acctNo),  
    atm_id       integer references ATMs(atmID),  
    amount       currency not null,  
    when         timestamp default now(),  
    primary key  (customer_id,account_id,atm_id)  
);
```

## ❖ Mapping Composite Attributes

Composite attributes are mapped by concatenation or flattening.

Example:



### *Relational Version #1*

Person		
<b>SSN</b>	name	address

### *Relational Version #2*

Person			
<b>SSN</b>	given	family	.....
.....	number	street	suburb pcode

## ❖ Mapping Composite Attributes (cont)

```
-- Version 1: concatenated
create table People (
    ssn      integer primary key,
    name     text not null,
    address  text not null
);

-- Version 2: flattened
create table People (
    ssn      integer primary key,
    given    text not null,
    family   text,
    number   integer not null,
    street   text not null,
    suburb   text not null,
    pcode    char(4) not null check (pcode ~ '[0-9]{4}')
);
```

**address = (number::text||' '||street||', '||suburb||' '||pcode)**

Searching: **suburb = 'Coogee' vs address like '%Coogee%'**

Sorting: **order by family** vs can't be done (easily)

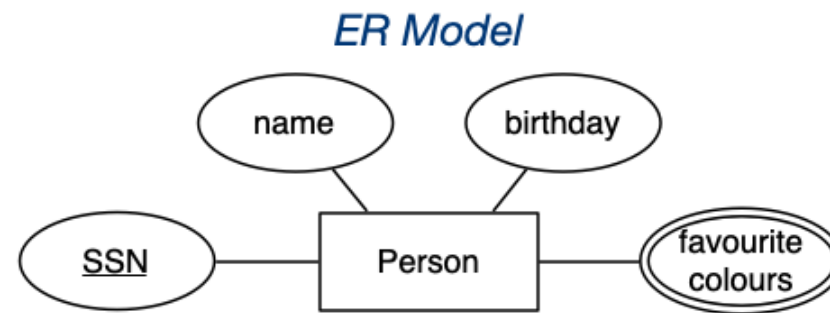




## ❖ Mapping Multi-valued Attributes (MVAs)

MVAs are mapped by a new table linking values to their entity.

Example:



*Relational Version*

<i>Person</i>	<b>SSN</b>	name	birthday
---------------	------------	------	----------

<i>FavColour</i>	<b>SSN</b>	colour
------------------	------------	--------

## ❖ Mapping Multi-valued Attributes (MVAs) (cont)

```
create table People (  
    ssn      integer primary key,  
    name     text not null,  
    birthday date  
);  
create table FavColour (  
    person_id integer references People(ssn),  
    colour     text,  
    primary key (person_id, colour)  
);
```

Note that **colour** is implicitly **not null** because it is part of the primary key

## ❖ Mapping Subclasses

---

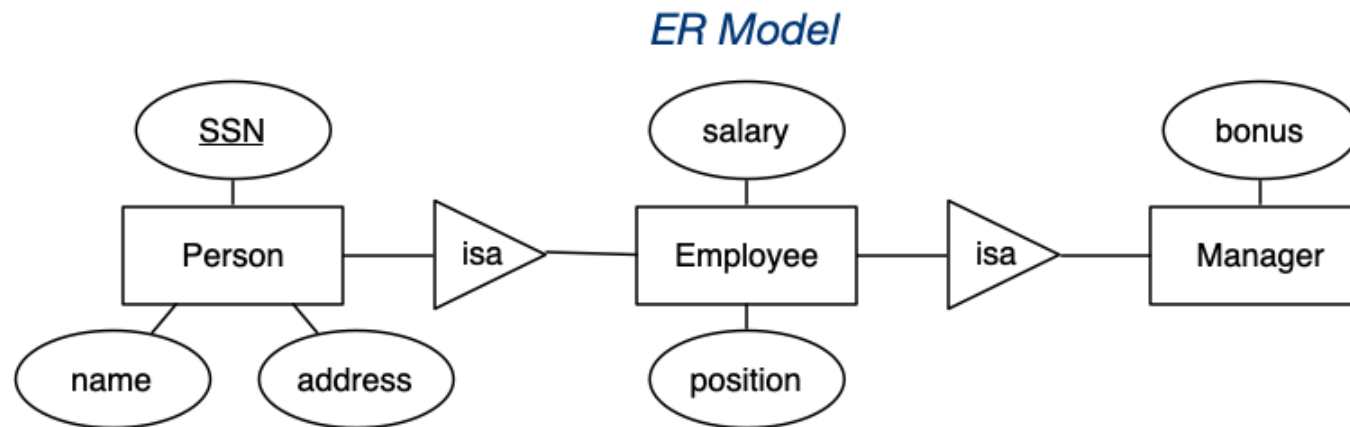
Three different approaches to mapping subclasses to tables:

- ER style
  - each entity becomes a separate table,
  - containing attributes of subclass + FK to superclass table
- object-oriented
  - each entity becomes a separate table,
  - inheriting all attributes from all superclasses
- single table with nulls
  - whole class hierarchy becomes one table,
  - containing all attributes of all subclasses (null, if unused)

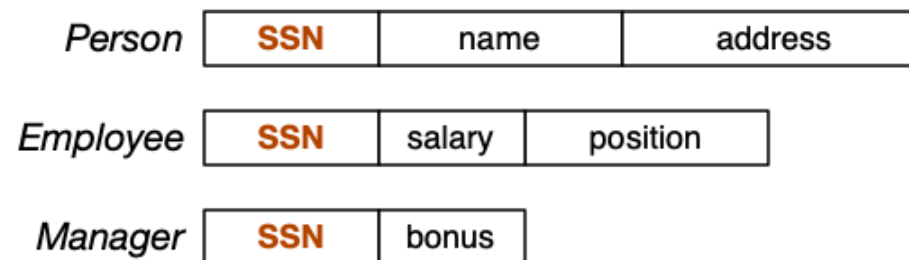
Which mapping is best depends on how data is to be used.

## ❖ Mapping Subclasses (cont)

Example of ER-style mapping:



### *Relational Version*

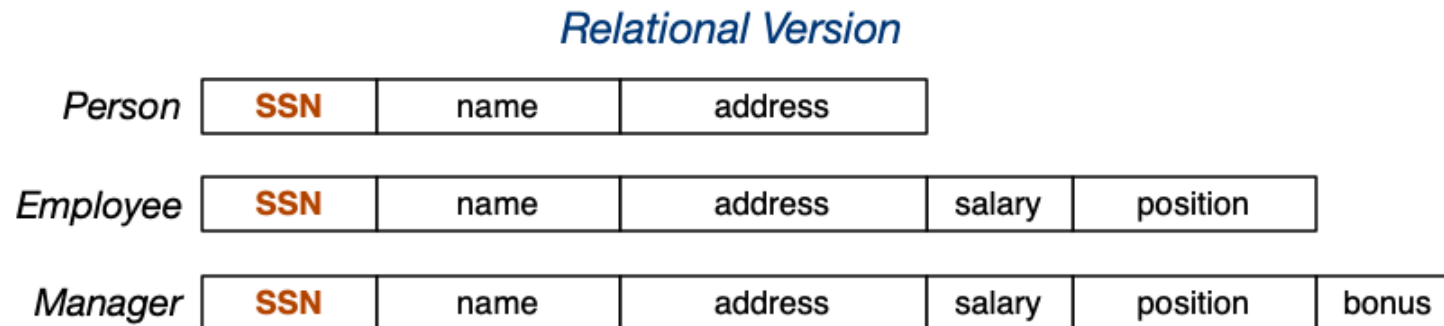
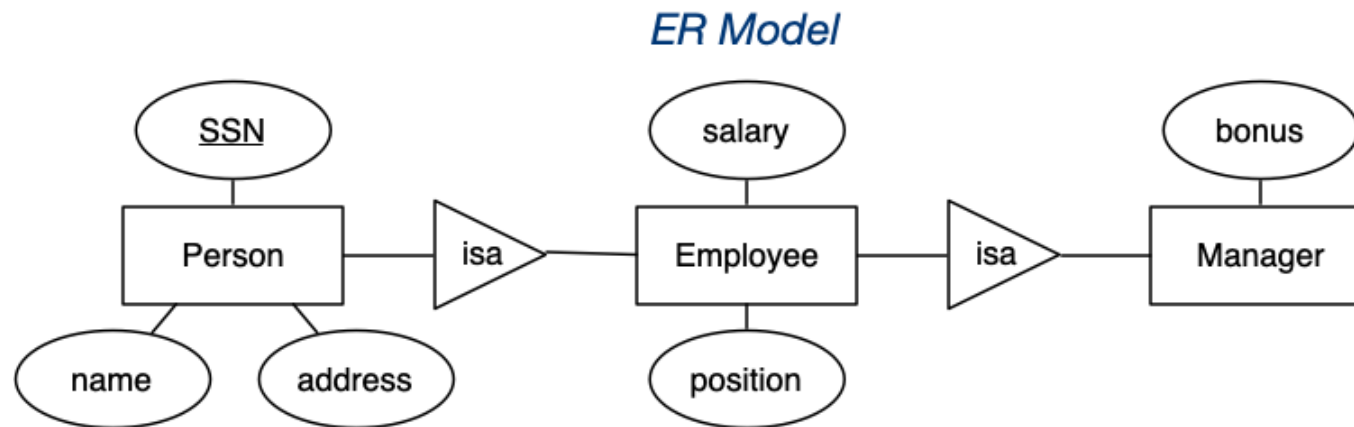


## ❖ Mapping Subclasses (cont)

```
create table People (  
    ssn      integer primary key,  
    name     text not null,  
    address  text  
);  
create table Employees (  
    person_id integer primary key,  
    salary     currency not null,  
    position   text not null,  
    foreign key (person_id) references People(ssn)  
);  
create table Managers (  
    employee_id integer primary key,  
    bonus        currency,  
    foreign key (employee_id)  
                references Employees(person_id)  
);
```

## ❖ Mapping Subclasses (cont)

Example of object-oriented mapping:



## ❖ Mapping Subclasses (cont)

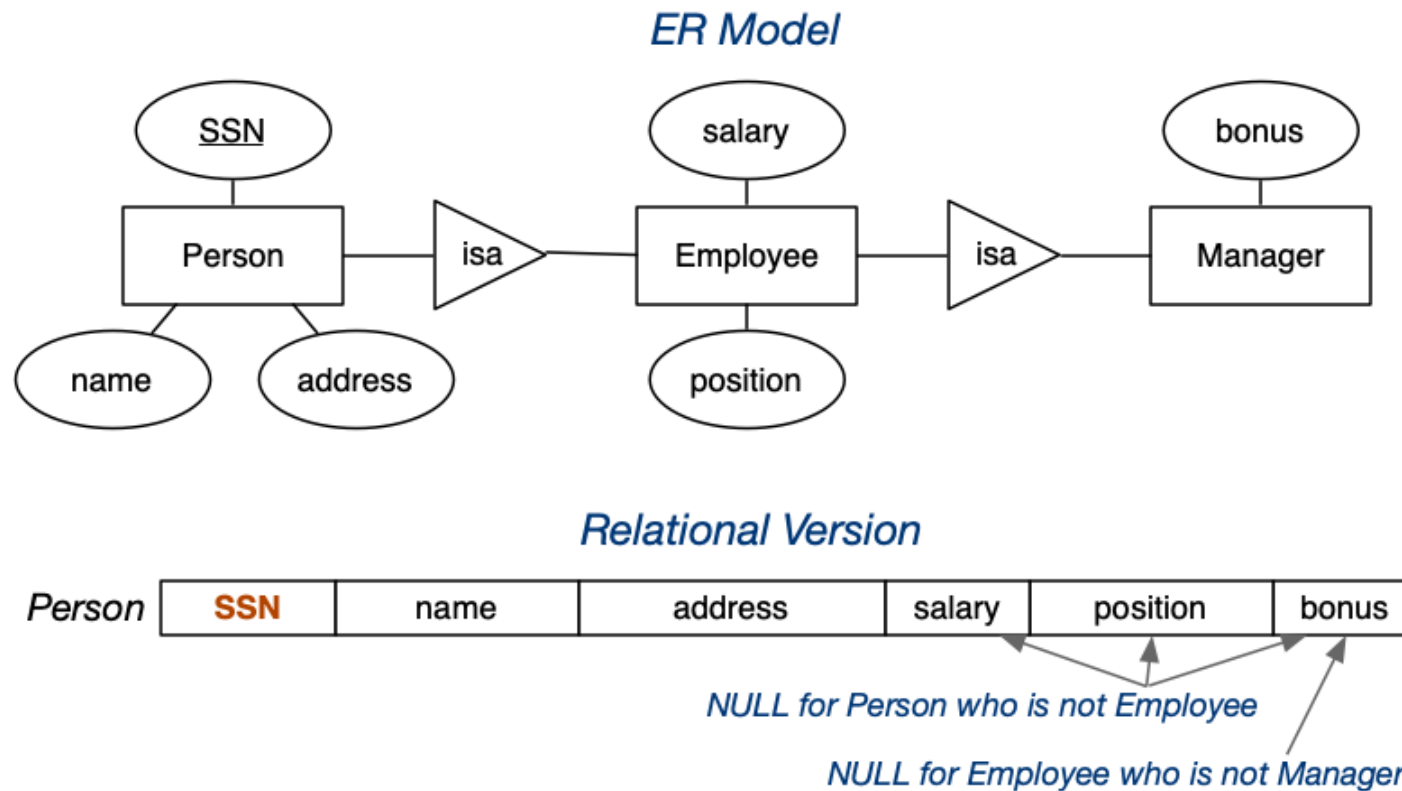
```
create table People (  
    ssn      integer primary key,  
    name     text not null,  
    address  text  
);  
create table Employees (  
    ssn      integer primary key,  
    name     text not null,  
    address  text  
    salary   currency not null,  
    position text not null,  
    foreign key (ssn) references People(ssn)  
);  
create table Managers (  
    ssn      integer primary key,  
    name     text not null,  
    address  text  
    salary   currency not null,  
    position text not null,  
    bonus    currency,  
    foreign key (ssn) references People(ssn)  
);
```





## ❖ Mapping Subclasses (cont)

Example of single-table-with-nulls mapping:

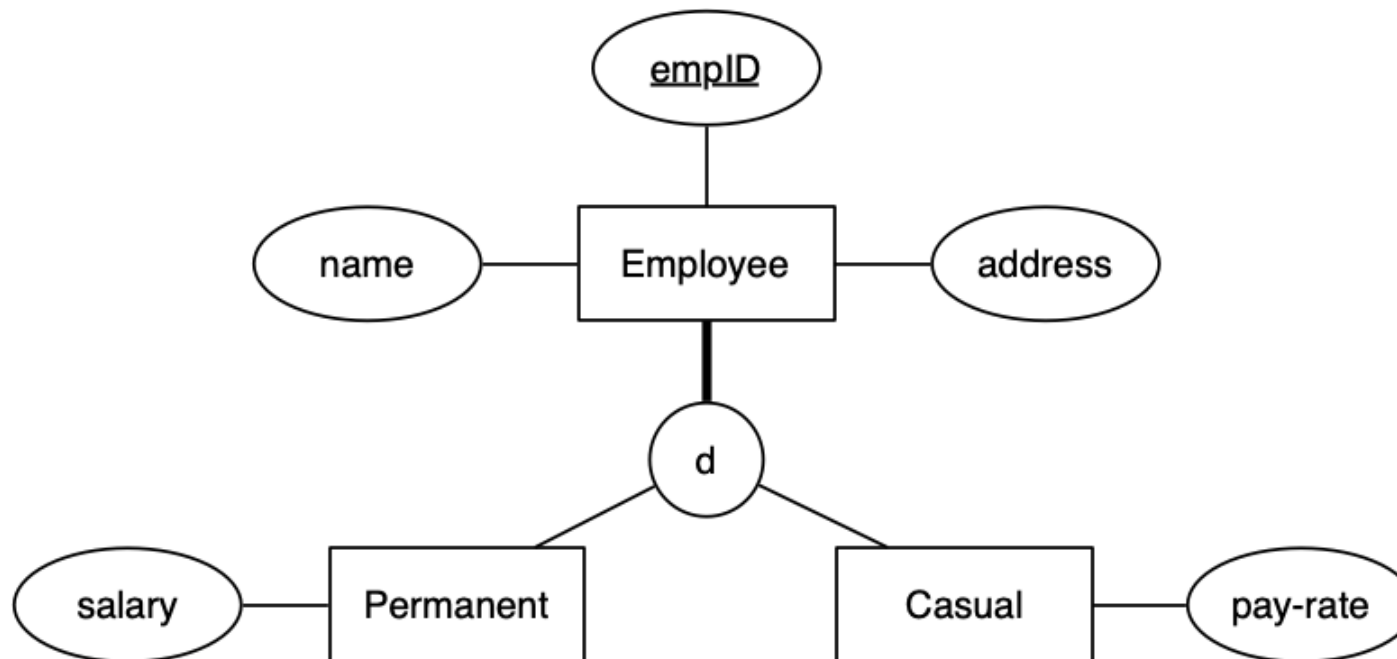


## ❖ Mapping Subclasses (cont)

```
create table People (  
    ssn          integer primary key,  
    ptype        char(1) not null  
                check (ptype in ('P','E','M')),  
    name         text not null,  
    address      text  
    salary       currency,  
    position     text,  
    bonus        currency,  
    constraint subclasses check  
        ((ptype = 'P' and salary is null  
          and position is null and bonus is null)  
         or  
         (ptype = 'E' and salary is not null  
          and position is not null and bonus is null)  
         or  
         (ptype = 'M' and salary is not null  
          and position is not null and bonus is not null))  
);
```

## ❖ Mapping Subclasses (cont)

Example:



Every employee is either permanent or casual, but not both.

## ❖ Mapping Subclasses (cont)

ER-style mapping to SQL schema:

```
create table Employees (  
    empID    serial primary key,  
    name     text not null,  
    address  text not null  
);  
create table Permanents (  
    employee_id integer primary key,  
    salary      currency not null,  
    foreign key (employee_id) references Employees(empID)  
);  
create table Casuals (  
    employee_id integer primary key,  
    pay_rate    currency not null,  
    foreign key (employee_id) references Employees(empID)  
);
```

Does *not* capture either participation or disjoint-ness constraints!

Would need to program a solution to this e.g web-form that requires user to enter both Employee and subclass info



Produced: 22 Sep 2020