DATABASES / DATABASES I

2024/2025 – Fall Semester

Practical Classes Workbook

Part 1- Class 2

Contents

Data Modelling

Conceptual Model (Entity Relationship Model)

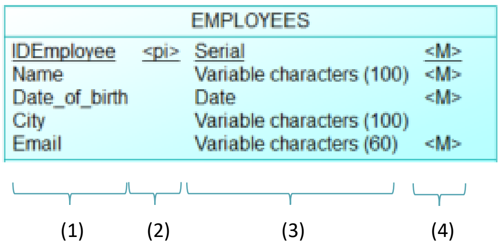
There is a reason for this model to be also called Entity Relationship Model. It represents a data structure using **entities** and **relationships** between these entities. Let's go over the core concepts of the Conceptual Model.

Entities

An entity is an object or concept about which we want to store information. It works as a **class** that stores multiple **instances**. Some examples would be an entity called EMPLOYEES or COMPANIES – inside, they would have information about each employee and about each company, respectively. The different pieces of information stored inside an entity are called **attributes**. For the EMPLOYEES we would have attributes like "Name" and "Date of birth".

In an Entity Relationship Model, an entity is represented graphically by a rectangle with its name on top and its attributes inside:

Figure 1 - visual representation of an entity



EMPLOYEES

IDEmployee	<pi>	Serial	<M>
Name		Variable characters (100)	<M>
Date_of_birth		Date	<M>
City		Variable characters (100)	
Email		Variable characters (60)	<M>

Instances

As said before, each entity can be thought of as a class that has its instances. But what are the instances in practice?

In Power Designer, we draw the entities, their attributes, and the relationships between entities but we don't see the instances. Each entity can be seen as a table and if we draw the instances of the entity in the figure above in an excel sheet it would correspond to something like:

IDEmployee	Name	Date_of_birth	City	Email
1	Francisca	20/08/1990	Lisboa	francisca@gmail.com
2	Pedro	17/03/1998	Setúbal	pedro@gmail.com
3	Ana	09/05/1971	Lisboa	ana@gmail.com

This entity currently has 3 instances – 3 employees – and each of them can be thought of as a row in the table EMPLOYEES. In Power Designer, we see the attributes one below the other, but if we think of an entity as a table with attributes and rows, the attributes are actually the columns.

Attributes

The attributes can be seen as the properties or characteristics of an entity, and they are atomic - there must not be an attribute that stores more than one piece of information, for example, "Email_and_Address".

All the instances of an entity have the same attributes, what varies are the values of the attributes.

In the entity EMPLOYEES, we have 5 attributes: "IDEmployee", "Name", "Date_of_birth", "City" and "Email" (1).

For each attribute, a **data type** must be specified (3). In Power Designed there are a lot of possible data types that can be assigned to the attributes, the possibilities can be seen here. Additionally, it is possible to further specify the set of values that an attribute can take – if the company was only recruiting people living in Lisbon and Setúbal, we can create a check constraint to have a constraint that only allows the insertion of these two values in the "City" field.

Besides data types, for each attribute, we can also specify if the attribute is **mandatory** or optional (4). If an attribute is mandatory, when an instance is inserted in the entity, the value for that attribute must be filled. If it is optional, it might be filled or not.

Primary identifiers

One essential concept related to the attributes of an entity is the **primary identifier** (2) – aka primary key in the physical diagram. A primary identifier attribute is something that identifies each instance of an entity and is usually underlined in the visual representation. In the example above, "IDEmployee" is the primary identifier of the entity, meaning each instance has their own "IDEmployee". There are **no duplicates** in this attribute. This is a **simple identifier** as it is only composed by one single attribute. There can be the case where we have a **compound identifier** – one attribute is not enough to identify an instance and the identifier is made of a set of attributes. If there is a compound identifier, the attributes that are part of it can have duplicates individually, but if we look at all the attributes making the primary identifier, then there must be no duplicates.

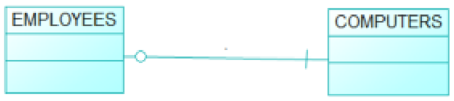
The primary identifier attributes do not accept null values.

Relationships

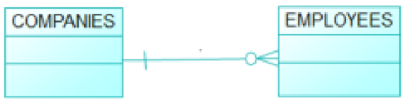
The relationships exist between entities and have no attributes of their own. To characterize a relationship, we must talk about its **cardinalities**. They are defined by the user when building the conceptual model and their goal is to restrict the relationship. There is a minimum and a maximum cardinality in each side of the relationship.

The **maximum cardinality** is the maximum number of instances from one entity that **could** participate in a relationship -> 1 or many. The **minimum cardinality** is the minimum number of instances from one entity that **must** participate in a relationship -> 0 (optional) or 1 (mandatory).

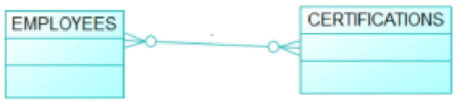
In the example below we have a 1:1 (**1 to 1**) relationship in terms of maximum cardinality. Power designer representation simplifies the representation by only showing the minimum cardinalities. In this example, an employee must have one and only one computer associated with him, but a computer can have 0 or 1 employee associated. Actually, if the graphical representation was complete, we would have 0:1 in the employees' side and 1:1 in the computers side as on the **left side the minimum cardinality is 0 and the maximum is 1**, while in the **right side the minimum and maximum cardinalities are both 1**. In terms of data insertion, this means that if we insert an instance in the entity EMPLOYEES, there must already exist an instance in the entity COMPUTERS to which the new employee will point to. On the other side, an instance of the entity COMPUTERS can be inserted without pointing to any employee.



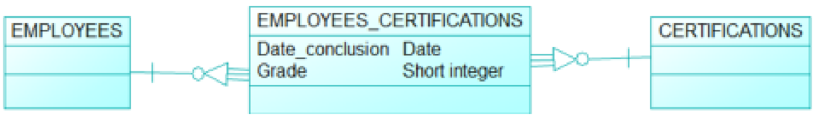
Let's consider the next example that depicts the relationship between Companies and Employees (again we have graphical simplification by Power Designer). In this second example we have a 1:N (**one to many**) relationship. In this example, a company can have 0 or many employees associated with it, but an employee must be associated to one and only one company. On the **left side the minimum and maximum cardinalities are both 1**, while in the **right side the minimum cardinality is 0 and the maximum is many**. In terms of data insertion, this means that if we insert an instance in the entity EMPLOYEES, there must already exist an instance in the entity COMPANIES to which the new employee will point to. On the other side, an instance of the entity COMPANIES can exist without employees pointing to it.



In the third example below, we have a N:M (**many to many**) relationship. In this example, an employee can have 0 or more certifications associated with him, while a certification can also have 0 or more employees associated to it (many employees can have the same certification). On both **left and right side the minimum cardinality is 0 and the maximum cardinality is many**. In terms of data insertion, this means that an instance of the entity CERTIFICATIONS can exist without being related to an EMPLOYEE and an EMPLOYEE can also exist without having a certification.



The example above sometimes evolves to something else… What if we want to store data regarding each certification of each employee, for example, the date it was concluded, and the grade obtained. These 2 attributes are not only related to an employee or to a certificate, they are related to both. Where should we add them in the data model?

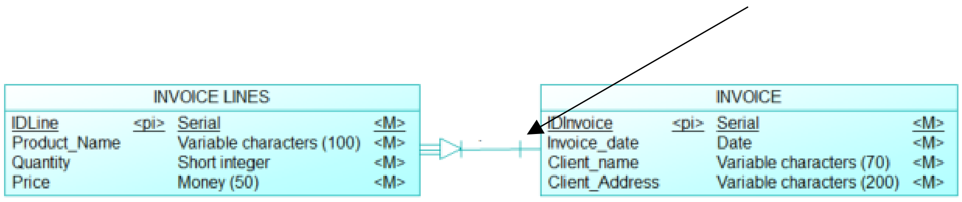


We must create a middle entity like EMPLOYEES_CERTIFICATIONS which will reference an employee and a certification and store the data that is specific to each combination of employee and certification. Usually, when we do this, the middle entity becomes a weak entity (see section below), but this is not mandatory, we can have the symbol of many we have seen before: . The choice depends on the attributes that we want to have as primary key in the middle table (see section below).

Dependency relationships

An entity can depend on another entity and the dependent one is called a weak entity. The weak entity will “inherit” the primary identifier of the “parent” entity to be part of its own primary identifier. Wanting this to happen is one of the reasons to create a dependency relationship.

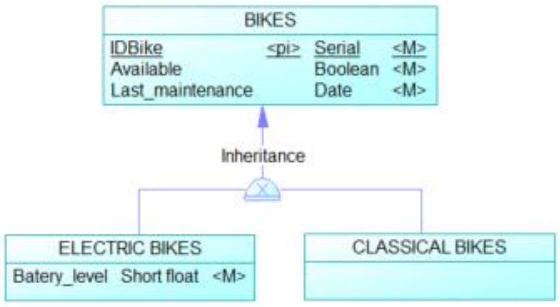
On the “parent” side, the minimum and maximum cardinality is always 1 as the weak entity depends on the other entity, so it should always be related with it.


INVOICE LINES
IDLine <pi> Serial <M>
Product_Name Variable characters (100) <M>
Quantity Short integer <M>
Price Money (50) <M>
INVOICE
IDInvoice <pi> Serial <M>
Invoice_date Date <M>
Client_name Variable characters (70) <M>
Client_Address Variable characters (200) <M>

Why do we need the primary key of the INVOICE to be part of the primary key of INVOICE LINES? Because in all the invoices there will be an “IDLine” with the value 1, 2, 3 and so on depending on the number of lines in the invoice. This means that in the INVOICE LINES we will have multiple instances with the same “IDLine”, but as we saw before the primary key must be unique (no duplicates are allowed). The solution is to have a **compound identifier** which includes the “IDLine” and the “IDInvoice” - this will be the result of the dependency relationship as the weak entity receives the primary key of the “parent” entity as part of its own identifier (see section Dependency Relationship in the chapter Relational Model). This way we can have an instance with “IDLine” = 1 and “IDInvoice” = 1 and another instance with “IDLine” = 1 but “IDInvoice” = 2.

Inheritance relationships

The inheritance relationships enable us to establish a **hierarchical structure** among entities, where child entities inherit the attributes (all) from their parent entity. The child entities are specific cases of the parent entity. It is mostly used when there are common attributes between all the child entity types (which are stored in the parent entity) but then there are specialized attributes to store in each specific child entity. We can see an example below, where electric and classical bikes (child entities) are types of bikes (parent entity) and for the electric ones we want to store a specific attribute. In the child entities we don't need to have a primary key because they will inherit the attributes in the parent entity, which includes one.



The inheritances have two characteristics: **completeness** and **exclusiveness**. An inheritance is considered complete if an instance in the parent entity is always in one of the child entities. In the example above, if we consider that the only existing types of bikes are electric and classical then the relationship is complete. The completeness is represented with the base of the inheritance symbol being present or not. The relationship is exclusive if one instance in the parent entity has an associated instance in one and only of the child entities, which is the case in our example – one bike is either electric or classical. The exclusiveness is represented by the X in the symbol in the middle.

Exercises

Entities and attributes

1. In our database we want to store data about cars. The data we want to store about each car is: Year of fabrication, License plate and colour.

   i. Draw a data model (entities and attributes) to store this data.

   ii. Each entity must have a primary identifier. In the entity drawn in i., which of its attributes could be used as a primary identifier and why?

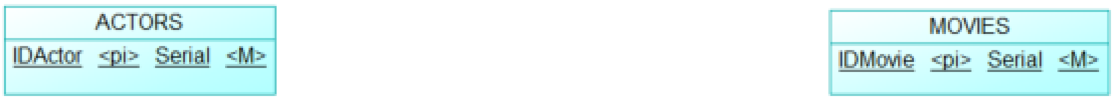
   iii. In terms of data insertion, what does it mean if the colour of the car is a non-mandatory attribute?

Relationships

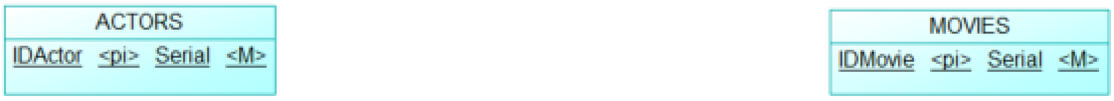
2. Draw the relationship between the 2 entities below considering that each person can have none or multiple phones and a phone belongs to a single person.

PEOPLE
PHONES

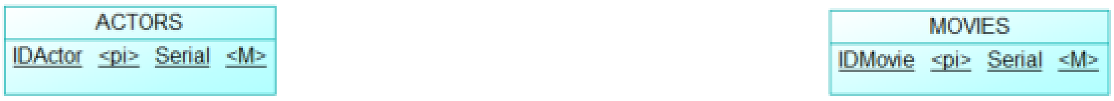
3. Draw the relationship between the 2 entities below considering that each movie can have none or many actors and each actor can take part in many movies.



   i. Take the same example as in Exercise 4, but now complete the model knowing that we need to store the name of the character each actor played in each movie he/she participated. Assume the actor can only play one character in each movie.



   ii. What needs to change if we want to allow the possibility of an actor playing more than one character in a movie?



4. Consider the requirements described in Exercise 1.

   i. Draw the conceptual data model using Power Designer.
   ii. Now, update the data model considering that a renter can answer multiple times to the same announcement. What needs to change?