E-R Models to Relation Schemas

Thomas Heinis

t.heinis@imperial.ac.uk
Scale Lab - scale.doc.ic.ac.uk

SCALE LAB

Imperial College London

Relations & Normalization

Open Orders				
Producer No	**Producer City**	**Product No**	**Price**	**Quantity**
3	London	52	65	4
22	Leeds	10	15	5
22	Leeds	12	4	11
3	London	44	43	32
3	London	43	3	27

Producers	
Producer No	**Producer City**
3	London
22	Leeds

Open Orders			
Producer No	**Product No**	**Price**	**Quantity**
3	52	65	4
22	10	15	5
22	12	4	11
3	44	43	32
3	43	3	27

Joins & Keys

Keys are used to join information when querying:

- Primary key: unique identifier of tuple/row
- Foreign key: primary key used in a different table
- E.g.: primary key in *producers*: producer no; foreign key in *open orders*: producer no

Producers

Producer No	Producer City
3	London
22	Leeds

Open Orders

Producer No	Product No	Price	Quantity
3	52	65	4
22	10	15	5
22	12	4	11
3	44	43	32
3	43	3	27

Open Orders

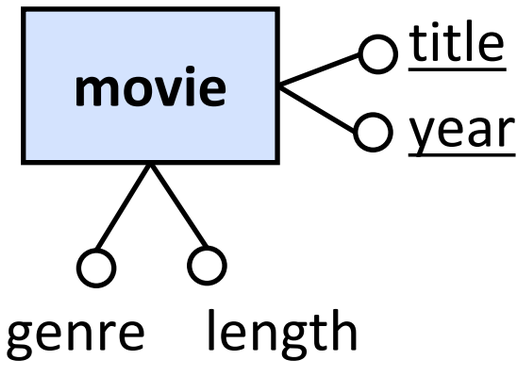
Producer No	Producer City	Product No	Price	Quantity
3	London	52	65	4
22	Leeds	10	15	5
22	Leeds	12	4	11

Mapping Data Models to Relations

- Once we have an Entity Relationship Model (or any high-level model), the next step is to map the E-R model into relational schemas.
- The schemas can then be further refined/enriched using functional dependencies and normalisation and implemented using a relational database language (e.g. SQL) and management system (e.g. PostgreSQL).
- A high-level data model isn't the only concern for a database designer. In fact, it's probably one of the simpler aspects. There a variety or other things that a database designer needs to consider. These include *application constraints* (e.g checks and assertions), *performance requirements* (e.g. query response times), *access control* (who or what is permitted to access or modify which data and under what circumstances), *continuity procedures* for when disaster strikes, how to anticipate and handle updates and changes in requirements, etc.

Strong Entity Sets with Simple Attributes

A strong entity set with simple attributes can be mapped directly to a relation with the same attributes. Each tuple in the relation would correspond to an entity in the entity set. The primary key of the entity set becomes the primary key of the relation.

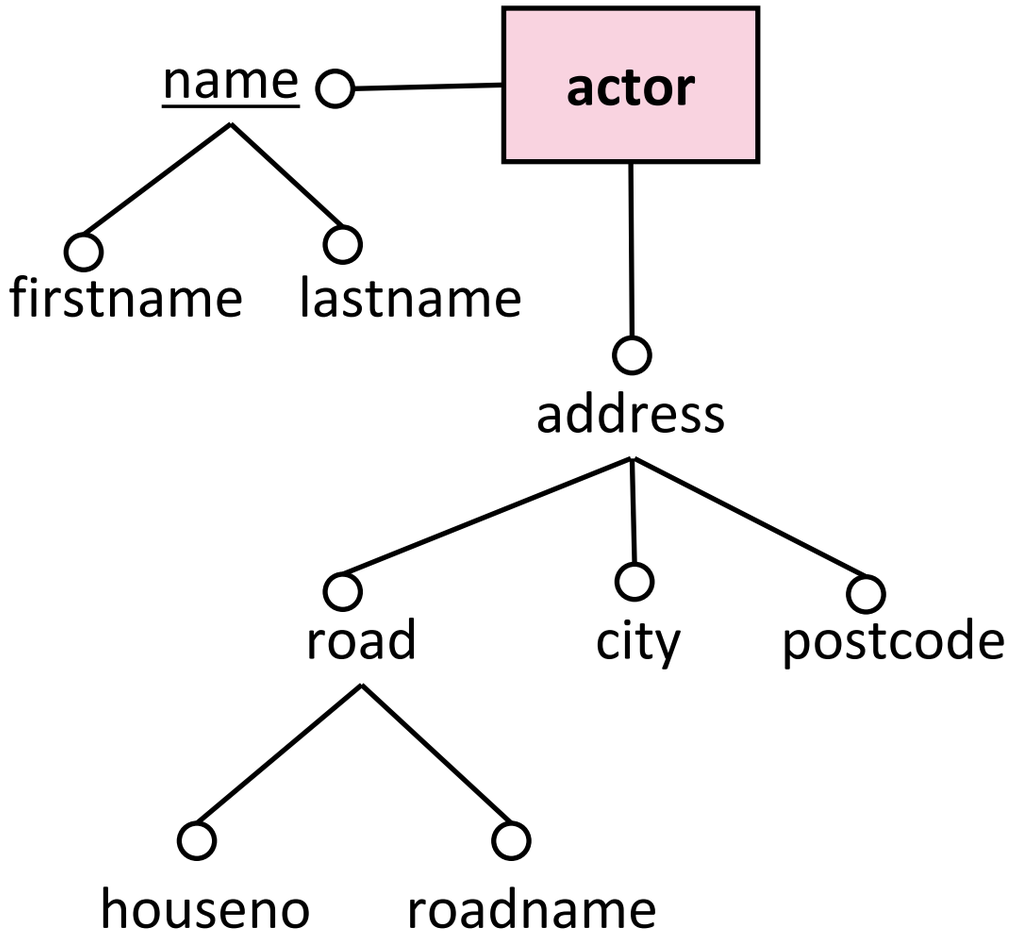


movie(<u>title</u>, <u>year</u>, length, genre)

```
create table movie (
        title       varchar(120),
        year        int,
        length      int,
        genre       char(20),

        primary key (title, year)
)
```

Composite Attributes

If an entity set has a composite attribute, we ‘flatten’ the attribute. That is, the mapped relation includes only the simple attributes within the composite attribute.

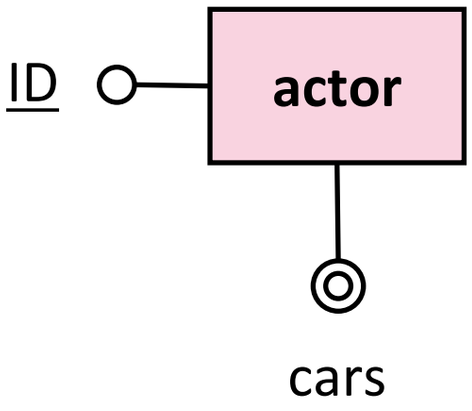


actor(<u>firstname</u>, <u>lastname</u>, houseno, roadname, city, postcode)

```
create table actor (
        firstname       varchar(30),
        lastname        varchar(30),
        houseno         int,
        roadname        varchar(30),
        city            varchar(40),
        postcode        varchar(10)

        primary key (firstname, lastname)

)
```

Multivalued Attributes

Multivalued attributes are mapped to their own relation and linked back to the entity set relation using a foreign key constraint:



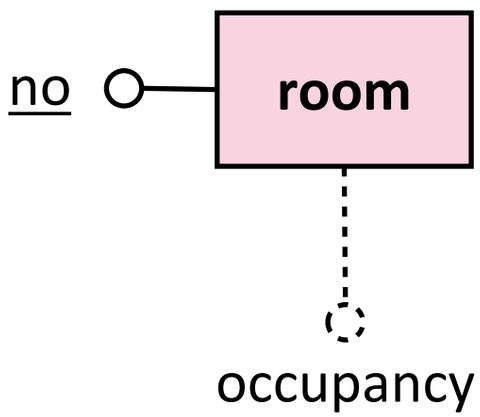
actor(<u>ID</u>, *otherattributes*)

actor_cars(<u>actorID</u>, <u>carID</u>, *otherattributes*)

```
create table actor_cars (
        actorID    int,
        carID      varchar(10),

        primary key (actorID, carID),
        foreign key (actorID) references actor.ID
)
```

Derived Attributes

The relational model doesn't support derived attributes like age from date of birth. Rather queries are expected to compute the derived attribute value when required using an expression (or expression in a view). We could map a derived attribute to a simple attribute, but would need to update it as required, e.g. when inserts, deletes, updates or external events (e.g. new year) affect the derived value.

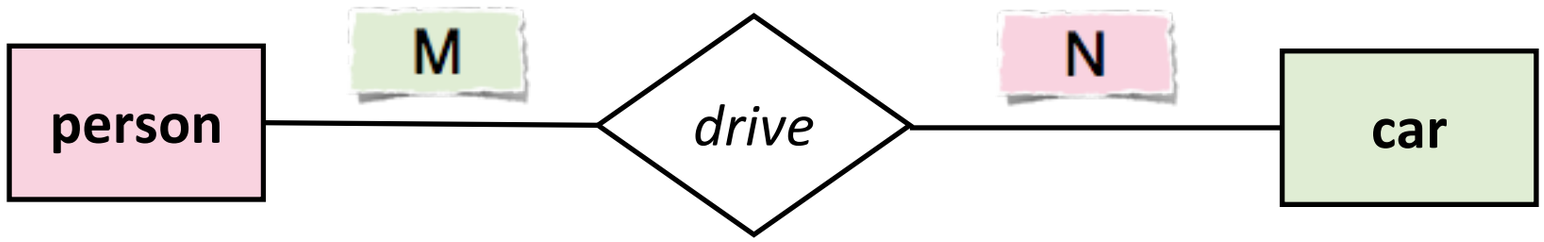


staff(<u>name</u>, roomno, *otherattributes*)
room(<u>no</u>, occupancy, *otherattributes*)

Many RDBMSs allow the programmer to define functions, which can be used in expressions and used to compute derived attributes. This improves reusability of code and reduces errors.

Many-to-Many Relationship Sets

We map a Many-to-Many (binary) Relationship set into a relation with two foreign keys, corresponding to the primary keys of each entity set involved in the relationship. The primary keys of the entity sets form the primary key of the relationship set relation.



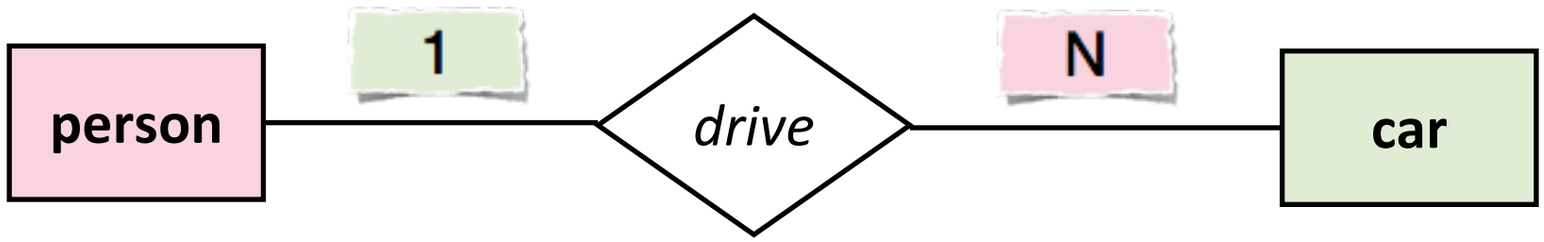
person(ID, *otherattributes*)
car(regno, *otherattributes*)
drive(personID, regno,*otherattributes*)

If the relationship set has its own attributes, they're included here, e.g isPolicyholder

```
create table drive (
        personID  varchar(10),
        regno       varchar(12),
        primary key (personID, regno),
        foreign key (personID) references person.ID on delete cascade,
        foreign key (regno) references car.regno on delete cascade

)
```

One-to-Many Relationship Sets

A One-to-Many Relationship set can be mapped like a Many-to-Many Relationship. However, it's simpler to directly include the primary key of the One relation as a foreign key attribute in the Many relation:



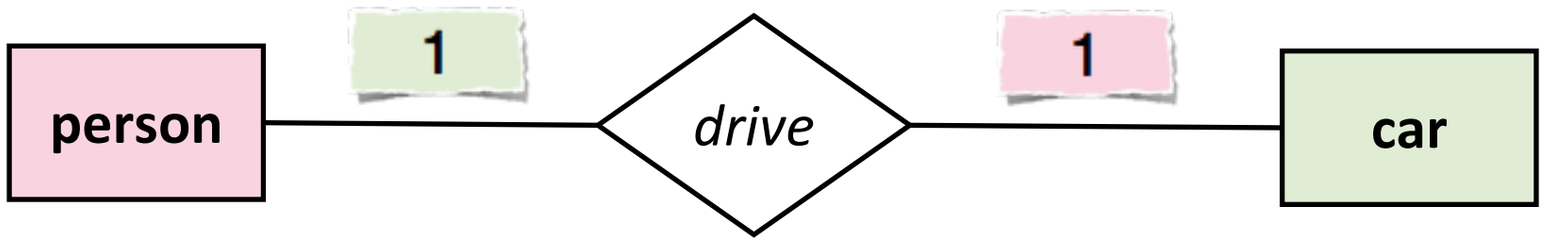
person(<u>ID</u>, *otherattributes*)
car(<u>regno</u>, personID, *otherattributes*)

```
create table car (
        regno      varchar(12),
        personID varchar(10),
        primary key (regno),
        foreign key (personID) references person.ID
)
```

If the relationship set has its own attributes, they're included in the Many relation.

One-to-One Relationship Sets

Like a One-to-Many Relationship we directly include the primary key of one of the One relations as a foreign key attribute in the other One relation. The choice is left to the database designer:



person(<u>ID</u>, <u>regno</u>, *otherattributes*)
car(<u>regno</u>, *otherattributes*)

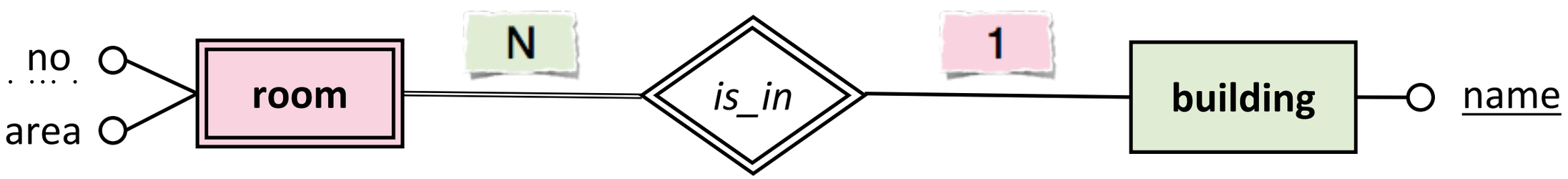
```
create table person (
        personID  varchar(12),
        regno     varchar(10),
        primary key (personID),
        foreign key (regno) references car.regno
)
```

Note: We can use the mapping on the previous slide also

If the relationship set has its own attributes, they're included in the relation with the foreign key.

Weak Entity Sets

Like strong entity sets, a weak entity set is mapped to its own relation and attributes but includes the primary key of the strong entity set as a foreign key with an on delete cascade constraint. The relationship isn't mapped.



building(<u>name</u>, *otherattributes*)
room(<u>no</u>, <u>buildingname</u>, *otherattributes*)

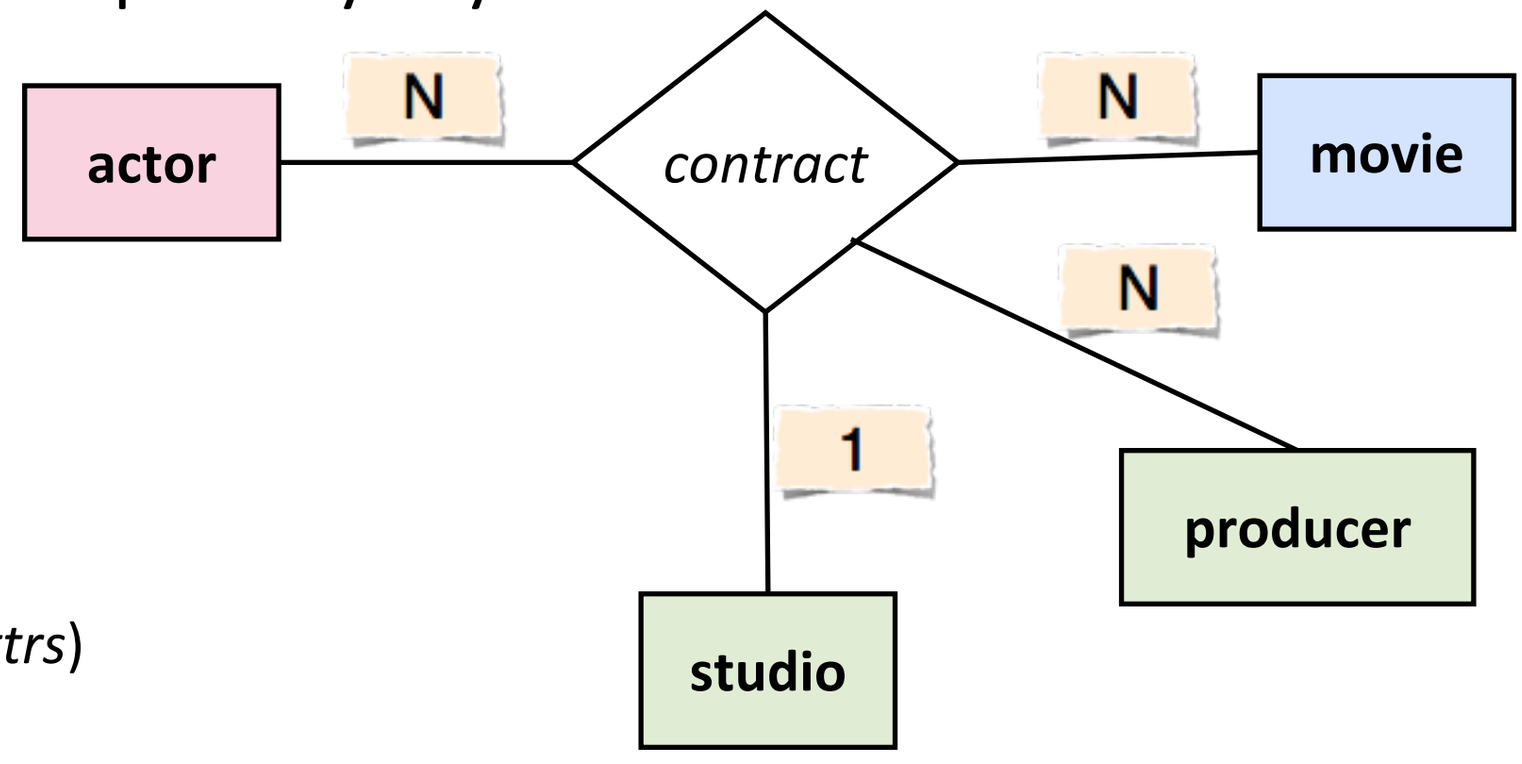
```
create table room (
        no    varchar(120),
        buildingname varchar(50) not null,

        primary key (no, buildingname),
        foreign key (buildingname) references building.name on delete cascade
)
```

Ensures total participation.

Note: The primary key for the Weak Entity set relation consists of the primary key of the strong entity set relation and one or more discriminating attributes from the weak entity set

Multiway Relationship Sets

We can map an n-way relationship set into several binary relationships or generalise the many-to-many relationship mapping to include primary keys from all the entity sets as foreign keys. The foreign keys of the Many entity sets form the primary key.

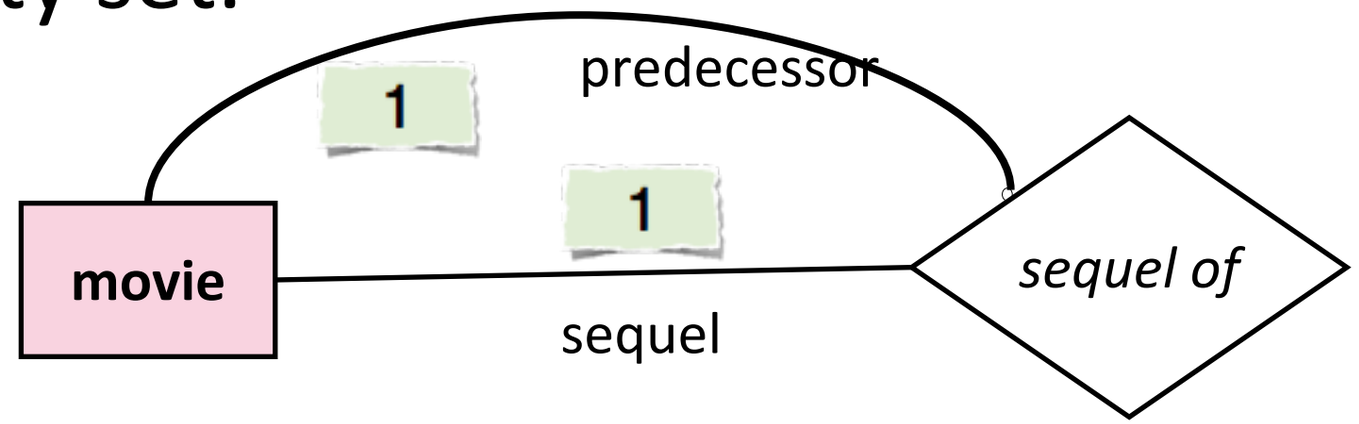


actor(<u>ID</u>, *otherattributes*)
movie(<u>ID</u>, *otherattributes*)
studio(<u>ID</u>, *otherattributes*)
producer(<u>ID</u>, *otherattributes*)
contract(<u>aID</u>, <u>mID</u>, sID, <u>pID</u>,*otherattrs*)

create table contract (
    actorID **int**, movieID **int**, studioID **int**, producerID **int**,
    primary key (actorID, movieID, producerID),
    plus foreign key declarations for actorID, movieID, studioID, producerID
)

Roles in relationships

For roles, we map each role into a foreign key attribute on the entity set.



movie(<u>ID</u>, *otherattributes*)
sequelof(<u>originalID</u>, <u>sequelID</u>, *otherattributes*)

```
create table sequelof (
        originalID  int,
        sequelID  int,
        primary key (originalID, sequelID),
        foreign key originalID references movie.ID,
        foreign key sequelID references movie.ID
)
```

Database Design Maxims

Here are three general maxims to consider when designing a database:

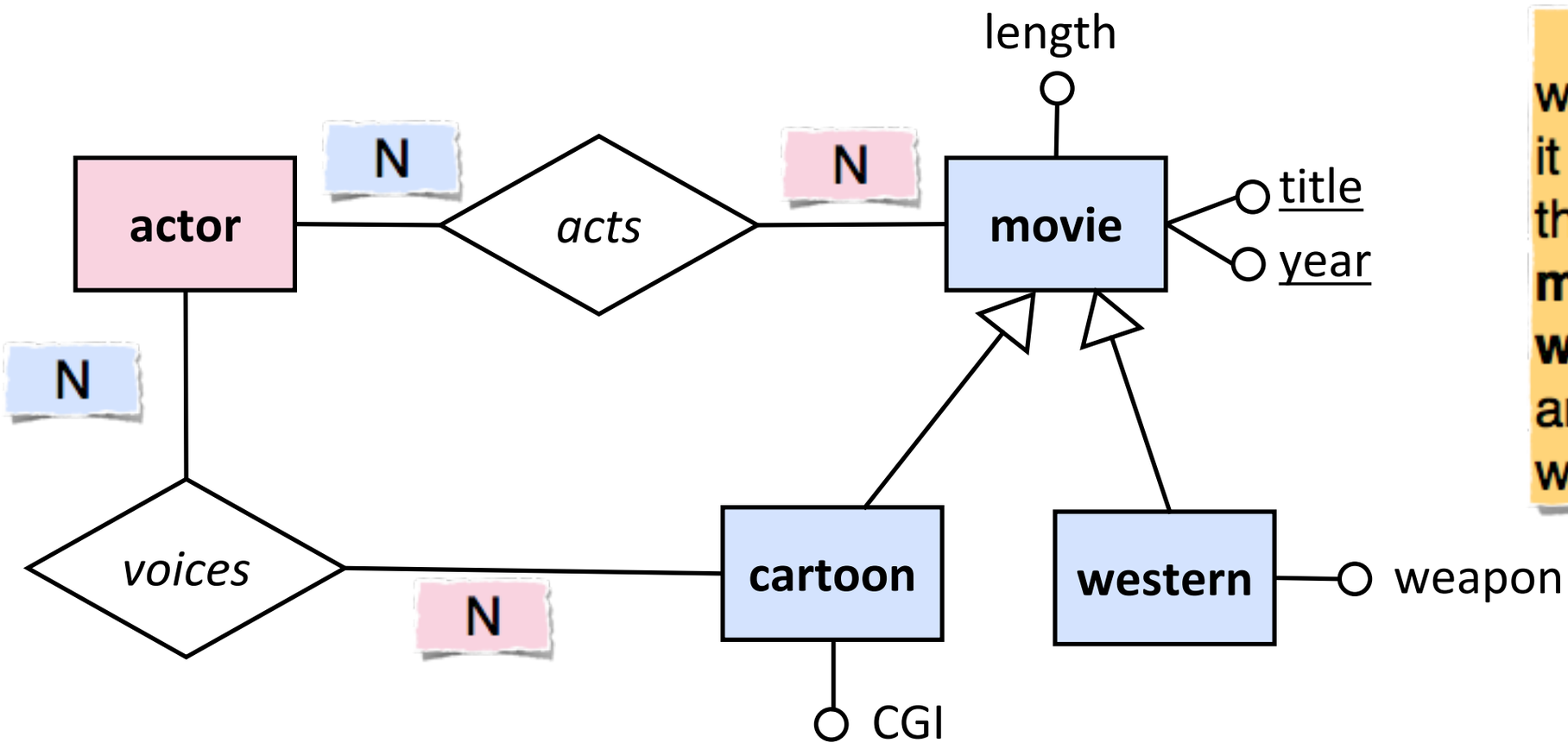
Model the "Real World"	Work closely with the client and try to reflect what their requirements are, study their workflows, and the external rules and practices that govern them & their organisation. *Ask lots of questions*.
Keep it simple stupid (KISS)	Don't get carried away with lots of entity and relationship sets. For example, if an entity has only one property and its only related to one entity, then it may better to just add the attribute to the other entity. Similarly, consider whether a particular relationship serves a useful purpose or whether it be could changed into attributes. However, if a simplification obscures the real-world, then avoid the simplification and follow Albert Einstein's advice "*Everything should be made as simple as possible, but no simpler*" .
Express it once only	

Extended E-R Models

- The E-R model that we've considered covers the classical features of E-R modelling.
- Newer E-R models have extended the classical E-R model with support for various concepts taken from *object-oriented design*. Such designs can be mapped to a relational database, or, if available, to an **Object-Relational Database Management System** (ORDBMS) that directly supports one or more of the concepts.
- To conclude our study of Databases we'll look at how E-R Models support specialisation/generalisation using **is-a hierarchies**.
- Specialisation and generalisation are the processes of identifying a containment relationship between a entity set (the higher-level entity set) and one or more lower-level entity sets. **Specialisation** proceeds top-down (high-level to low-level) emphasising entity subsets and attribute differences. **Generalisation** proceeds bottom up emphasising similarities and common attributes. The result is normally the same. c.f superclasses and subclasses in OO programming.

is-a Relationships

Specialisation/generalisation is represented in an ERD using *is-a* relationships. We'll represent *is-a* using lines with a hollow arrow-head:

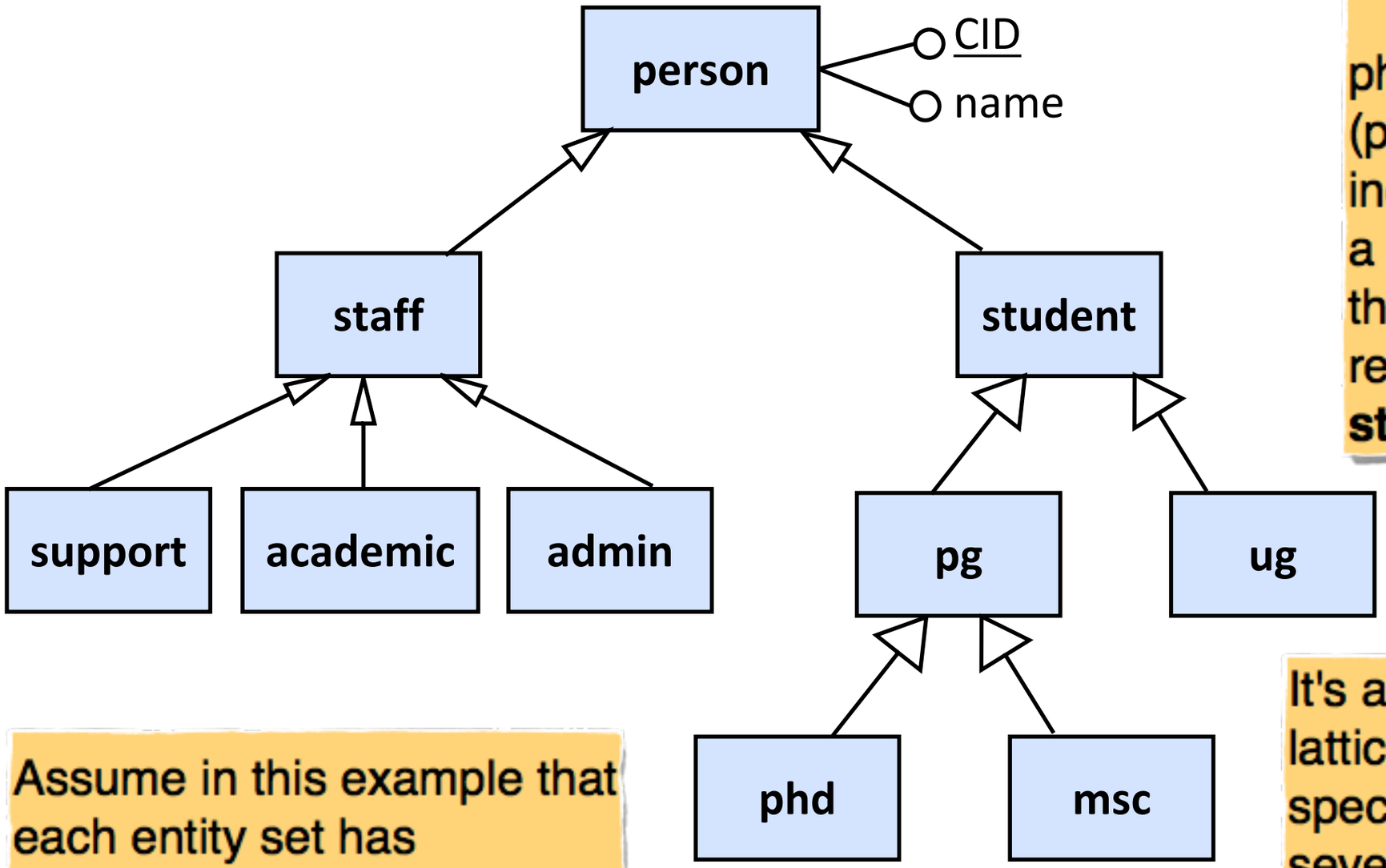


western *is-a* **movie**, it has (*inherits*) all the attributes of **movie** plus **weapon**, it also has an *acts* relationship with **actor**.

cartoon *is-a* **movie**, it has all the attributes of movie. It also has an *acts* relationship with **actors** and a *voices* relationship with **actor**.

is-a Hierarchies

We can specialise/generalise at multiple levels, forming an *is-a* hierarchy:



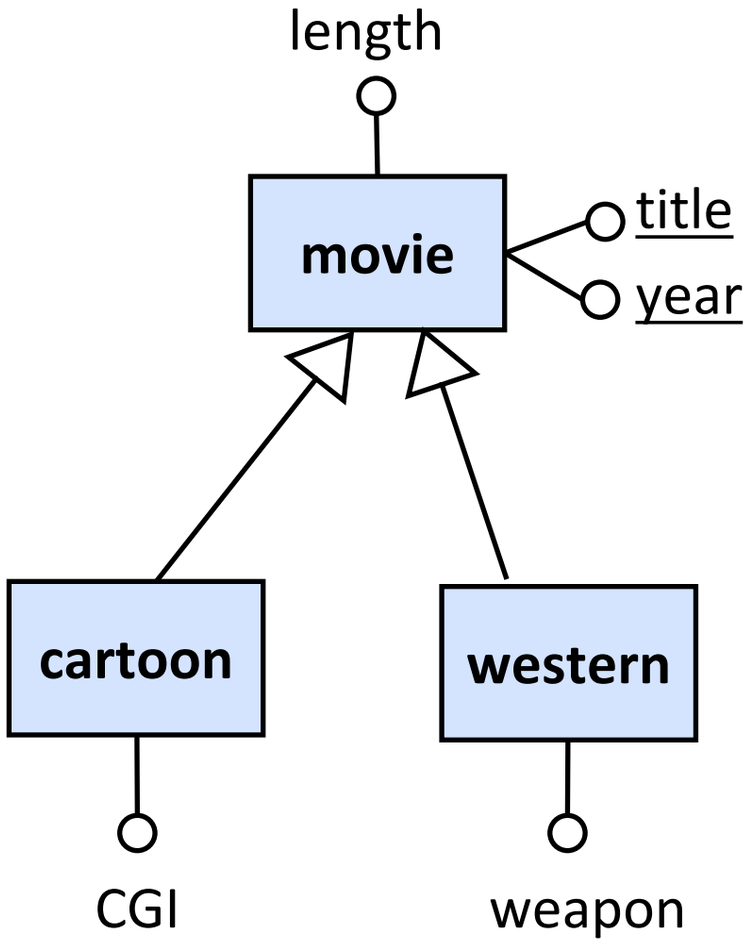
phd *is-a* pg (postgraduate) and indirectly a **student** and a **person**. It inherits all the attributes and relationships of **pg**, and **student** and **person**.

Assume in this example that each entity set has attributes and relationships.

It's also possible to have *is-a* lattices, where entity sets specialise and inherit from several higher-level entity sets.

Mapping is-a Hierarchies to Relations 1

There are 3 approaches we can take to map *is-a* hierarchies to relation schemas. The first approach relates each lower-level entity set with its root level entity-set by including the primary key of the root level entity set with the lower-level entity-set:



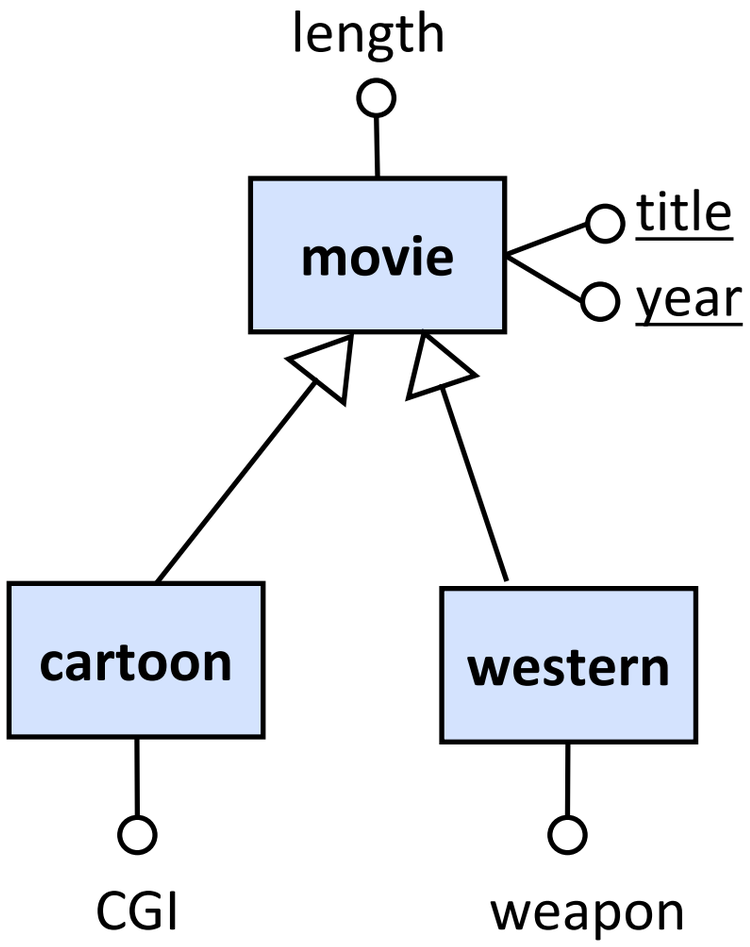
movie(<u>title</u>, <u>year</u>, length)
western(<u>title</u>, <u>year</u>, weapon)
cartoon(<u>title</u>, <u>year</u>, CGI)

We also need to create foreign-key constraints in **western** and **cartoon** for **title** and **year**.

Note: the data for a **western** is held in two tuples, one for **cartoon** and one for **movie**. We have to use joins on the primary key to recover inherited attributes, e.g. **length** of cartoons.

Mapping is-a Hierarchies to Relations 2

In the second approach, the relation for each lower-level entity set includes all the attributes of higher-level entities:

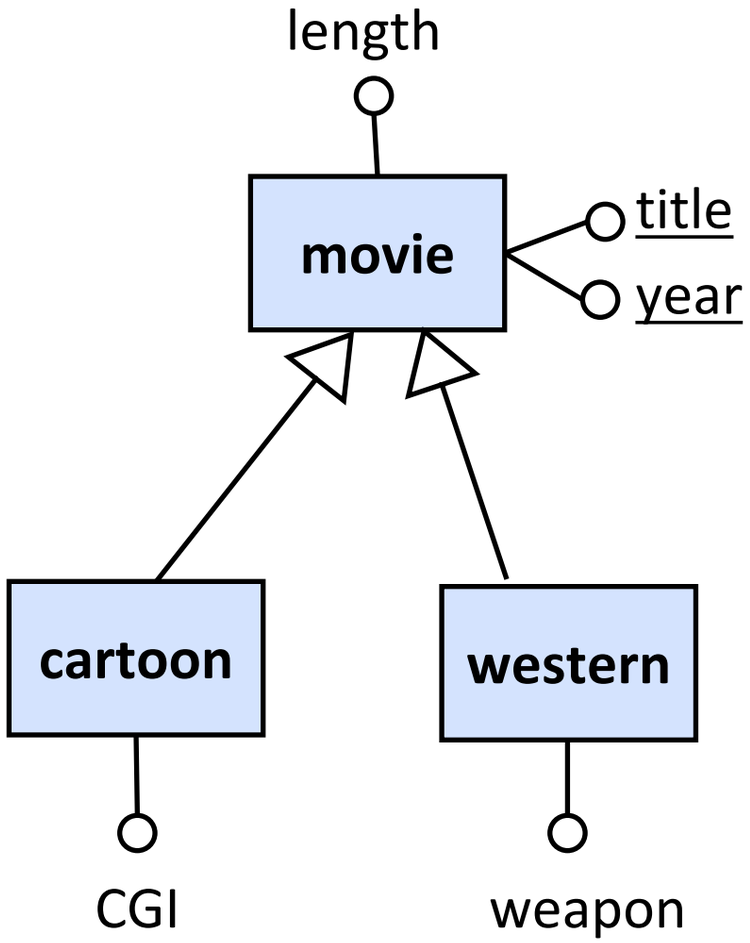


movie(<u>title</u>, <u>year</u>, length)
western(<u>title</u>, <u>year</u>, length, weapon)
cartoon(<u>title</u>, <u>year</u>, length, CGI)
cartoonWestern(<u>title</u>, <u>year</u>, length, CGI, weapon)

If we need to find information on all movies, for example, all movies longer than 2 hours, we'd need to do a query that performs the union of queries on 4 relations.

Mapping is-a Hierarchies to Relations 3

In the third approach, we use a single relation for the entire *is-a* hierarchy and include all attributes of all entity-sets in the relation.



movie(<u>title</u>, <u>year</u>, length, weapon, CGI)

We use **null** to discriminate entities. For example, movies that aren't cartoons would have **CGI** set to **null**. Movies that aren't westerns nor cartoons, have **weapon** and **weapon** set to **null**.

The **null** approach produces big tuples and requires carefully formulated queries that select appropriate lower-level entities.

Translation Example