>>

COMP3311 Week 2 Wednesday Lecture

- Week 02 Wednesday
- Assignment 1 Database
- Recap
- Exercise: ER-to-SQL for Beer Database
- Mapping Composite Attributes
- Mapping Multi-valued Attributes (MVAs)
- Mapping Subclasses
- Exercise: ER-to-SQL (1)
- Exercise: ER-to-SQL (2)
- Exercise: ER-to-SQL (3)
- PostgreSQL Databases
- **`psql`**
- Exercise: Creating a database
- Populating a Database
- Exercise: Inserting Tuples
- Exercise: More Inserting Tuples
- Bulk Insertion
- Dump/Restore
- Exercise: Playing with Beer Database

∧ >>

❖ Week 02 Wednesday

In today's lecture ...

- More ER → SQL, and SQL DDL
- Building/restoring a database

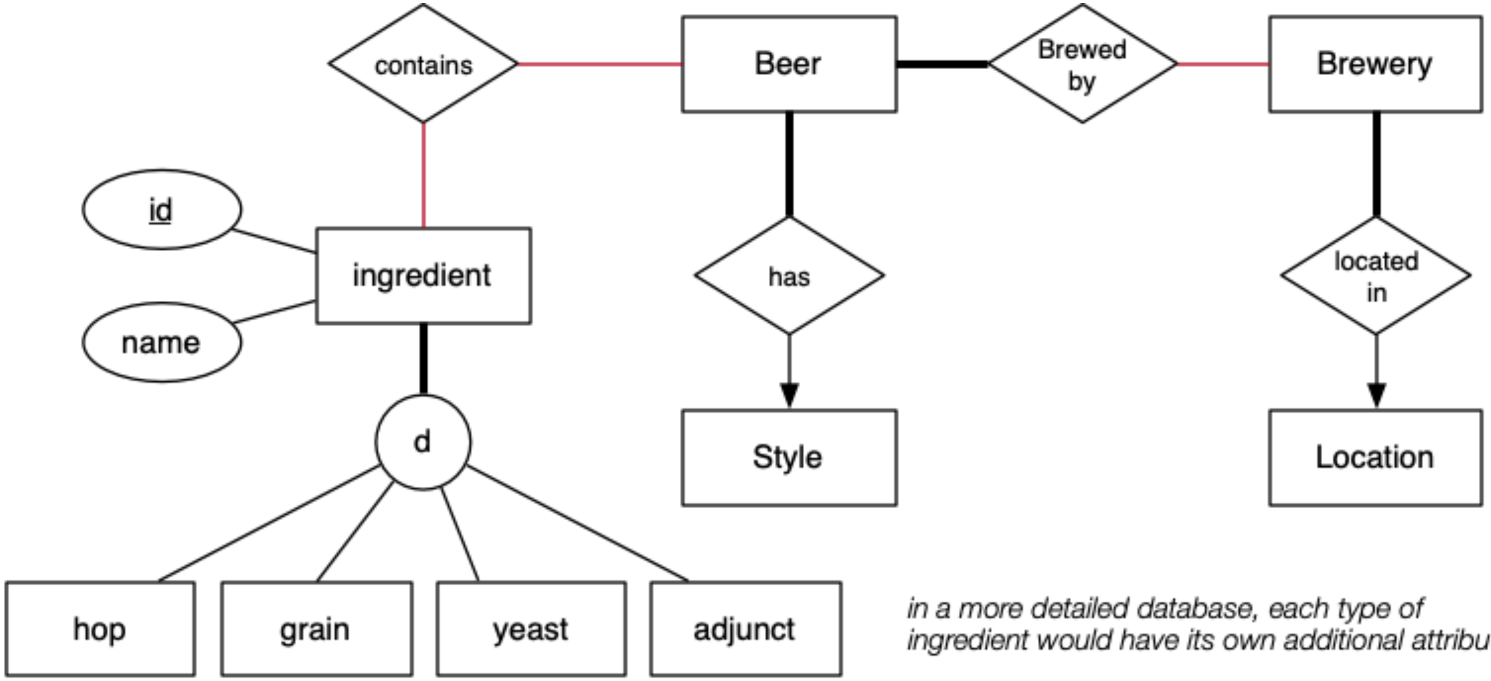
Things to do ...

- Quiz before Friday midnight
- Set up your PostgreSQL server
  (300 students have logged in to vxdb2 and have /localstorage)
- Help Session, Friday 4pm, Location: TBA

<< ∧ >>

❖ Assignment 1 Database

Database about beer and breweries



in a more detailed database, each type of ingredient would have its own additional attributes

<< ^ >>

❖ Assignment 1 Database (cont)

Details of entities, with example data (does not include all entities):

```
Beers(id, name, brewed, style, abv, ibu, sold_in, volume, notes, rating)

(123, 'VB', 2020, *Lager, 5.0, 30, can, 375, 'Worst beer in world', 1)

Brewers(id, name, founded, website, located_in)

(321, 'Carlton', 1899, 'www.carlton.com.au', *Melbourne)

Styles(id, name, min_abv, max_abv)

(456, 'Lager', 4.0, 6.0)

Ingredients(id, itype, name)

(654, 'hop', 'Cascade')

Brewed_by(beer, brewery)

(*VB, *Carlton) ... represented as (123, 321)
```

<< ∧ >>

❖ Recap

ER → Relational/SQL Mapping

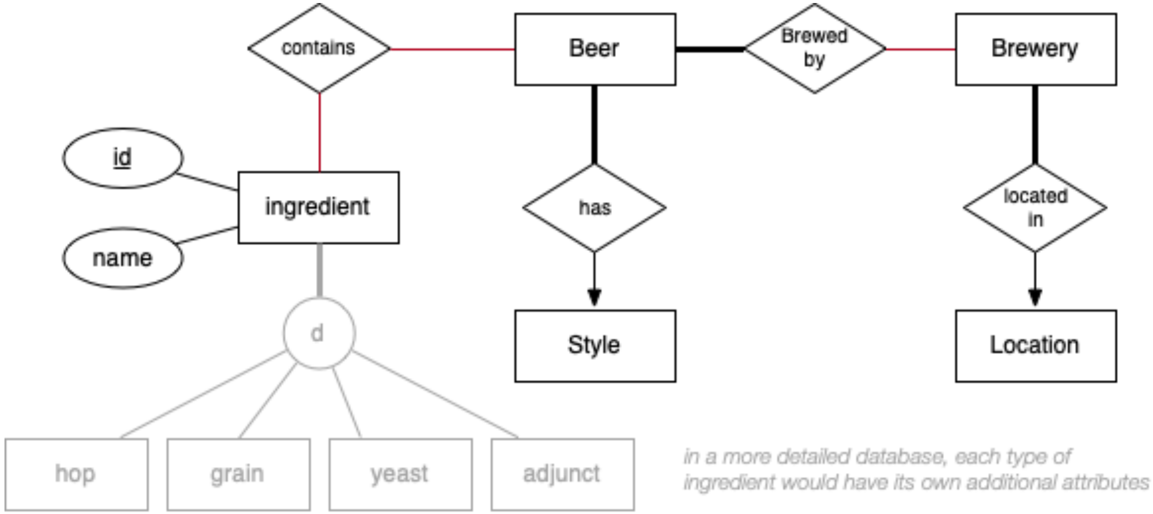
- attributes → attributes
- entities → relations/tables
- 1:1 relationships → foreign key
- 1:n relationships → foreign key
- n:m relationships → link table
- composite attributes → attributes
- multi-valued attributes → table

<< ∧ >>

❖ Exercise: ER-to-SQL for Beer Database

Convert the beer ER data model to an SQL schema

- assume that each entity has attributes **id** and **name**
- treat **Ingredient** as simple entity; ignore sub-classes



in a more detailed database, each type of ingredient would have its own additional attributes

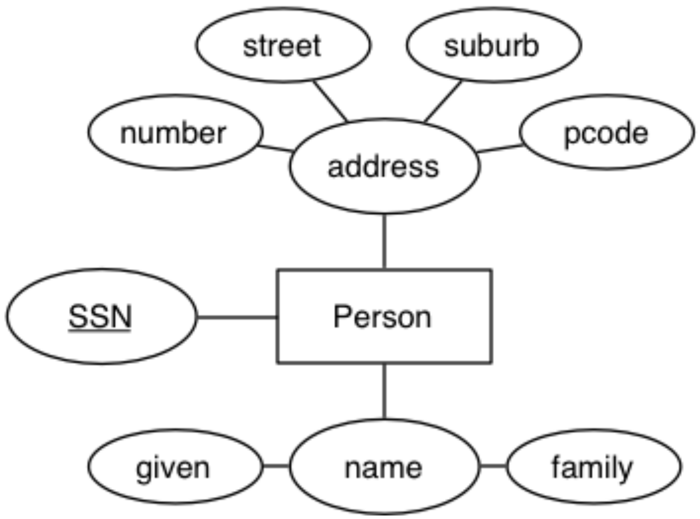
<< ^ >>

❖ Mapping Composite Attributes

Composite attributes are mapped by concatenation or flattening.

Example:

ER Model



Relational Version #1

Person

SSN	name	address

Relational Version #2

Person

SSN	given	family

.....

.....

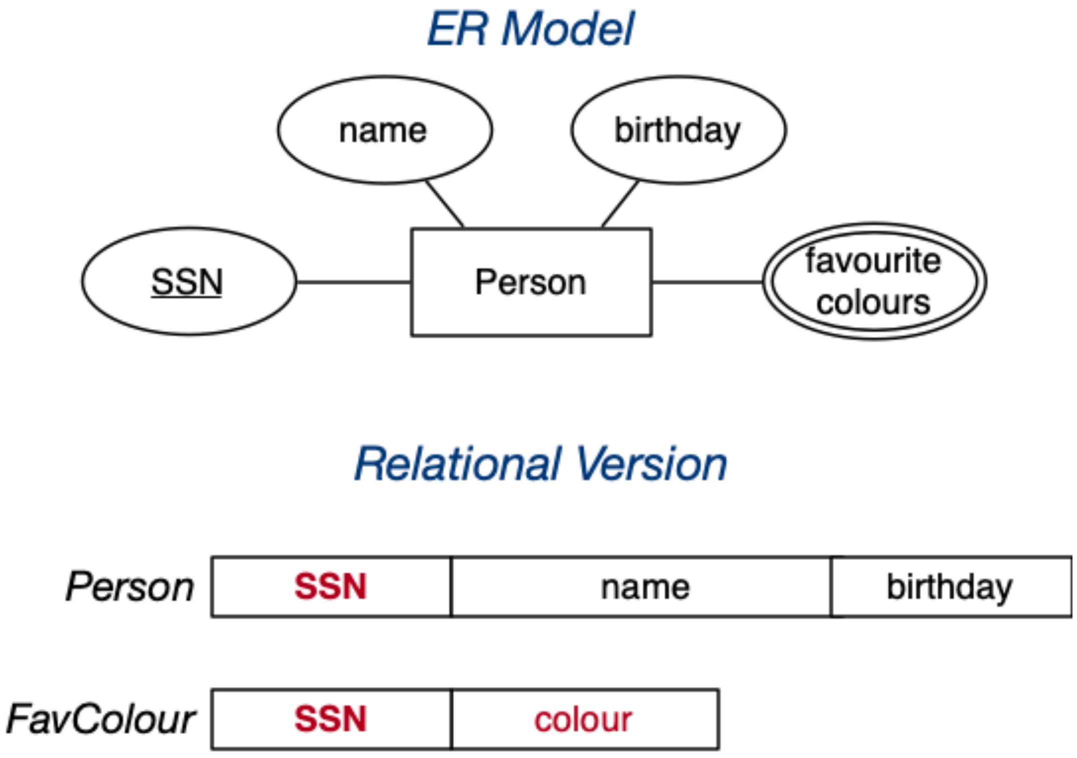
number	street	suburb	pcode

<< ∧ >>

❖ Mapping Multi-valued Attributes (MVAs)

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

Example:



ER Model

Relational Version

Person	SSN	name	birthday
FavColour	SSN	colour	

<< ∧ >>

❖ 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)

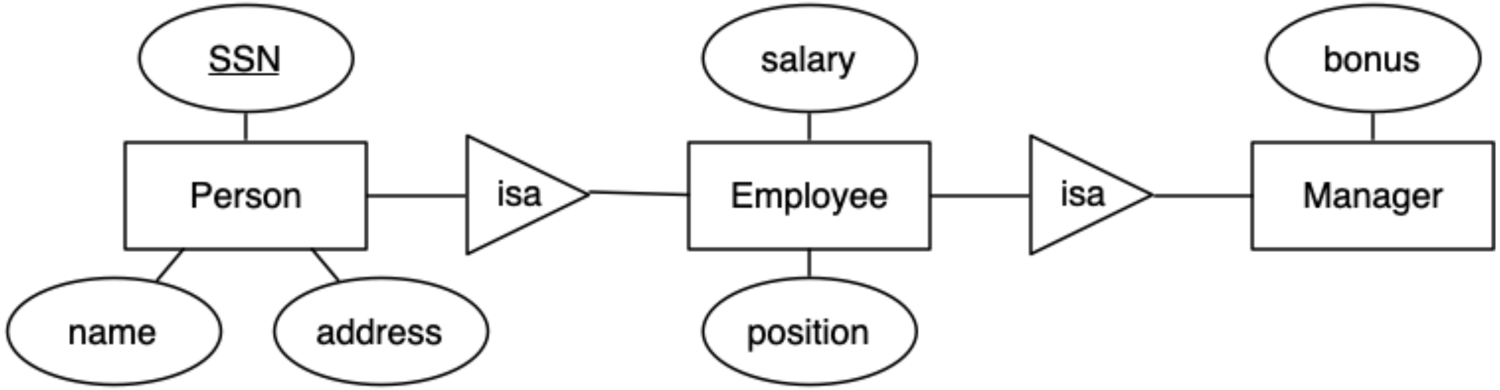
Which mapping is best depends on how data is to be used.

<< ^ >>

❖ Mapping Subclasses (cont)

Example of ER-style mapping:

ER Model



Relational Version

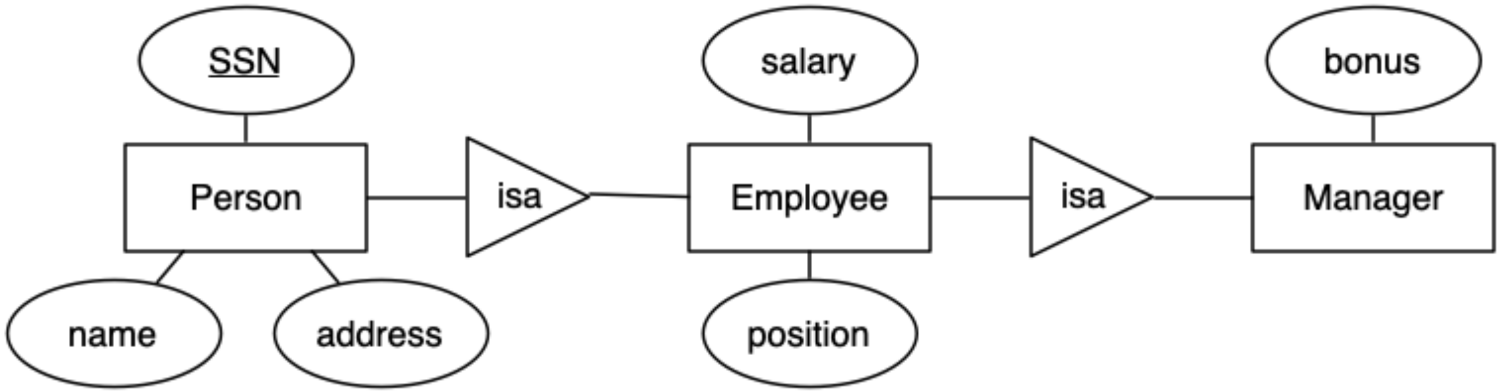
Person	**SSN**	name	address
Employee	**SSN**	salary	position
Manager	**SSN**	bonus	

<< ∧ >>

❖ Mapping Subclasses (cont)

Example of object-oriented mapping:

ER Model



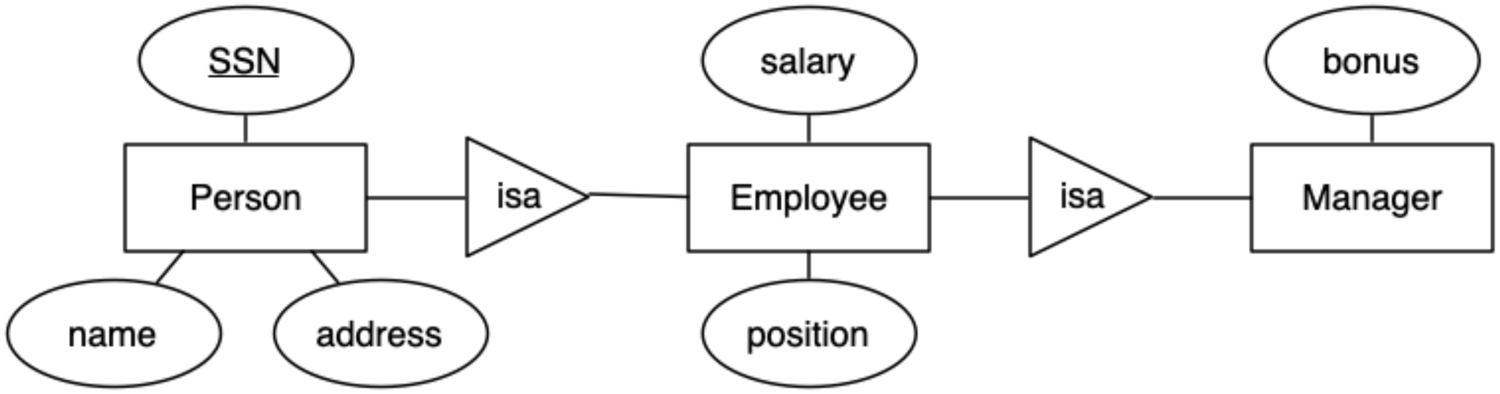
Relational Version

Person	SSN	name	address			
Employee	SSN	name	address	salary	position	
Manager	SSN	name	address	salary	position	bonus

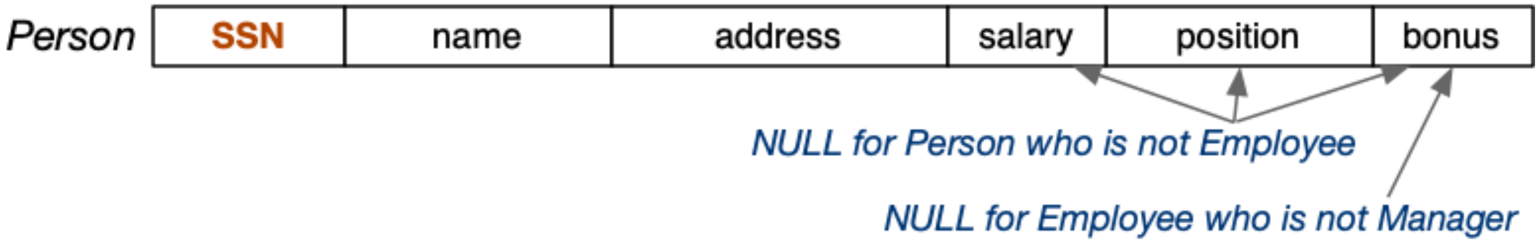
❖ Mapping Subclasses (cont)

Example of single-table-with-nulls mapping:

ER Model



Relational Version



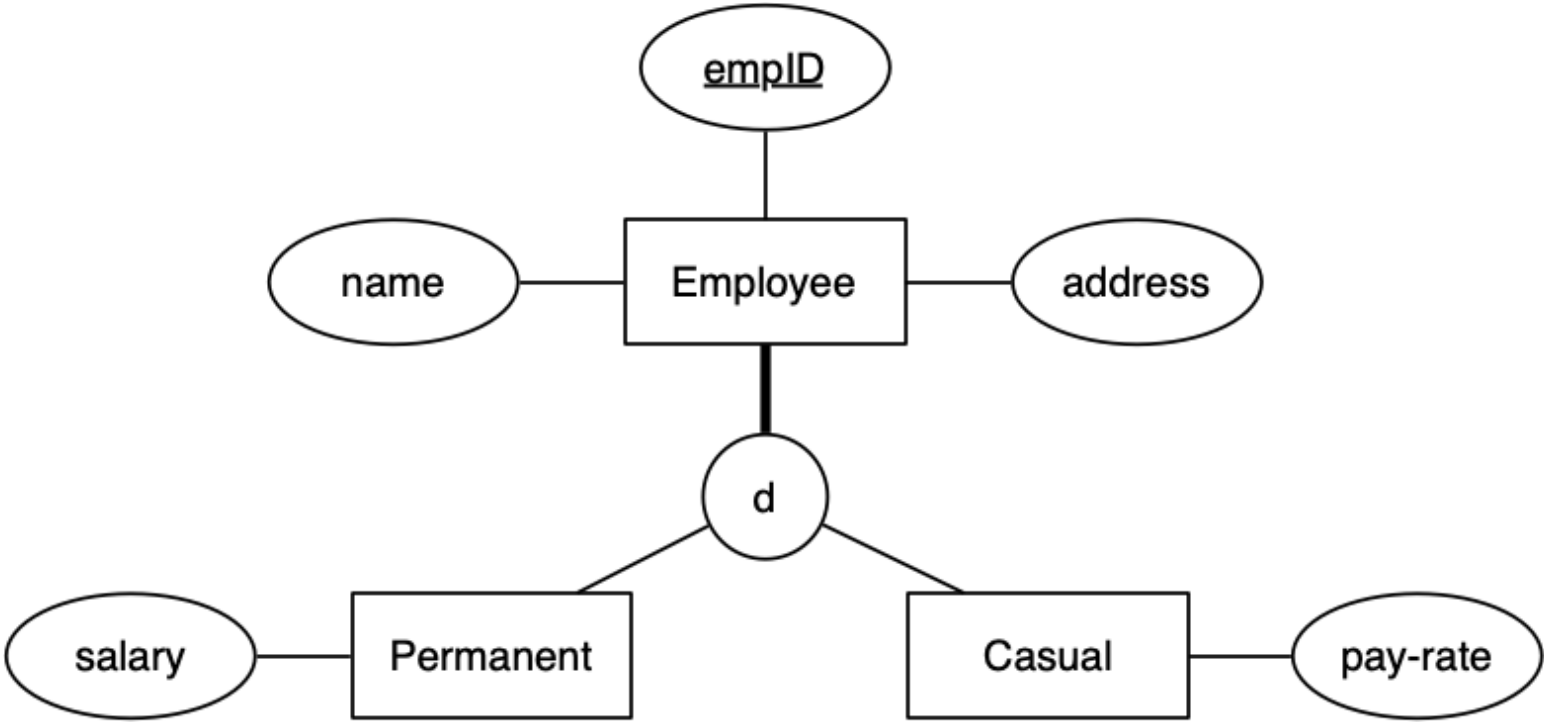
Person	**SSN**	name	address	salary	position	bonus

NULL for Person who is not Employee

NULL for Employee who is not Manager

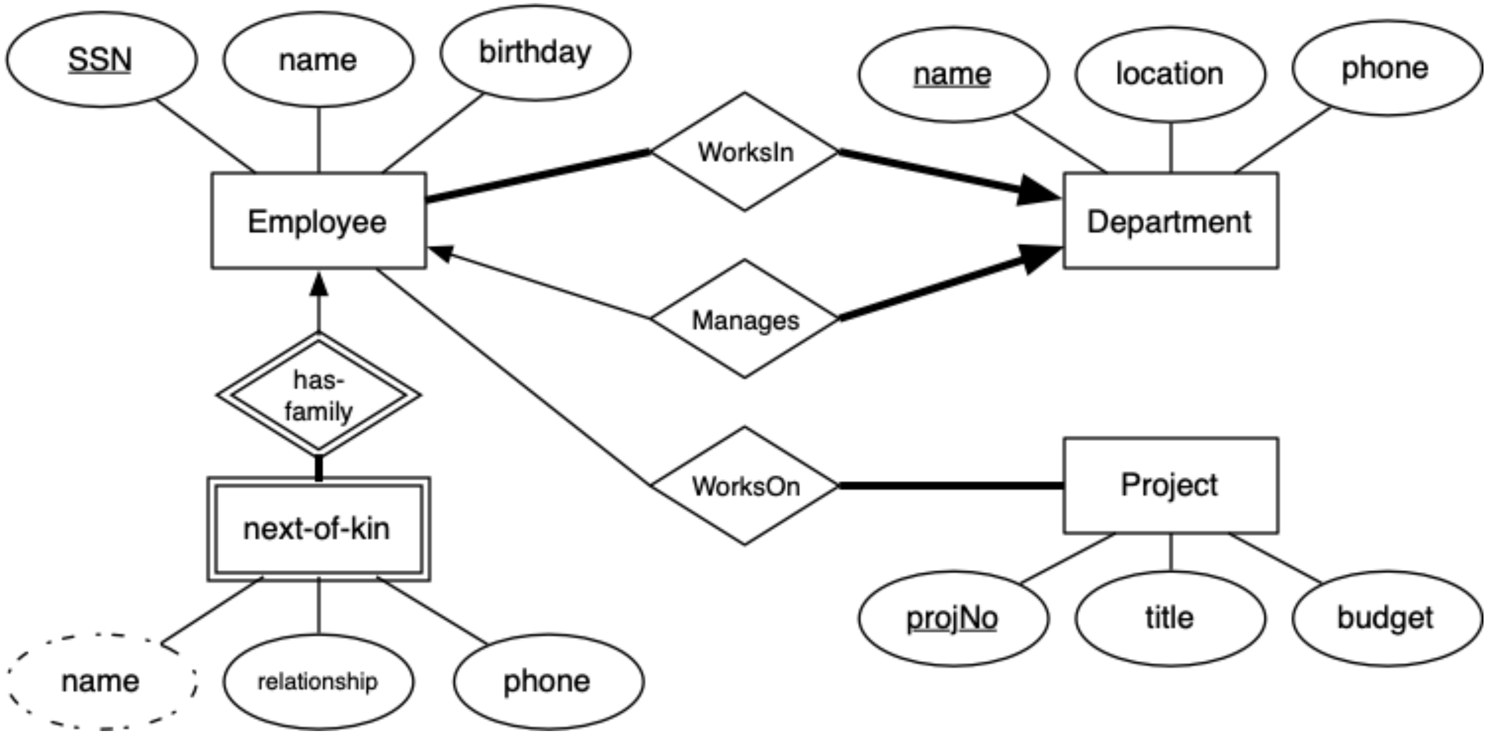
❖ Exercise: ER-to-SQL (1)

Convert the following class hierarchy to SQL using ER mapping:



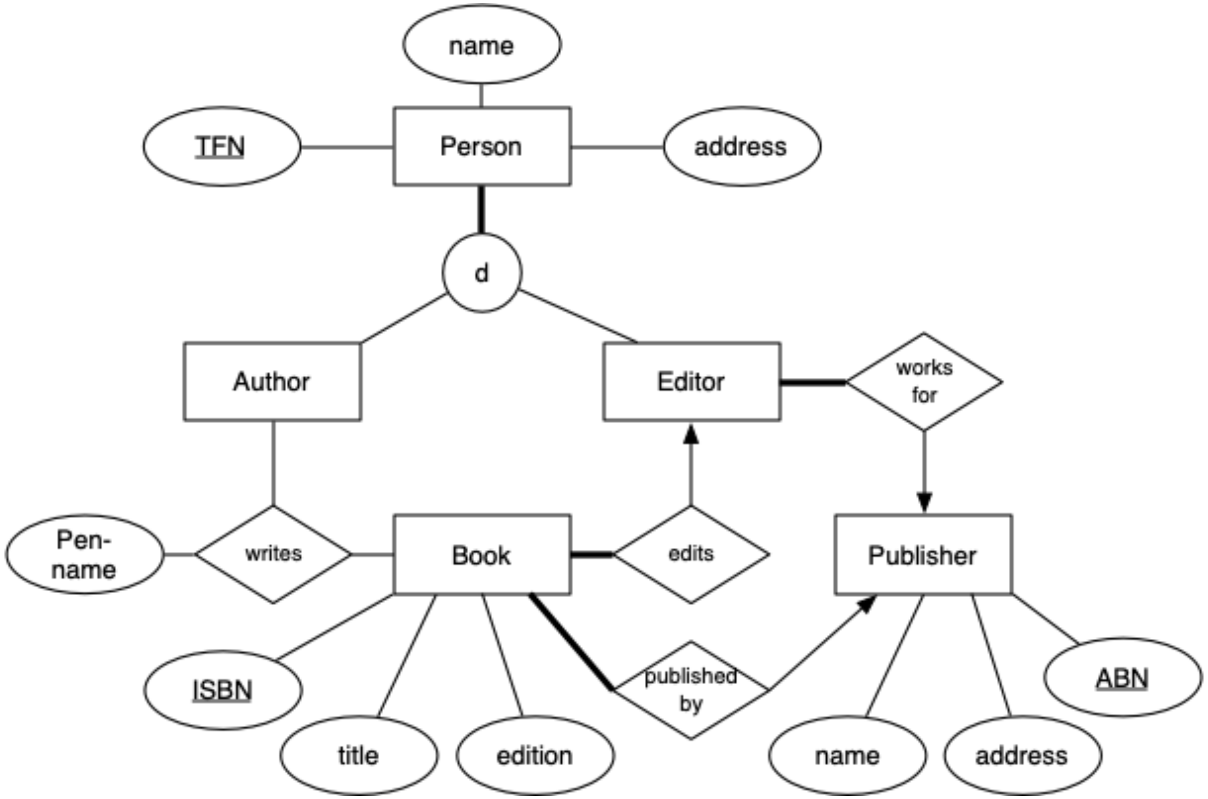
❖ Exercise: ER-to-SQL (2)

Convert this ER design to SQL:



❖ Exercise: ER-to-SQL (3)

Convert the Publishing ER model to SQL



- A TFN is stored as a 9-digit number
- An ABN is stored as an 11-digit number

- An ISBN (13-digit version) looks like 978-3-16-148410-0

<< ^ >>

❖ PostgreSQL Databases

Create a database in PostgreSQL via

```
$ createdb  DatabaseName
```

Creates an empty database (no schema, no data)

Remove a database in PostgreSQL via

```
$ dropdb  DatabaseName
```

Removes schema and all data permanently

Remove an entire PostgreSQL server (on **vxdb2**)

```
$ rm -fr /localstorage/$USER/pgsql
```

Removes all server files, all databases, all data !

<< ^ >>

❖ psql

The **psql** command is a shell that allows you to

- conect to databases (one at a time)
- ask SQL queries on a database
- find information (meta data) about a database
- add/delete/update tuples in tables

Usage:

```
$ psql  mydb
...
mydb=# \d
...
mydb=$ select * from SomeTable;
...
mydb=# \q
```

<< ^ >>

❖ psql (cont)

A useful way to use **psql**:

```
$ psql -l
```

Gives a list of all databases under your PostgreSQL server.

The "databases" **postgres**, **template1**, **template2**

- are special databases used internally by PostgreSQL
- do not **dropdb** them

<< ∧ >>

❖ `psql` (cont)

The **`psql`** command has several prompts

- **`db=#`** ... waiting to start a command
- **`db-#`** ... waiting for rest of command
- **`db(#`** ... waiting for rest of expression **`(...)`**
- **`db'#`** ... waiting to finish a string (expecting closing `'`)

Note that **db** will be (replaced by) the name of the current database

Note that **#** means you are super-user; normal users get **>**

SQL statements can span several lines, terminated by typing **`;`**

`psql` meta-commands are single-line commands

<< ∧ >>

❖ `psql` (cont)

`psql` has a range of meta-commands, beginning with **`\`**

- **`help`** ... a quick list of useful meta-commands
- **`\?`** ... a list of all meta-commands (very many of them)
- **`\d`** ... list of all tables and views in current schema
- **`\dt`** ... list of all tables in current schema
- **`\d`** *Table* ... list of all attributes in *Table*
- **`\df`** ... list of all user-defined functions in current schema
- **`\q`** ... exit **`psql`**

<< ^ >>

❖ Exercise: Creating a database

On **vxdb2**, do the following:

- create a database called **xyz**
- examine its (empty) schema
- within **xyz** create a table

```
create table R (
   x integer primary key,
   y float not null,
   z text
);
```

- examine the schema again
- examine the attributes of table **R**
- how many tuples are in table **R**?
- remove the database **xyz**

<< ∧ >>

❖ Populating a Database

Basic way of adding tuples to a database:

```
db=# insert into Table values (val1,val2,...);
```

Adds a tuple to table ***Table*** assuming

- values satisfy all constraints on tuples/table

Ways it can fail ...

- value for primary key field is already in the table
- tuple has **null** values for fields defined as **not null**
- value for some field violates constraints on that field
- etc. etc. etc.

<< ^ >>

❖ Exercise: Inserting Tuples

Which **insert** statements are successful?
If successful, what tuple value is inserted?

```
create type Mood as enum ('sad','happy');

create table People (
        name text not null,
        feels Mood
);

insert into People values ('John','happy');
insert into People values ('Andrew','angry');
insert into People values ('Tina',null);
insert into People(name) values ('Anne');
insert into People(feels) values ('happy');
```

<< ∧ >>

❖ Exercise: More Inserting Tuples

Which **`insert`** statements are successful?
If successful, what tuple value is inserted?

```
create domain PosInt as integer check (value > 0);

create table Points (
        x PosInt default 1,
        y posint
);

insert into Points values (3,4);
insert into Points values (3,null);
insert into Points values (-3,4);
insert into Points(y) values (5);
insert into Points values (default,5);
```

<< ∧ >>

❖ Bulk Insertion

Entering tuples interactively one-by-one is not feasible

Alternative: put **`insert`** statements in a file and run

```
$ psql DatabaseName -f FileName
```

Attempts to execute each **`insert`** statement:

- if tuple valid, inserted into database
- if tuple not valid, prints error message, then continues

Note that ***`FileName`*** can contain any SQL statements

Might consist only of **`create table`** statements to build a schema

<< ^ >>

❖ Bulk Insertion (cont)

A common way of building a database

```
$ createdb mydb
$ psql mydb -f schema.sql
$ psql mydb -f data.sql
```

where

- **`schema.sql`** contains table and type definitions
- **`data.sql`** contains statements to insert tuples

<< ∧ >>

❖ Bulk Insertion (cont)

How I "debug" a database schema

```
$ dropdb mydb
$ createdb mydb
$ psql mydb -f schema.sql > .errs 2>&1
$ vi .errs
  # fix any errors that appear in .errs
$ psql mydb -f data.sql > .errs 2>&1
$ vi .errs
  # fix any errors that appear in .errs
```

Repeat until **`.errs`** contains no lines with **`ERROR`**

<< ^ >>

❖ Bulk Insertion (cont)

Alternative way of inserting tuples

```
copy TableName ( attribute names ) from stdin;
... lines containing tab-separated values ...
... one value for each of the named attributes ...
\.
```

Difference between **copy** and multiple **insert**s

- with **insert** ...
  - all tuples with valid values are inserted
  - tuples with invalid values are not inserted
- with **copy** ...
  - if any tuple contains invlid values, nothing is inserted

<< ∧ >>

❖ Dump/Restore

Once a database is built, can make a complete copy in a text file

- the whole schema (including types, constraints, etc), and all data

by running the command

```
$ pg_dump -O -x DatabaseName > DumpFileName
```

and can make a new copy via

```
$ createdb newdb
$ psql newdb -f DumpFileName
```

We generally supply databases using pre-built dump files

<< ^

❖ Exercise: Playing with Beer Database

Load up the database from **`ass1.dump`**

Guess some SQL to answer the following:

- what tables are in the database
- what fields/attributes are in the **`Beers`** table
- what fields/attributes are in the **`Breweries`** table
- how many beers there are
- how many breweries there are
- what beers have "Black Lung" in their name

Produced: 20 Sep 2023