RELATIONAL DATABASES
4. High-level database models

Carmelo Vaccaro

University of Paris-Saclay

Master 1 - AI
2022/23: first semester

The content of these slides is taken from the book
Database Systems - The Complete Book,
by Hector Garcia-Molina, Jeffrey D. Ullman and Jennifer Widom,
published by Pearson, 2014.

High-level database models 1/2

When we design a database it is often easier to start with a higher-level model and then convert the design into a relational model.

The reason for doing so is that the relational model has only one concept — the relation — rather than several complementary concepts that more closely model real-world situations.

High-level database models 2/2

Databases management systems of relational type are the most used because the simplicity of concepts in the relational model makes implementation of database operations very efficient.

However when designing a database it is better to start with a more complex model, like the "entity-relationship model", and then convert the obtained schema into a relational schema.

Contents of the chapter

1. The entity/relationship model

The entity/relationship model: element types

In the *entity-relationship model* (or *E/R model*) the structure of data is represented graphically, as an "entity-relationship diagram," using three principal element types:

1. entity sets,
2. attributes,
3. relationships.

Entity sets

An *entity* is an abstract object of some sort, and a collection of similar entities forms an entity set.

An entity in some ways resembles an "object" in the sense of object-oriented programming. But since the E/R model is a static concept, involving the structure of data and not the operations on data, there are no methods associated with an entity set as one would with a class.

Therefore the entity set resembles more the "struct" in the C language.

Entity sets: example

In the movie-database example each movie is an entity, and the set of all movies constitutes an entity set.

Likewise, the stars are entities, and the set of stars is an entity set.

A studio is another kind of entity, and the set of studios is a third entity set.

Attributes 1/2

Entity sets have associated *attributes*, which are properties of the entities in that set.

For instance, the entity set *Movies* might be given attributes such as *title* and *length*. Thus the attributes for the entity set *Movies* resemble the attributes of the relation *Movies* in our example.

Attributes 2/2

It is common for entity sets to be implemented as relations, although not every relation in our final relational design will come from an entity set.

We shall assume that attributes are of primitive types, such as strings, integers, or reals. There are other variations of the E/R model where attributes can have some limited structure.

Relationships

Relationships are connections among two or more entity sets.

For instance, if *Movies* and *Stars* are two entity sets, we could have a relationship *Stars-in* that connects movies and stars.

The intent is that a movie entity m is related to a star entity s by the relationship *Stars-in* if s appears in movie m.

Binary relationships, those between two entity sets, are by far the most common type of relationship. However the E/R model allows relationships to involve any number of entity sets.

Entity-relationship diagrams

An *E/R diagram* is a graph representing entity sets, attributes, and relationships.

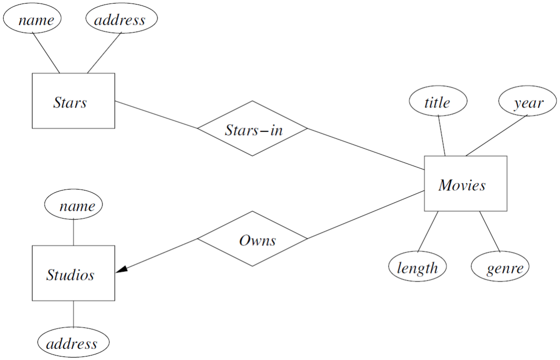
Elements of each of these kinds are represented by nodes of the graph having the following shapes:

- entity sets are represented by rectangles,
- attributes are represented by ovals,
- relationships are represented by diamonds.

Edges connect an entity set to its attributes and also connect a relationship to its entity sets.

Entity-relationship diagrams: example

The following is an E/R diagram that represents a simple database about movies. The entity sets are *Movies*, *Stars*, and *Studios*; the relationships are *Stars-in* and *Owns*.



Instances of an E/R diagram

E/R diagrams are in general used only for modeling purposes and real data are used to populate only the relational tables.

However, it is often useful to visualize the data in the entity/relationship form. While entities resemble tables in the relational model, relationships are really a concept specific to the E/R model. We can imagine the tuples of an entity set as made by values corresponding to attributes.

On the other hand the "tuples" of a relationship set are not really tuples of a relation, since their components are entities rather than primitive types such as strings or integers. The columns of the table are headed by the names of the entity sets involved in the relationship, and each list of connected entities occupies one row of the table.

Instances of an E/R diagram: example

An instance of the *Stars-in* relationship could be visualized as a table with pairs such as:

Movies	*Stars*
Basic Instinct	Sharon Stone
Total Recall	Arnold Schwarzenegger
Total Recall	Sharon Stone

Multiplicity of binary E/R relationships

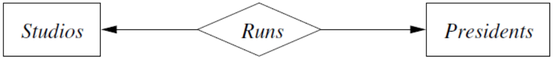
Suppose R is a relationship connecting entity sets E and F. Then:

- If each member of E can be connected by R to at most one member of F, then we say that R is *many-one* from E to F or *one-many* from F to E.
- If an entity of either entity set can be connected to at most one entity of the other set then we say that R is *one-one*. A one-one relationship between E and F is both many-one and one-many from E to F.
- If R is neither many-one nor one-many from E to F, then we say that R is *many-many*.

Notation for multiplicities

If a relationship from an entity set E to an entity set F is (many, one)-one, then it is denoted with an arrow pointing to the set F.

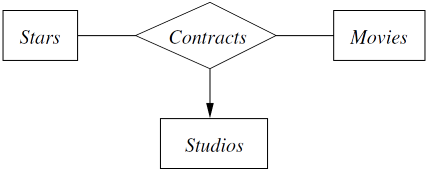
For example in the following figure the relationship between *Studios* and *Presidents* is one-one.



Multiway relationships 1/2

In the E/R model it is possible to define relationships involving more than two entity sets. These relationships, called *multiway*, are represented by lines from the relationship diamond to each of the involved entity sets.

The following is a relationship *Contracts* that involves a studio, a star, and a movie. This relationship represents that a studio has contracted with a particular star to act in a particular movie.



Multiway relationships 2/2

In multiway relationships, an arrow pointing to an entity set E means that if we select one entity from each of the other entity sets in the relationship, those entities are related to at most one entity in E.

In the figure of the previous slide we have an arrow pointing to entity set *Studios*, indicating that for a particular star and movie, there is only one studio with which the star has contracted for that movie.

Roles in relationships

It is possible that one entity set appears two or more times in a single relationship.

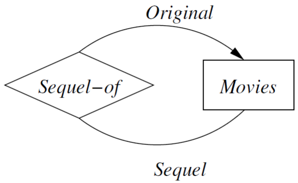
If so, we draw as many lines from the relationship to the entity set as the entity set appears in the relationship. Each line to the entity set represents a different role that the entity set plays in the relationship.

We therefore label the edges between the entity set and relationship by names, which we call “roles.”

Roles in relationships: example

Below is a relationship *Sequel-of* between the entity set *Movies* and itself. Each relationship is between two movies, one of which is the sequel of the other. To differentiate the two movies in a relationship, one line is labeled by the role *Original* and one by the role *Sequel*.

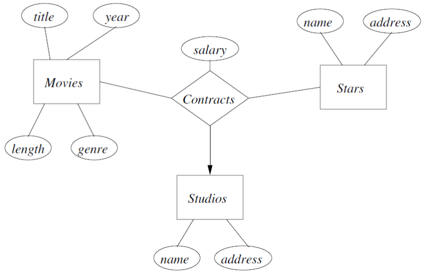
We assume that a movie may have many sequels, but for each sequel there is only one original movie. Thus, the relationship is many-one from *Sequel* movies to *Original* movies, as indicated by the arrow.



Attributes on relationships

We may place one or more attributes on any relationship. These attributes are functionally determined by the entire tuple in the relationship set for that relation.

Below we see that the relationship *Contracts* has attribute salary.



Subclasses in the E/R model

An entity set A is a *subclass* of another entity set B if A has all attributes and relationships of B but it can have extra attributes and relationships with respect to B.

In this case we say that A is in an *isa* relationship with B (A *is a* B). Every isa relationship is one-one.

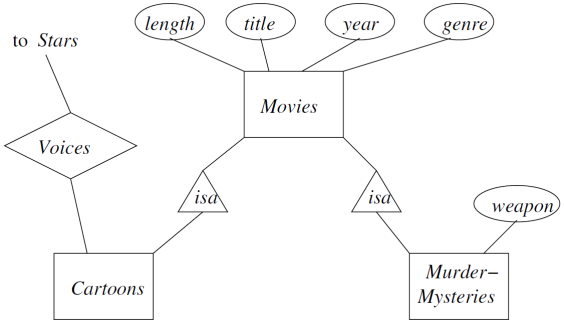
Isa relationships are denoted with a triangle: the bottom of the triangle is connected to the subclass and the top to the superclass.

Subclasses in the E/R model: example 1/2

Among the special kinds of movies we might store in our example database are cartoons and murder mysteries. For each of these special movie types, we could define a subclass of the entity set *Movies*. For instance, let us postulate two subclasses: *Cartoons* and *Murder-Mysteries*.

A cartoon has, in addition to the attributes and relationships of *Movies*, an additional relationship called *Voices* that gives us a set of stars who speak, but do not appear in the movie. Movies that are not cartoons do not have such stars. *Murder-mysteries* have an additional attribute *weapon*.

Subclasses in the E/R model: example 2/2



Tree structures of isa relationships

An entity of entity sets involved in an isa relationship belongs not to a single entity set but to a subtree containing the root of the isa structure of the entity sets involved in an isa relationship.

Tree structures of isa relationships: example 1/2

A movie which is neither a cartoon nor a murder-mystery will have a component only in the entity set *Movies* and thus only the attributes and relationships of *Movies*.

A cartoon that is not a murder-mystery will have two components, one in *Movies* and one in *Cartoons*. It will have the attributes and relationships of *Movies* and *Cartoons*.

Tree structures of isa relationships: example 2/2

A murder-mystery that is not a cartoon will have two components, one in *Movies* and one in *Murder-Mysteries* and thus will have attributes and relationships of *Movies* and *Murder-Mysteries*.

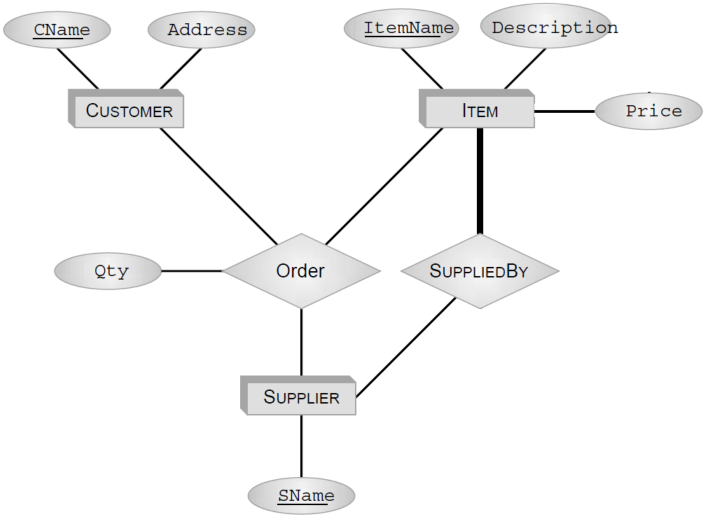
Finally, a movie like *Roger Rabbit*, which is both a cartoon and a murder-mystery, will have components in all three of the entity sets *Movies*, *Cartoons*, and *Murder-Mysteries*. It will have attributes and relationships of *Movies*, *Cartoons*, and *Murder-Mysteries*.

Exercise

Draw the E/R diagram for a database about orders made by customers and constituted by items. An order specifies the quantity of an item and the supplier of the item. A customer has a name and an address. An item has a name, a description and a price and is supplied by a supplier. The supplier has a name.

Specify the keys.

Solution


CName
Address
ItemName
Description
Customer
Item
Price
Qty
Order
SuppliedBy
Supplier
SName

2. Constraints in the E/R model

Constraints in the E/R model

Like the relational model, the E/R model has ways to express the constraint of key and referential integrity. The E/R model has also ways to express degree constraints by limiting the possible multiplicities.

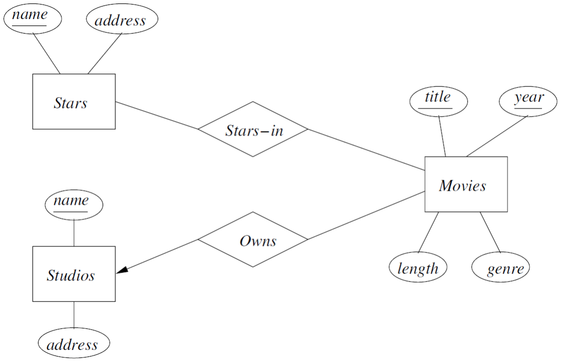
Keys in the E/R model

The concept of key for the E/R model is the same as for the relational model: a *key* for an entity set E is a set K of one or more attributes such that, given any two distinct entities e_1 and e_2 in E, e_1 and e_2 cannot have identical values for all the attributes in the key K.

If K consists of more than one attribute, then it is possible for e_1 and e_2 to agree in some of these attributes, but never in all attributes.

Notation for keys

In the E/R-diagram notation the attributes belonging to a key are underlined. For example, the figure below shows the E/R diagram for movies with key attributes underlined.



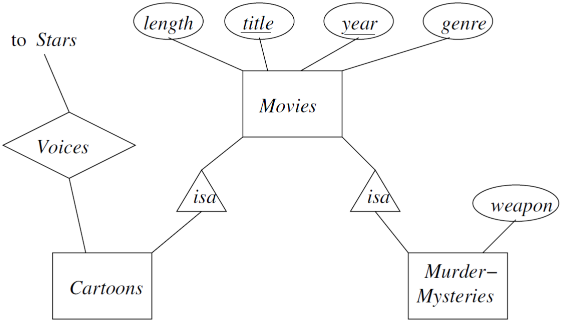
Keys in an isa-hierarchy

When an entity set is involved in an isa-hierarchy, we require that the root entity set have all the attributes needed for a key, and that the key for each entity is found from its component in the root entity set.

For example, movies in the *Cartoon* or *Murder-Mysteries* entity sets have as key the attributes *title* and *year* of the *Movie* entity set, the root of the isa hierarchy.

Example

Cartoons, *Murder-Mysteries* have as key the attributes *length* and *title*, the same key as *Movies*, from which they inherit.



Referential integrity

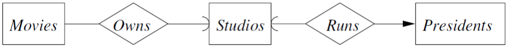
Given entity sets E and F we say that there is a *referential integrity* constraint between E and F if there is a (many, one)-one relationship between E and F and we require that the unique entity of F associated with an entity in E must be present in the table of F.

The notation for a relationship between E and F with referential integrity is a rounded arrowhead pointing to F.

Referential integrity: example

In the diagram below we see that every movie must be owned by one studio, and this studio must be present in the *Studios* entity set. Also every president runs a studio that exists in the *Studios* entity set. This is because if a studio ceases to exist, its president can no longer be called a president and would be deleted from the entity set *Presidents*.

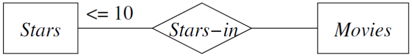
On the other hand there is no a referential integrity constraint in the relationship *Runs* from *Studios* to *Presidents* because if a president were fired or resigned, the studio would continue to exist: a studio has at most one president, but might have no president at some time.



Degree Constraints

In the E/R model, we can attach a bounding number to the edges that connect a relationship to an entity set, indicating limits on the number of entities that can be connected to any one entity of the related entity set.

For example, we could decide that a movie entity cannot be connected by relationship *Stars-in* to more than 10 star entities.



Exercise

Draw the E/R diagram for a database about a library.

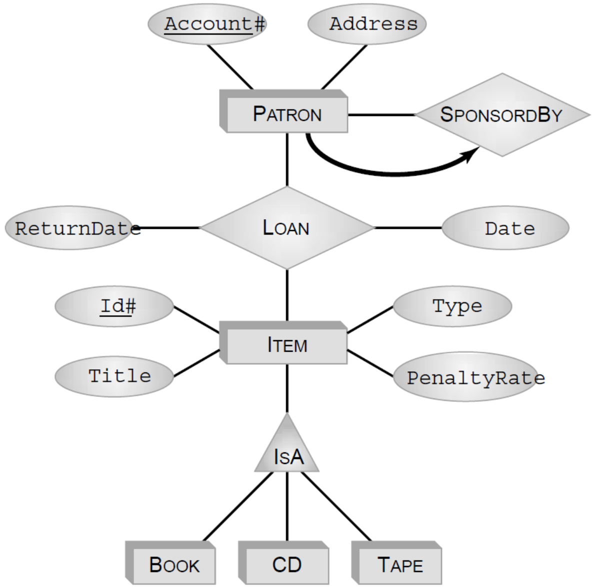
The library has three different kinds of items (books, CDs and tapes) which are lent to library patrons. A loan has a starting date and a return date.

The items have an id number, a title, a type indicating if it can be lent for short or long periods and a penalty rate.

The patrons have an account number and an address. Some patrons are minors and they must have one sponsoring patron.

Specify the keys, multiplicities and use isa relationships.

Solution


Account#
Address
PATRON
SPONSORDBY
ReturnDate
LOAN
Date
Id#
Type
ITEM
Title
PenaltyRate
ISA
BOOK
CD
TAPE

3. Weak entity sets

Weak entity sets

An entity set is called a *weak entity set* if some of the attributes composing its key belong to another entity set.

The principal reason why we need weak entity sets is that sometimes an entity set can fall into a hierarchy based on classifications unrelated to the "isa hierarchy".

For example if entities of set E are subunits of entities in set F, then it is possible that the names of E-entities are not unique until we take into account the name of the F-entity to which the E entity is subordinate.

Example 1

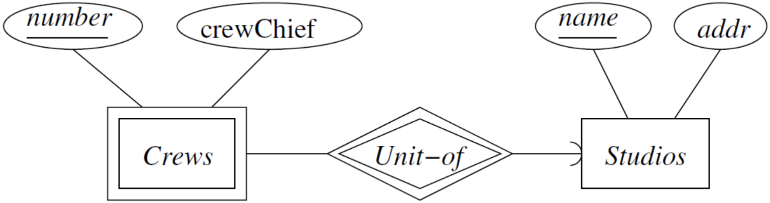
A movie studio might have several film crews. The crews might be designated by a given studio as crew 1, crew 2, and so on.

However, other studios might use the same designations for crews, so the attribute *number* is not a key for crews. Rather, to name a crew uniquely, we need to give both the name of the studio to which it belongs and the number of the crew.

Example 1

The double-rectangle indicates a weak entity set, and the double-diamond indicates a many-one relationship that helps provide the key for the weak entity set.

The key for weak entity set *Crews* is its own *number* attribute and the *name* attribute of the unique studio to which the crew is related by the many-one *Unit-of* relationship.

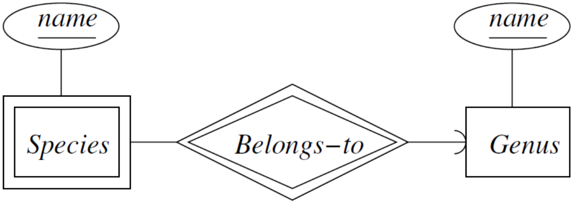


Example 2 1/2

A species is designated by its genus and species names. For example, humans are of the species *Homo sapiens*; *Homo* is the genus name and *sapiens* the species name. In general, a genus consists of several species, each of which has a name beginning with the genus name and continuing with the species name. Species names, by themselves, are not unique.

Example 2 2/2

Two or more genera may have species with the same species name. Thus, to designate a species uniquely we need both the species name and the name of the genus to which the species is related by the *Belongs-to* relationship. *Species* is a weak entity set whose key comes partially from its genus.



Supporting relationships and supporting entity sets

Let E be a weak entity set, let F be an entity set and let R be a binary many-one relation R from E to F.

We say that R is a *supporting relationship* for E and that F is a *supporting entity set* for E if R has referential integrity from E to F, that is, for every E-entity there must be exactly one F-entity related to it by R.

In that case we say also that F *supports* E or that E *is supported* by F.

Requirements for weak entity sets

If E is a weak entity set then its key consists of:

1. zero or more of its own attributes and
2. the key attributes from the supporting entity set.

Remark

Let E be a weak entity set and let F be a supporting entity set for E. If F too is a weak entity set and is supported by the entity set G then some of the key attributes of F supplied to E will be the key attributes of G.

Weak entity set notation

1. If an entity set is weak, it will be shown as a rectangle with a double border.
2. Its supporting many-one relationships will be shown as diamonds with a double border.
3. If a weak entity set supplies any attributes for its own key, then those attributes will be underlined.

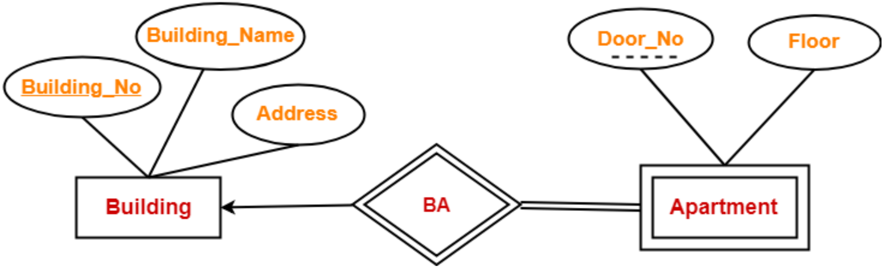
Exercise

Draw an E/R diagram for the relation describing apartments and buildings.

An apartment belongs to a building. An apartment has a door number and is located in a floor. The door number of an apartment is unique in each building, but two different apartments belonging to different building can have the same door number.

A building has a unique building number, a name and an address.

Solution


Building_Name
Building_No
Address
Building
BA
Door_No
Floor
Apartment

4. From E/R diagrams to relational databases

Main rules to convert an E/R design to a relational database schema

The main rules to convert an E/R design to a relational database schema are the following:

- turn each entity set into a relation with the same set of attributes
- replace a relationship by a relation whose attributes are the keys for the connected entity sets.

Special situations

While the previous two rules cover much of the normal situations, there are also special cases that need to be dealt differently:

- weak entity sets cannot be translated straightforwardly to relations;
- "isa" relationships and subclasses require careful treatment;
- sometimes, it is good to combine two relations, especially the relation for an entity set E and the relation that comes from a many-one relationship from E to some other entity set.

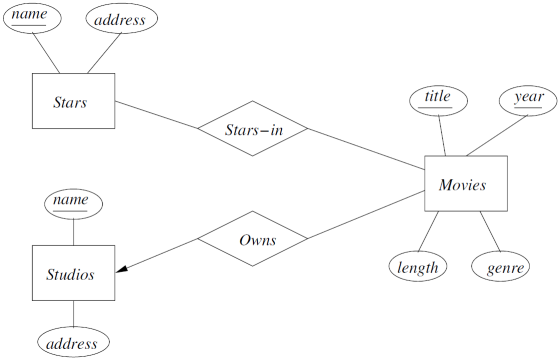
From non-weak entity sets to relations

A non-weak entity set is converted to a relation of the same name and with the same set of attributes. The attributes that are a key in the E/R diagram are a key in the relation.

This relation will not have any indication of the relationships in which the entity set participates.

Example

For example the entity sets *Stars*, *Studios* and *Movies* in the following E/R diagram become the relations *Stars(<u>name</u>, address)*, *Studios(<u>name</u>, address)*, *Movies(<u>title</u>, <u>year</u>, length, genre)*.



From E/R relationships to relations

Relationships in the E/R model are converted to relations.

The relation obtained from a relationship R has as key attributes the key attributes of each entity set involved in R. The attributes of the relationship (if any) become also attributes of the relation.

If one entity set is involved several times in a relationship in different roles, then its key attributes each appear as many times as there are roles. We must rename the attributes to avoid name duplication.

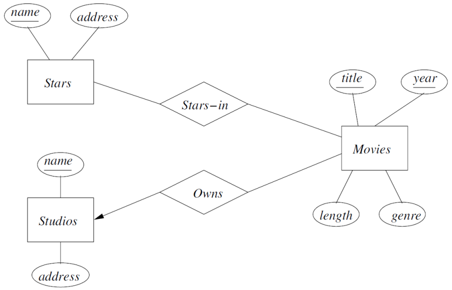
Rename to avoid name duplication or ambiguity

It can happen that an entity set E has a key attribute with the same name as an attribute of a relationship R into which it is involved. In that case the relation created from R has two attributes with the same name. In that situation in order to avoid name duplication we have to rename attributes.

We can also rename an attribute in order to make it easier to understand from which entity set it comes from.

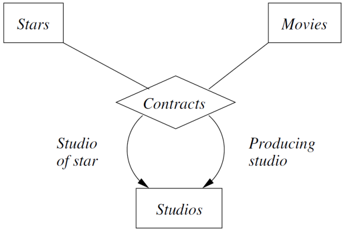
Example 1

The relationships *Owns* and *Stars-in* in the E/R diagram below become the relations *Owns(<u>title</u>, <u>year</u>, <u>studioName</u>)* and *Stars-In(<u>title</u>, <u>year</u>, <u>starName</u>)*. We have renamed the attributes *name* coming from *Studios* and *Stars* respectively to *studioName* and *starName*.



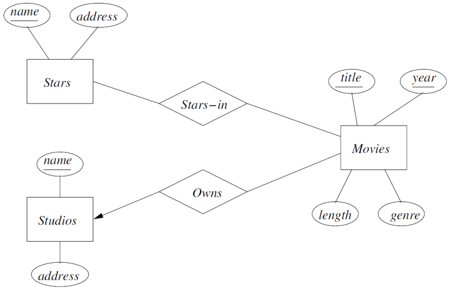
Example 2

Consider the E/R diagram below with a multiway relationship and two roles for the entity set *Studios* (we have omitted the attributes: *Star* has key *name*, *Movies* has key *title*, *year* and *Studios* has key *name*).
The relation obtained from this diagram is
Contracts(<u>starName</u>, <u>title</u>, <u>year</u>, <u>studioOfStar</u>, <u>producingStudio</u>).



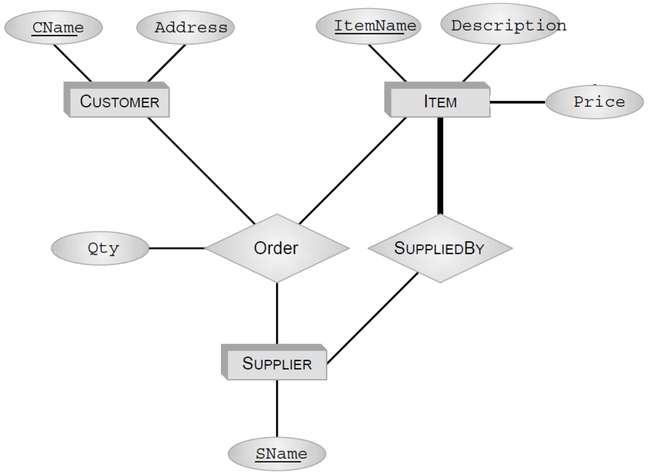
Example: conclusion

Stars(<u>name</u>, address), Studios(<u>name</u>, address),
Movies(<u>title</u>, <u>year</u>, length, genre), Owns(<u>title</u>, <u>year</u>, <u>studioName</u>)
Stars-In(<u>title</u>, <u>year</u>, <u>starName</u>)



Exercise

Convert the following E/R diagram into a relational schema.



Solution

CUSTOMER(<u>CName</u>, Address)

ITEM(<u>ItemName</u>, Description, Price)

SUPPLIER(<u>SName</u>)

Order(<u>CName</u>, <u>ItemName</u>, <u>SName</u>, Qty)

SuppliedBy(<u>SName</u>, <u>ItemName</u>)

Combining relations: an example

The previous example shows that the relations *Movies* and *Owns* have both <u>title</u> and <u>year</u> as key. Moreover since the relationship from *Movies* to *Owns* is many-one, the <u>title</u> and <u>year</u> of an entity of *Movies* uniquely determine an entity in *Studios*.

So it would good to merge *Movies* to *Owns* to get a relation with attributes all the attributes of the two relations (we call this relation *Movies*, but we could have chosen another name):
Movies(<u>title</u>, <u>year</u>, length, genre, studioName)

We observe that *studioName* is not a key anymore, because it is functionally determined *title* and *year*.

Combining relations

We generalize this situation. When there is a many-one relationship R from an entity set E to an entity set F, we can create two relations, instead of three: one for E and R that we describe below and one for F as usual.

The new relation for E and R will have as attributes

1. all the attributes of E,
2. the key attributes of F and
3. any attributes belonging to R.

The the key attributes of E will be the key attributes of this relation.

For an entity e of E that is not related to any entity of F, the attributes of types (2) and (3) will have null values in the tuple for e.

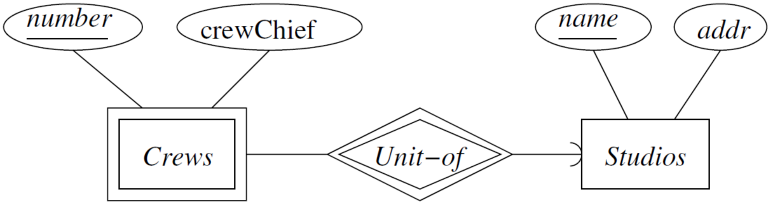
Handling weak entity sets

When a weak entity set appears in an E/R diagram, we need to do the following:

1. The relation for the weak entity set W itself must include not only the attributes of W but also the key attributes of the supporting entity sets and the attributes of the supporting relations.
2. A supporting relationship R from the weak entity set W to a supporting entity set must not be converted to a relation.
3. The relation for any relationship in which the weak entity set W appears (which is not a supporting relation for W) must use as a key for W all of its key attributes, including those of other entity sets that contribute to W's key.

Example

The diagram below



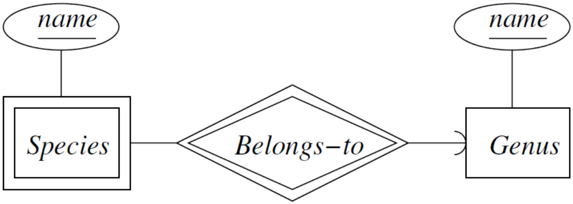
is converted into the two relations

Studios(<u>name</u>, addr)

Crews(<u>number</u>, <u>studioName</u>, crewChief)

Exercise

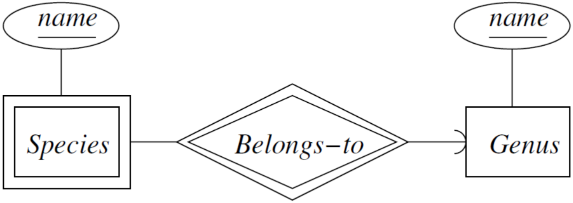
Convert the following diagram



into a relational schema.

Solution

The diagram



is converted into the relational schema

Genus(<u>name</u>)

Species(<u>name</u>, <u>genusName</u>)

5. Converting subclass structures to relations

Conversion strategies

We present the strategies used to convert an E/R isa hierarchy into a relational schema.

We observe that we will not create a relation for the isa relationship, but only for the isa hierarchy.

E/R isa hierarchy

We recall that in an isa hierarchy:

- there is a root entity set for the hierarchy,
- this entity set has a key that serves to identify every entity represented by the hierarchy, and
- a given entity may have components that belong to the entity sets of any subtree of the hierarchy, as long as that subtree includes the root.

Three main strategies

The three main strategies to convert an E/R isa hierarchy into a relational schema are:

1. *E/R-style conversion.* Follow the E/R viewpoint : for each entity set E in the hierarchy, create a relation that includes the key attributes of the root and any attributes belonging to E.
2. *Object-oriented approach.* Treat entities as objects belonging to a single class: for each possible subtree that includes the root, create one relation, whose schema includes all the attributes of all the entity sets in the subtree.
3. *Using null values to combine relations.* Create one relation with all the attributes of all the entity sets in the hierarchy. Each entity is represented by one tuple, and that tuple has a null value for whatever attributes the entity does not have.

E/R-style conversion

With this approach we create a relation for each entity set.

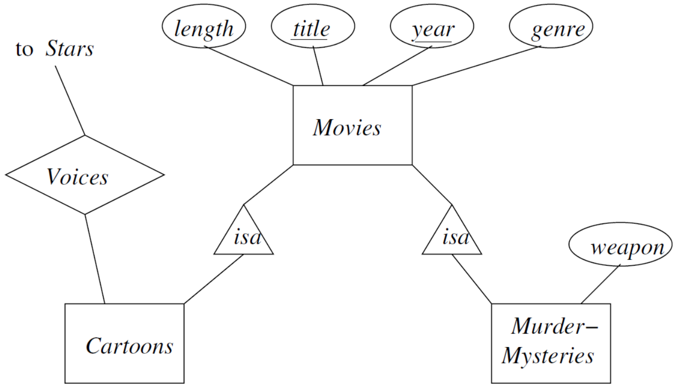
If the entity set E is not the root of the hierarchy, then the relation for E will include the key attributes of the root plus all the attributes of E.

In addition, if E is involved in a relationship, then we use these key attributes to identify entities of E in the relation corresponding to that relationship.

In an isa hierarchy an entity belongs to all the entities sets that are part of a subtree containing the root. Thus if e is an entity belonging to an entity set E and if E is not the root of the isa hierarchy, then e will appear in more than one relation, namely in all the relations corresponding to the entities sets that are part of the subtree containing E.

Example 1/3

Let us show how to convert the following diagram



into a relational schema by using the E/R-style conversion.

Example 2/3

The E/R-style conversion creates a relation for each entity set, thus the relational schema is

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

Murder-Mysteries(<u>title</u>, <u>year</u>, weapon)

Cartoons(<u>title</u>, <u>year</u>)

We observe that a murder mysteries movie will have tuples in the relation *Movies* and in the relation *Murder-Mysteries*; a cartoon will have tuples in the relation *Movies* and in the relation *Cartoons*; and a murder mysteries cartoon will have tuples in all the three relations *Movies*, *Murder-Mysteries* and *Cartoons*.

Example 3/3

The relationship *Voices* will be converted to the relation *Voices(<u>title</u>, <u>year</u>, <u>starName</u>)*, where *title* and *year* are the keys for *Cartoons* and *starName* is the key for *Stars*.

We notice that the relation *Cartoons* has a schema that is a subset of the schema for the relation *Voices*. We could choose to eliminate the relation *Cartoons* since its schema is contained in *Voices*.

However, there may be silent cartoons in our database, that have no voices, and we would therefore lose information about them if we eliminate the relation *Cartoons*.

Object-oriented approach

With this approach we create a relation for each subtree of the isa hierarchy. That relation has all the attributes of all the entity sets in the subtree.

An entity can have components in at most one of these subtrees.

We call this approach “object-oriented” since it is motivated by the assumption that entities are “objects” that belong to one and only one class.

Example

The diagram shown before is converted into the relations

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

Murder-Mysteries(<u>title</u>, <u>year</u>, length, genre, weapon)

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

Murder-Mysteries_ Cartoons(<u>title</u>, <u>year</u>, length, genre, weapon)

The relationship *Voices* will be converted to the relation *Voices(<u>title</u>, <u>year</u>, <u>starName</u>)* as seen with the previous method.

Using null values to combine relations

If we are allowed to use NULL (the null value as in SQL) as a value in tuples, we can handle a hierarchy of entity sets with a single relation.

This relation has all the attributes belonging to the entity sets of the hierarchy.

An entity is then represented by a single tuple. This tuple has NULL in each attribute that is not defined for that entity.

Example

The diagram shown before is converted into the single relation
Movies(<u>title</u>, <u>year</u>, length, genre, weapon)

Those movies that are not murder mysteries would have NULL in the weapon component of their tuple. It would also be necessary to have a relation *Voices* to connect those movies that are cartoons to the stars performing the voices.

We observe that silent cartoons and non-cartoon movies cannot be distinguished in this relation. Both are those movies not appearing in the *Voices* relation.

Comparison of approaches

Each of the three approaches has advantages and disadvantages.

We make comparison based on some predefined goals:

- Minimize the number of relations.
- Minimize space and avoid repeating information.
- Minimize the number of queries involving several relations.
- Compatibility with the object oriented model.

Minimize the number of relations

The null values approach is the best for this goal since it only uses one relation.

In the straight-E/R approach, there is one relation per entity set in the hierarchy. So if there are n nodes there will be n relations.

With the object-oriented approach the number of relations grows like 2^n.

Minimize space and avoid repeating information 1/2

The object-oriented approach is the best one with respect to this goal since the data for an entity are just in one table and all the attributes are relevant for all the tuples.

Minimize space and avoid repeating information 2/2

The null values approach also has only one tuple per entity, but these tuples are “long”; i.e., they have components for all attributes, whether or not they are appropriate for a given entity. If there are many entity sets in the hierarchy, and there are many attributes among those entity sets, then a large fraction of the space could be wasted in the nulls approach.

The straight-E/R method has several tuples for each entity, but only the key attributes are repeated. Thus, this method could use either more or less space than the null values method.

Minimize the number of queries involving several relations 1/3

It can be expensive to answer queries involving several relations, so we would prefer to find all the attributes we needed to answer a query in one relation. The null values approach uses only one relation for all the attributes, so it has an advantage in this regard.

The other two approaches have advantages for different kinds of queries.

Minimize the number of queries involving several relations 2/3

A query like “what films of 2008 were longer than 150 minutes?” can be answered directly from the relation *Movies* in the straight-E/R approach.

However, in the object-oriented approach, we need to examine *Movies*, *Cartoons*, *Murder-Mysteries* and *Murder-Mysteries_ Cartoons*, since we have to check all movies belonging to any of these four relations.

Minimize the number of queries involving several relations 3/3

On the other hand, a query like “what weapons were used in cartoons of over 150 minutes in length?” gives us trouble in the straight-E/R approach. We must access *Movies* to find those movies of over 150 minutes. We must access *Cartoons* to verify that a movie is a cartoon, and we must access *Murder-Mysteries* to find the murder weapon.

In the object-oriented approach, we have only to access the relation *Murder-Mysteries_ Cartoons*, where all the information we need will be found.

Compatibility with the object oriented model

If our database is embedded in a program written in an object-oriented language, then it could be desirable that the relational model be close to the object-oriented model.

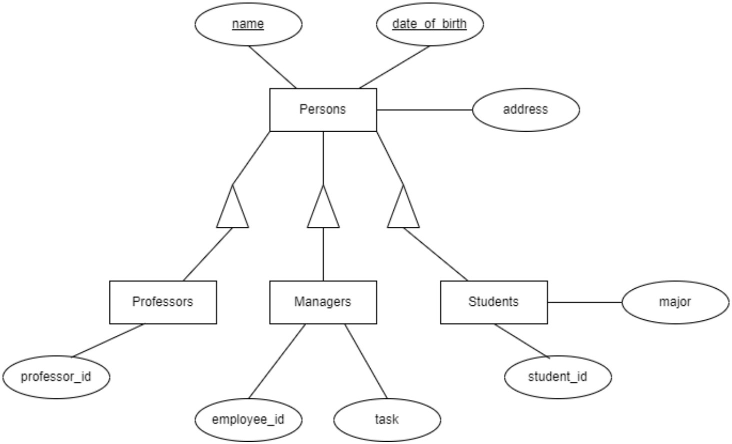
The object-oriented approach is the best one with respect to this goal since each relation represents a class.

With the straight-E/R approach the relations are close to classes, but a single relation does not have all the attributes of a class, which are spread into different relations.

The null values approach creates a single relation which is not related at all to a class. In an object-oriented setting this approach can be very confusing.

Exercise

Convert the following subclass hierarchy into relations schemas using the three approaches illustrated in the lectures.



Solution 1/3

E/R-style conversion.

Persons(<u>name</u>, <u>date_of_birth</u>, address)

Professors(<u>name</u>, <u>date_of_birth</u>, professor_id)

Managers(<u>name</u>, <u>date_of_birth</u>, employee_id, task)

Students(<u>name</u>, <u>date_of_birth</u>, student_id, major)

Solution 2/3

Object oriented approach.

Persons(name, date_of_birth, address)

Professors(name, date_of_birth, address, professor_id)

Managers(name, date_of_birth, address, employee_id, task)

Students(name, date_of_birth, address, student_id, major)

Professors-Managers(name, date_of_birth, address, professor_id, employee_id, task)

Professors-Students(name, date_of_birth, address, professor_id, student_id, major)

Managers-Students(name, date_of_birth, address, employee_id, task, student_id, major)

Professors-Managers-Students(name, date_of_birth, address, professor_id, employee_id, task, student_id, major)

Solution 3/3

Null values approach.

Persons(name, date_of_birth, address, professor_id, employee_id, task, student_id, major)

The end.