Relational databases

Joel Silva, PhD

February 2, 2021

NOVA Information Management School

Introduction

Introduction

In this lecture we'll cover most of the database design and development concepts necessary to go from the idea to the database itself.

We'll take the "learn as you go" approach, where we start with a brief terminology so that we all share the same vocabulary and then we'll move directly to the problem.

As we solve the problem, I introduce the main concepts and procedures. At the end, I hope, you will have a balance between theory and practice.

Terminology

Basic terminology

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Figure 1: Database basic terminology

Basic terminology

An **attribute** is a characterizing piece of data. Alternative terms are field and column.

A **tuple** is an instance of the class of things that we are modeling. Alternative terms are record and row.

A **relation** is a set of tuples that have the same attributes; the class of things we're modeling. Alternative terms is table.

The **cardinality** and **degree** are the number of tuples and the number of attributes of a table, respectively. However, people will simply say "number of lines" and "number of rows".

Basic terminology

A **primary key** is an attribute or set of attributes that uniquely specifies a tuple within a table.

A **foreign key** is an attribute in a table that matches the primary key of another table.

These two concepts are one of the most powerful inventions in the IT world. Worship them!

Basic terminology

The **data type** is the basic kind of data that can be stored in a column. The data types are of different types: integers, float-poing numbers, strings, dates, etc.

The **domain** of an attribute is the subset of the data type that make sense for a particular application. For example, *birthdays* are of type date but maybe we are only interested in dates above 01-01-2000. So the domain of *birthdays* is the set of dates above 01-01-2000. Defining the domain of an attribute is very important to evaluate the consistency of the data. We don't want to introduce odd records.

Database modeling

Modeling - What is

The purpose of databases, besides the obvious objective of saving large amounts of data, is to allow users to query and extract useful information about a particular situation.

Modeling a problem consists in understanding what are the entities and what are their relations so that we can then translate that onto a database.

If the modeling component is correctly created, we are then able to levarage database technology and concepts to extract information about the situation that otherwise would be difficult to obtain.

We'll approach database modeling from a very pragmatic point of view.

Modeling - Chen's linguistic approach

In his "English, Chinese and ER diagrams" (1997) Peter Chen highlighted a pragmatic way to database modeling.

The basic idea is to get a set of propositions that describe the problem, and perform a linguistic analysis of the problem. In other worlds, break the set of propositions into its basic grammatical elements.

Chen proposed the following "rules of thumb" for mapping natural language descriptions into ER diagrams:

Modeling - Chen's linguistic approach

- **Common noun** represents a type of things, and should be modeled as a table
- **Proper noun** represents a single instance, and should be mapped onto a tuple of a table
- **Transitive verb** represents a relationship type between two or more things, and should be mapped either onto a table or a foreign key
- **Intrasitive verb** represents an attribute type, and thus is mapped onto an attribute of a table
- **Adjective** represents an attribute of a thing, and this is mapped onto an attribute of a table
- **Adverb** represents an attribute of a relationship between things, and thus is an attribute of a table

Modeling - Classic Models problem

The Classic Models Inc. example database has been developed as part of the Eclipse BIRT (Business Intelligence Reporting Tools) project.

Its main goal is to be obvious and simple, yet able to support a wide range of interesting query examples.

The database represents a fictitious company: Classic Models Inc. which buys collectable model cars, trains, trucks, buses, trains and ships directly from manufacturers and sells them to distributors across the globe.

Modeling - Classic Models problem

Next you are presented with a description of the situation that we will model.

Let's read each slide carefully and perform Chen's magical formula.

We'll start by first identifying the entities (nous) and then we'll try to find the relationships between these entities.

Modeling - Classic Models problem

Classic Models Inc. has 7 offices worldwide (San Francisco, Boston, NYC, Paris, Tokyo, Sydney, London) and is headquartered in San Francisco, CA.

Based on geography each office is assigned to a sales territory (APAC, NA, EMEA or JAPAN).

Modeling - Classic Models problem

The company has 23 employees: 6 Execs and 17 Sales Reps, all assigned to one of the company's seven offices.

Sales Reps are also assigned to a number of customers (distributors) in their respective geographies that theysell to.

New Sales Reps (that are still in training) don't have customers assigned to them. Each Sales Rep reports to the Sales Manager for his/her territory.

The only exceptions are the two Sales Reps in the Tokyo office. One of them acts as a Sales Manager and reports directly to the VP of Sales. The second one reports to him.

Modeling - Classic Models problem

The Execs: President, VP Sales, VP Marketing, Sales Manager (JAPAN, APAC), Sales Manager (EMEA), Sales Manager (NA) don't work directly with customers. Each Sales Manager reports to the VP of Sales. Nobody reports to the VP of Marketing. The two VPs report to the company's President.

Modeling - Classic Models problem

Classic Models Inc. has 122 customers across the world. Approximately 20 of those are brand new customers that don't have an assigned sales rep and haven't placed any orders yet. Each customer has a credit limit which determines their maximum outstanding balance.

Modeling - Classic Models problem

Customers place their orders and expect to receive them approximately within 6 to 10 days. Once an order is placed it's assembled and shipped within 1 to 6 days (7-8 for Japan). There are a total of 326 orders, which span the period from 1/1/2003 to 1/6/2005. Orders can be in one of these states: In Process (the initial state for all orders), Shipped, Cancelled (used to indicate that the customer called to cancel the order right after the order was placed and typically before it was shipped), Disputed (used to indicate that the customer received the order but doesn't like it), Resolved (used to indicate that the order was disputed, but successfully resolved) or On Hold (used to indicate that the order will not ship until a payment is received because the customer's credit limit has been exceeded). Approximately 93% of the orders are in the Shipped state.

Modeling - Classic Models problem

Each order contains an average of 9 unique products (order line items) with an average quantity of 35 per product (so there is an average total of 9x35 items per order). Each order line item reflects the negotiated price per product (which is based on the corresponding product's MSRP) as well as the quantity per product.

Modeling - Classic Models problem

Customers make payments (by check) on average 2-3 weeks after they place an order. In some cases one check covers more than 1 order

Modeling - Classic Models problem

Products are classified as 7 distinct product lines. Each product line is associated with a text description, html description as well as with an image.

Modeling - Classic Models problem

What are the main entities you have identified from the propositions set?

Modeling - Classic Models problem

- Offices
- Employees
- Orders
- Products
- Payments
- Product lines

Modeling - Classic Models problem

Is anything missing?

Modeling - Classic Models problem

How are orders characterized?

Modeling - Classic Models problem

Do you think we can fit all that data in one row to characterize one order?

Modeling - Classic Models problem

Probably not. So to solve that problem, we create a new table to hold the details of a order.

The final list of entities is then:

Modeling - Classic Models problem

- Offices
- Employees
- Orders
- Products
- Payments
- Product lines
- Order details

Modeling - Classic Models problem

Next question: What are the relationships between these elements?

Relationships between entities

Modeling - Relationships

Before we go further, let's learn more about entities' relationships.

When we talk about entities' relationships, we think about relationships between two entities, and these are the most frequent. However, we can have relationships between more than two elements.

To characterize a relationship we need to determine its **multiplicity** and its **membership**.

Modeling - Multiplicity relationships

There are three types of relationships multiplicity:

- One-to-one relationship
- One-to-many relationship
- Many-to-many relationship

Modeling - One-to-one relationships

If two entities A and B are in a **one-to-one relationships** then for each instance of A is in relation with one and only one instance of B and vice-versa. This means that if we know the instance of B, we are able to know the instance of A that relates to that instance of B.

For example, in a concert hall, each ticket holder has a seat for a single performance (the seat number will appear on the ticket).

Only one person can sit in one seat at each performance. The relationship between a member of the audience and a seat is therefore one-to-one.

Modeling - One-to-one relationships

If two entities A and B are in a **one-to-one relationships** then for each instance of A is in relation with one and only one instance of B and vice-versa. This means that if we know the instance of B, we are able to know the instance of A that relates to that instance of B.

For example, in a concert hall, each ticket holder has a seat for a single performance (the seat number will appear on the ticket).

Only one person can sit in one seat at each performance. The relationship between a member of the audience and a seat is therefore one-to-one.

Modeling - One-to-many relationships

If two entities A and B are in a **one-to-many relationships** then for each instance of A is in relation with one and only one instance of B but the same instance of B may be in relation with many different instances of A.

This means that if we know the instance of B, we are not able to determine the instance of A that relates to that instance of B but a group of instances of A in relation with that instance of B.

For example, an orchestra will have more than one musician playing a particular type of instrument; for example, it is likely that there will be several members of the orchestra each playing a violin. The relationship is therefore one-to-many from a type of musical instrument to a member of the orchestra

Modeling - Many-to-many relationships

If two entities A and B are in a **many-to-many relationships** then for each instance of A is in relation with many instances of B, and vice-versa.

For example, there is a many-to-many relationship between ticket holders and concert performances, as one ticket holder may attend many performances, and each performance is likely to have many ticket holders present.

Modeling - Relation membership

The participation condition defines whether it is mandatory or optional for an entity to participate in a relationship.

This is also known as the membership class of a relationship.

As there are two kinds of participation conditions (mandatory and optional), and most entities are involved in binary relationships, it follows that there are four main types of membership relationships, as follows:

Modeling - Relation membership

- Mandatory for both entities
- Mandatory for one entity, optional for the other
- Optional for one entity, mandatory for the other
- Optional for both entities

Let see some examples:

Modeling - Relation membership

Mandatory for both entities: A member of staff must be assigned to a given department, and any department must have staff. There can be no unassigned staff, and it is not possible to have an 'empty' department.

Mandatory for one entity, optional for the other: Any member of staff must be attached to a department, but it is possible for a department to have no staff allocated.

Optional for one entity, mandatory for the other: A member of staff does not have to be placed in a department, but all departments must have at least one member of staff.

Optional for both entities: A member of staff might be assigned to work in a department, but this is not compulsory. A department might, or might not, have staff allocated to work within it.

Modeling - Relation multiplicity and membership

In this way, multiplicity is the maximum number of elements that are in relation with particular instance, and the membership is the minimum.

Modeling - Crow's foot notation

Often database models are simplified using a diagramme.

A common diagramme is call the crow's foot notation.

Let's see how we represent multiplicity and membership with this notation.

Modeling - Crow's foot notation

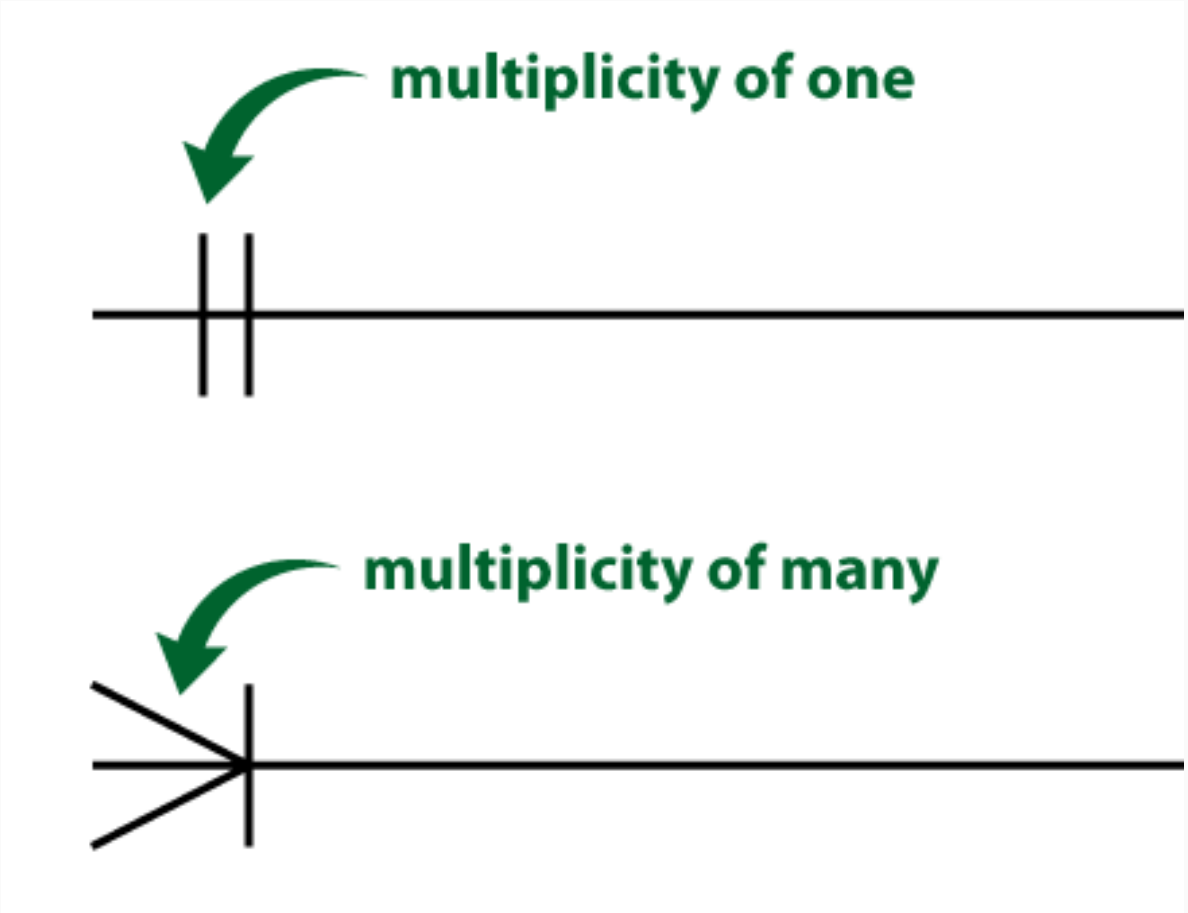


Figure 2: One vertical bar represents at most one element in relation, and the fork at most many. Ignore the rightmost vertical bar for now.

Modeling - Crow's foot notation

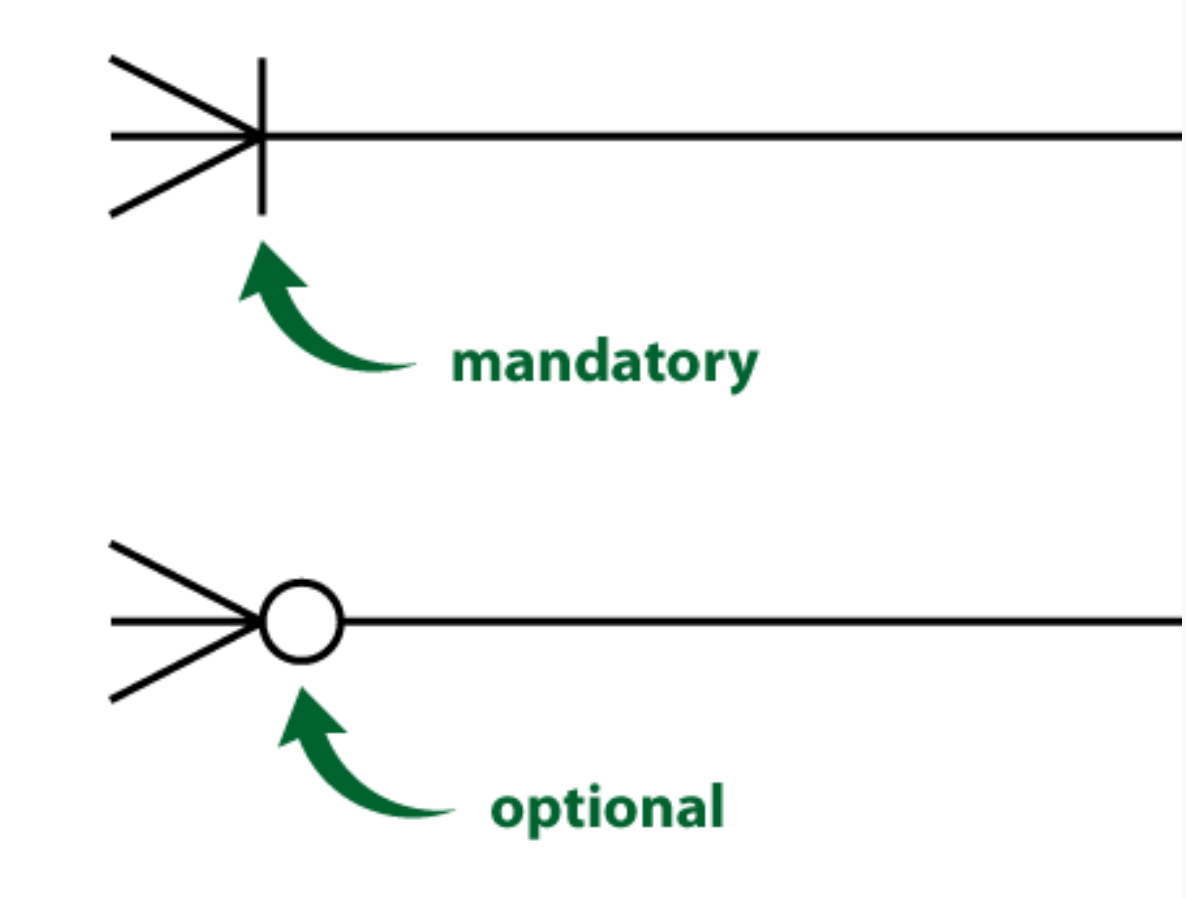


Figure 3: One vertical bar represents mandatory relation, i.e. at least one, and the circle represents option, i.e. zero or more.

Modeling - Crow's foot notation

Let's combine all these elements.

Modeling - Crow's foot notation

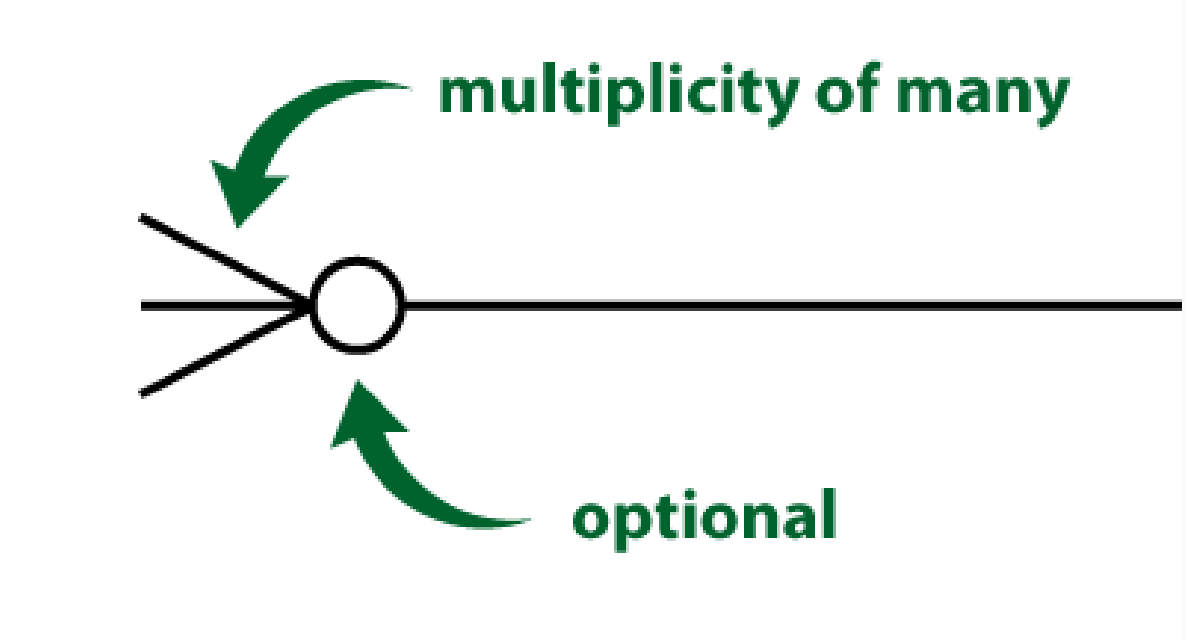


Figure 4: Zero or many elements in relation

Modeling - Crow's foot notation

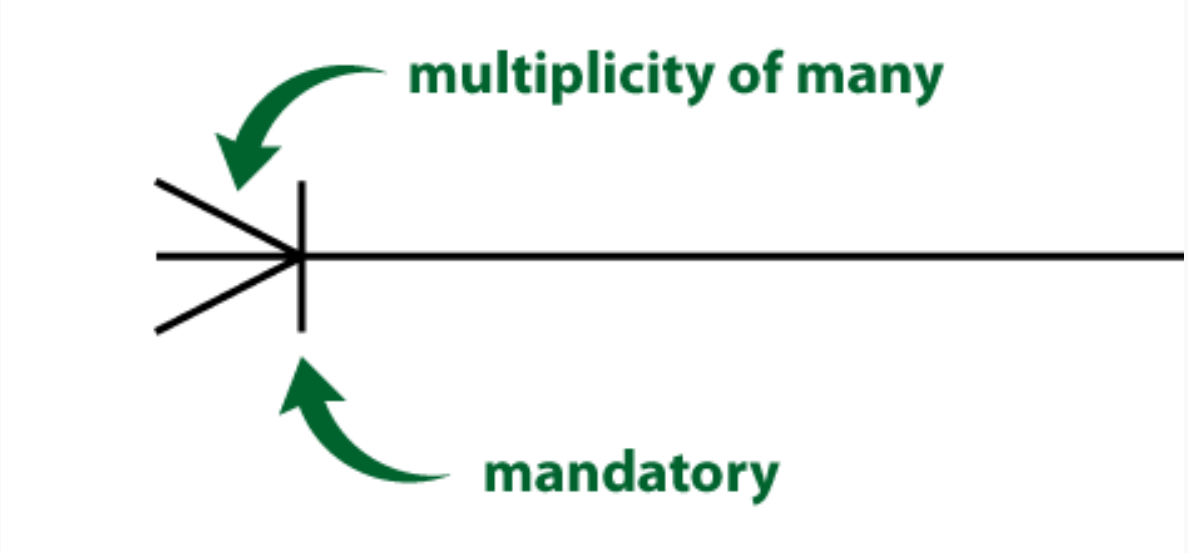


Figure 5: one or many elements in relation

Modeling - Crow's foot notation

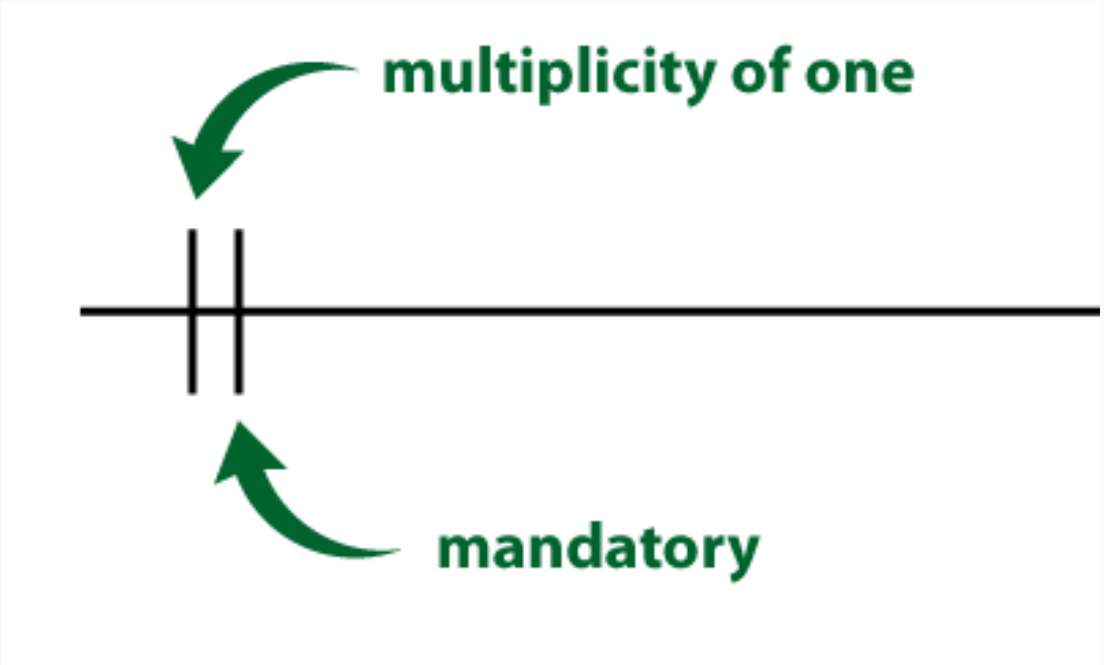


Figure 6: One and only one elements in relation

Modeling - Crow's foot notation

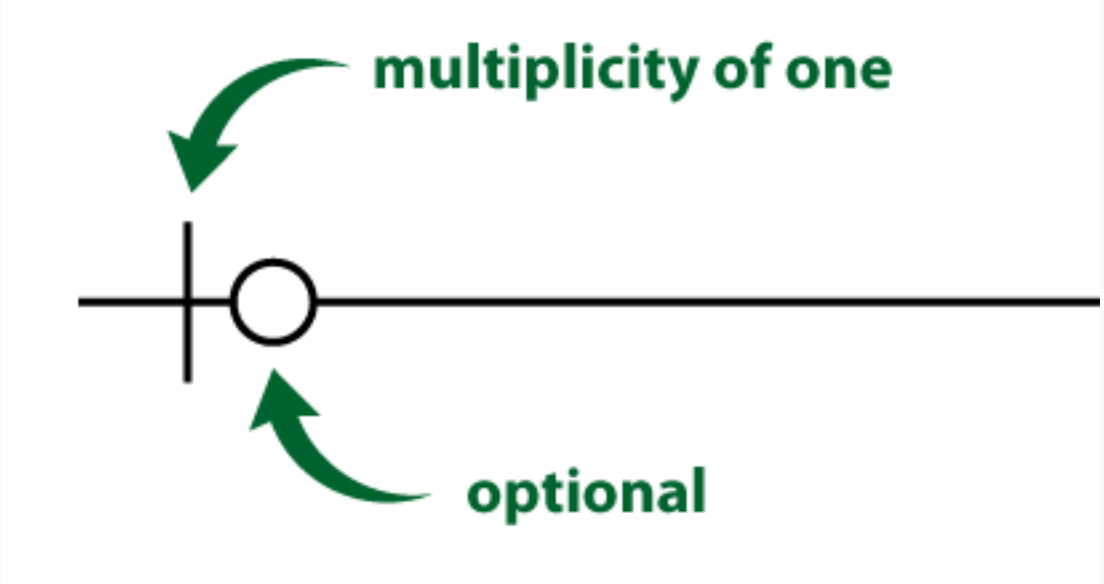


Figure 7: Zero or one elements in relation

Modeling - Crow's foot notation

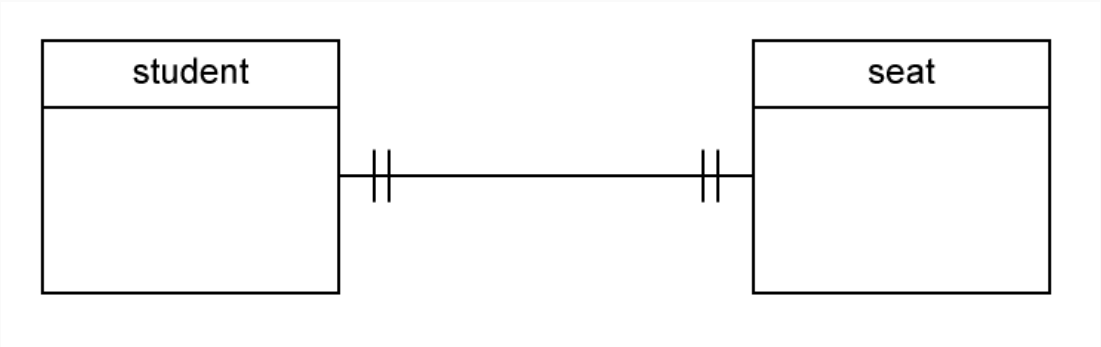


Figure 8: Example 1 - a student has one and only one seat, and a seat has one and only one student.

Modeling - Crow's foot notation

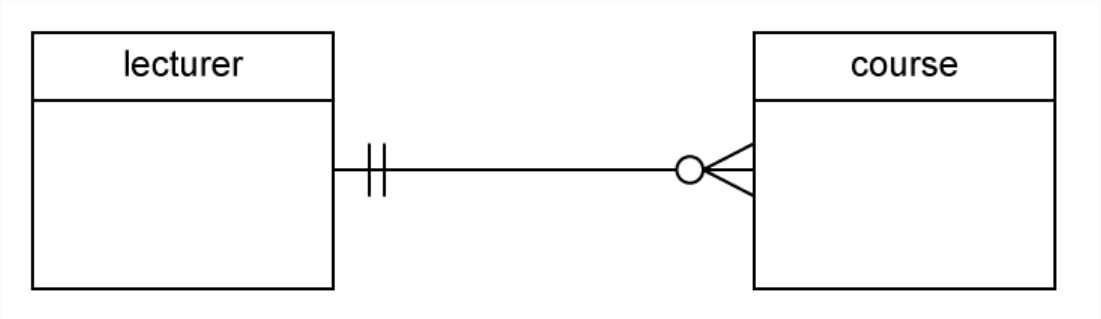


Figure 9: Example 2 - a lecture lectures zero or many courses, and a course is lectured by one and one lecturer.

Modeling - Crow's foot notation

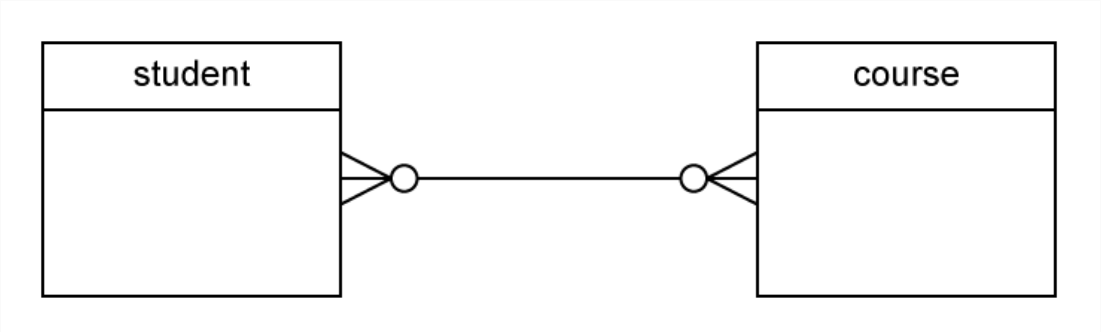


Figure 10: Example 3 - a student is enrolled in zero or many courses, and a course has zero or many students enrolled.

Back to our modeling problem

Modeling - Classic Models problem

Now that we know how entities relate, let's answser the question.

What are the relationships between these elements?

Modeling - Classic Models problem

If you want to know if two entities are in relation you should ask yourself:

"Is there any proposition where these two entities show up?"

If so, most likely you have found a relation that needs to be model.

Let's take two entities, *"products"* and *"product lines"*.

What do you know about the relation between these two entities?

Modeling - Classic Models problme

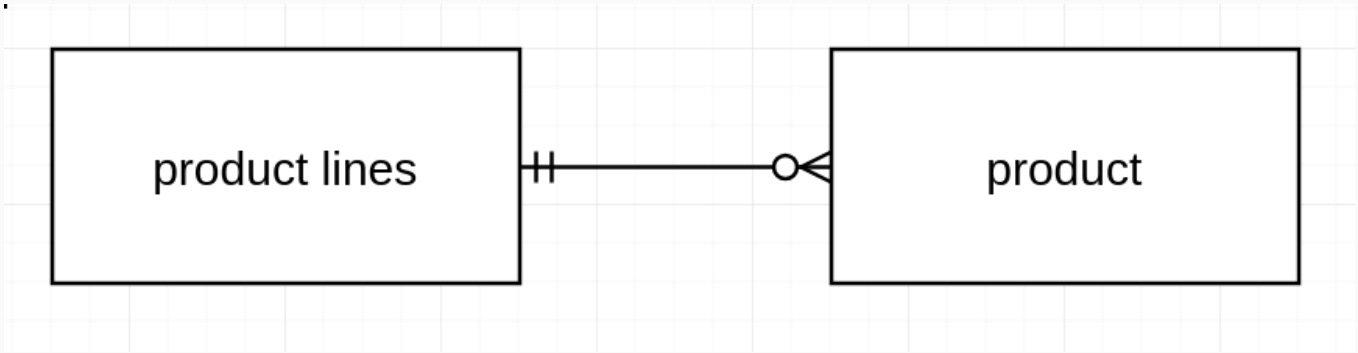


Figure 11: A product belongs to one and only one product line, and conversely a product line has many types of products.

Modeling - Classic Models problem

Have a look at figure 11.

The attributes are missing. Can you characterize these two entities?

Modeling - Classic Models problme

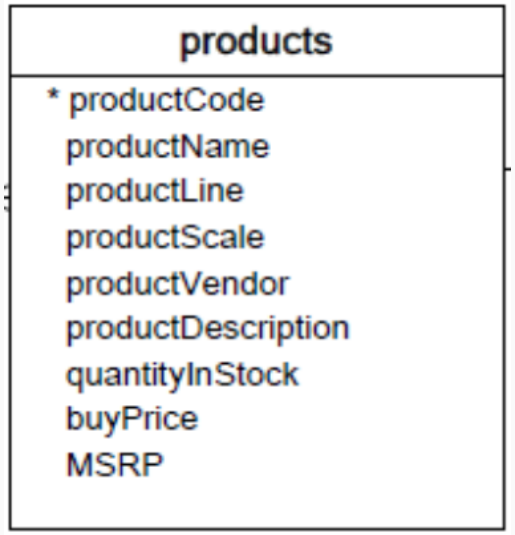


Figure 12: Product entitie with attributes. The * indicates the primary key.

Modeling - Classic Models problem

We have now all the tools necessary to compose the diagramme that describes the situation.

Modeling - Classic Models problem

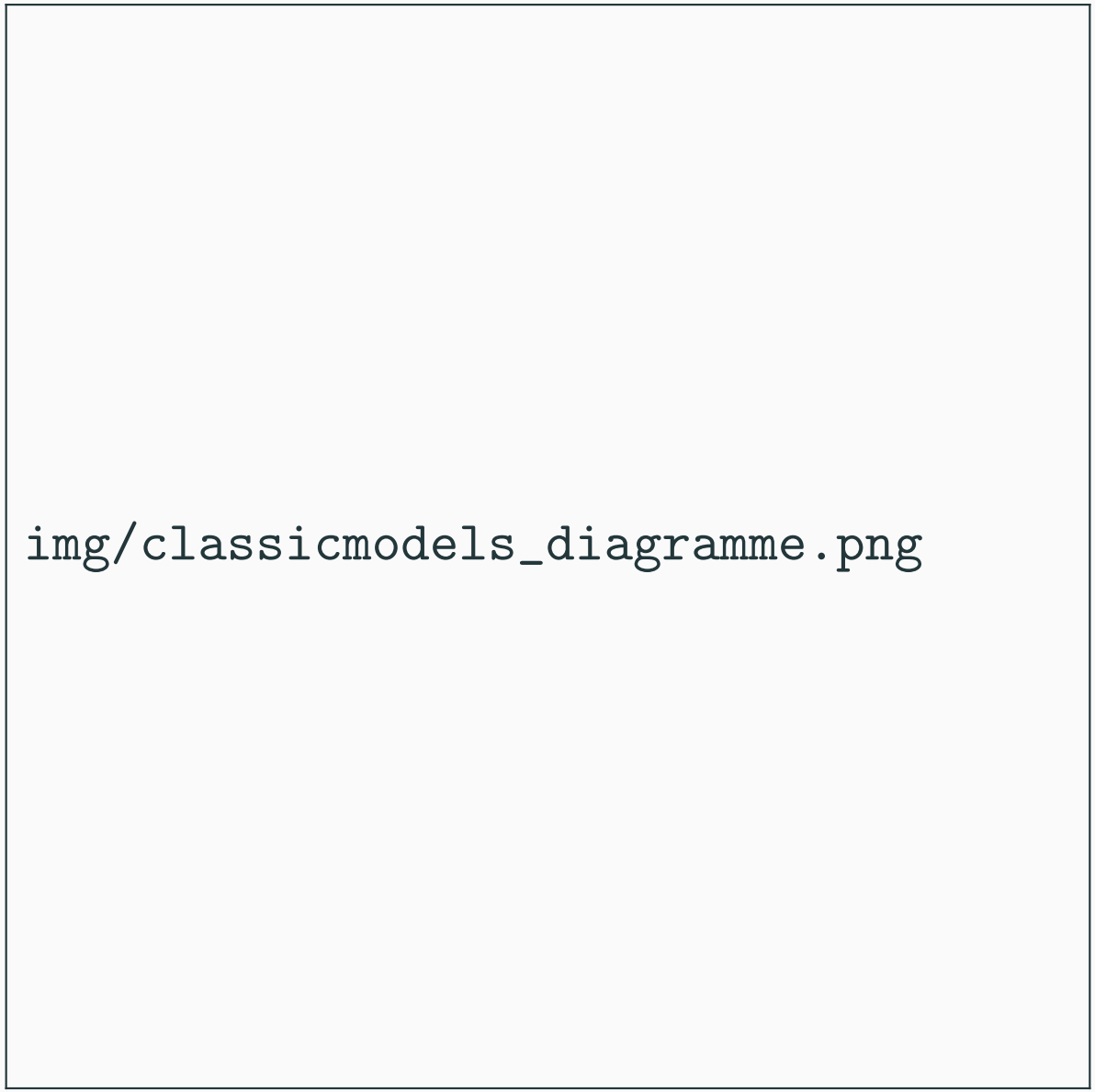


Figure 13: The classic models entity-relation diagramme

Modeling - Classic Models problem

Let's test our understanding of the model.

Try to interpret the relation between employees.

Modeling - Classic Models problem

A employee reports to zero or one employee (the manager), and a employee (the manager) get reports from zero or many employees.

Does this mean that all employees have a manager?

Modeling - Classic Models problem

Interpret the relation between employees and offices.

Modeling - Classic Models problem

Interpret the relation between employees and offices.

A employee works in one and only one office, and a office zero or many employees.

Does this mean that it is possible to have empty offices?

Modeling - Classic Models problem

Why does order details and payments entities have two primary keys?

Modeling - Classic Models problem

Why does order details and payments entities have two primary keys?

These entities don't have two primary keys. The primary keys of these entities is composed by two attributes.

For example, in the case of payments, what identifies a payment is the person that performs the payment (the primary key of the costumer) and the check number, which we assume to be unique.

Modeling - Classic Models problem

So far we have the entities, the primary keys and the relations between entities.

To write a SQL script we need to be more specific. To do so, we need to learn about constraints.

Constraints

Modeling - Constraints

Constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Modeling - Constraints

The following constraints are commonly used in SQL:

- **not null** - a column cannot have null values
- **unique** - a column cannot have duplicates
- **primary key** - a column is the primary key
- **foreign key** - a column is a foreign key
- **check** - check if a condition is true, e.g. $income \geq 0$
- **default** - defines the default value

Don't worry about the SQL scripting at this point. Keep only in mind the type of constraints we can impose on a table.

Modeling - Constraints

Note that the primary key constraint combines two types of constraints: unique and not null.

Knowing this constraints we are now able to detail the relations we have defined in our model.

Defining foreign keys

Modeling - Defining foreign keys

A few guidelines:

- If the relation has single multiplicity, add foreign key to the table with single multiplicity.
- If the relation is mandatory, add "not null" constraint to foreign key.
- If the relation is many-to-many, create an auxilary table with two one-to-many relationships.

Let's see a few examples

Modeling - Constraints

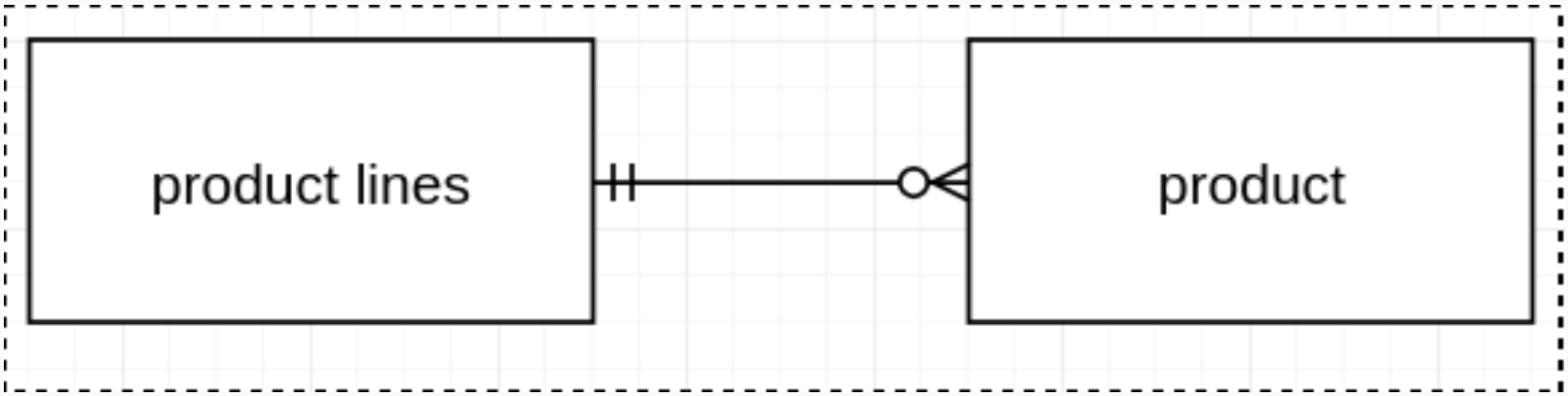


Figure 14: Here its the product that takes the foreign key pointing to a instance of product line, because the multiplicity 1 is on the side of the entity product. And because its a mandatory relation, this has to be set as not null.

Modeling - Constraints

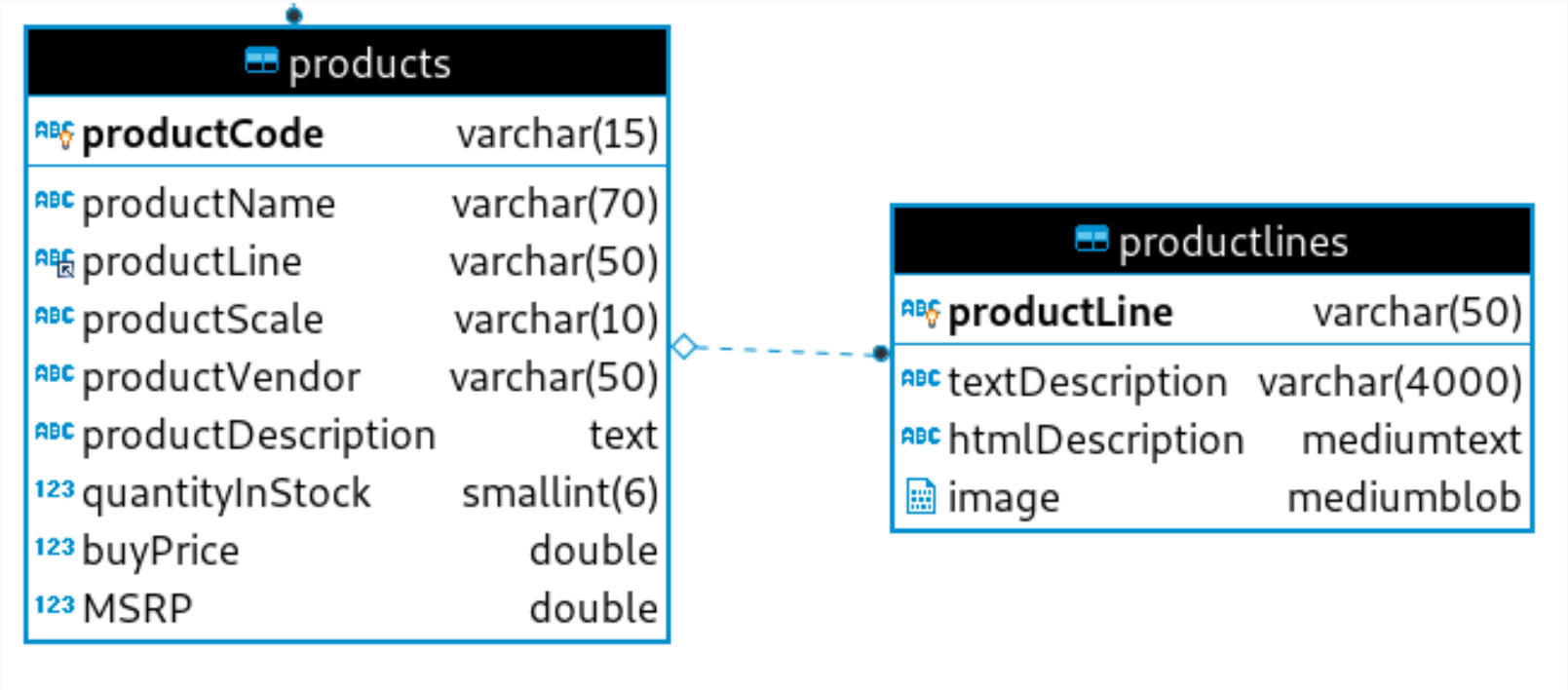


Figure 15: The dot represents multiplicity one and the diamond multiplicity many. The dashed line means the foreign keys is not of the primary key of the entity products. This is called non-identifying relationship, but is out of the scope of this course.

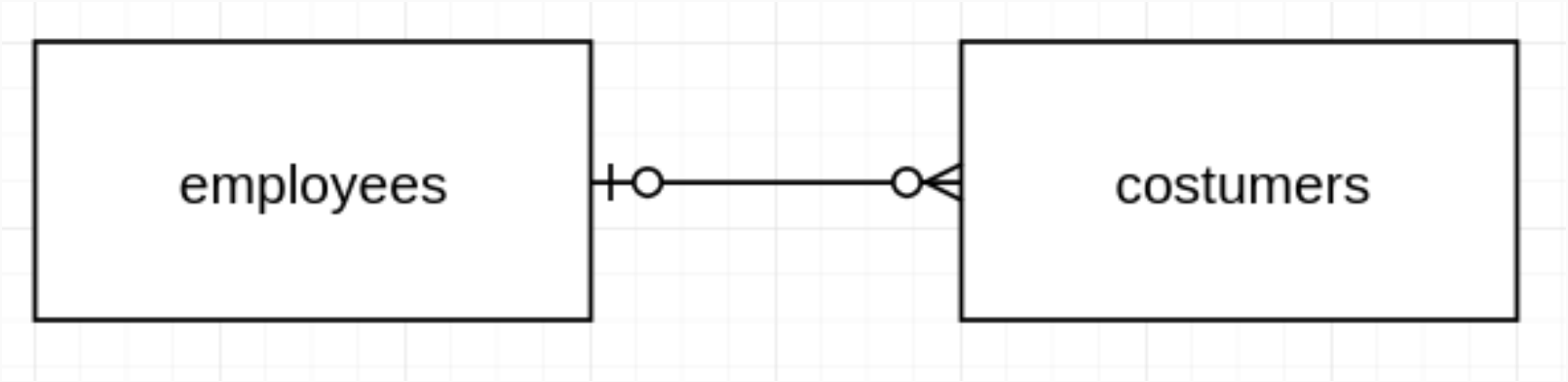


Figure 16: Here its the entity costumers that takes the foreign key. And because an optional relationship, the foreign key can be null.

Modeling - Constraints

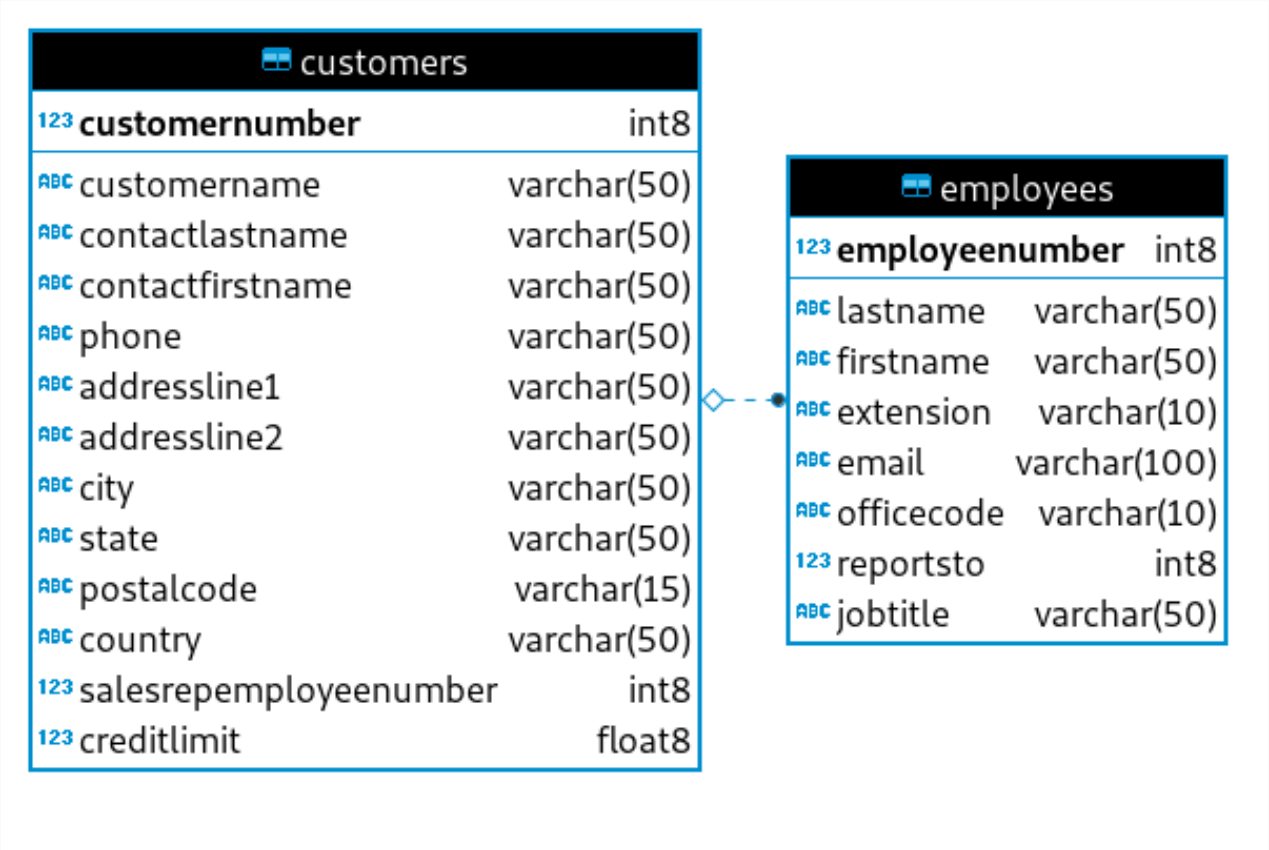


Figure 17: Note that 'salesrepemployeenumber' references 'employeenumber', the primary key of the employee entity.

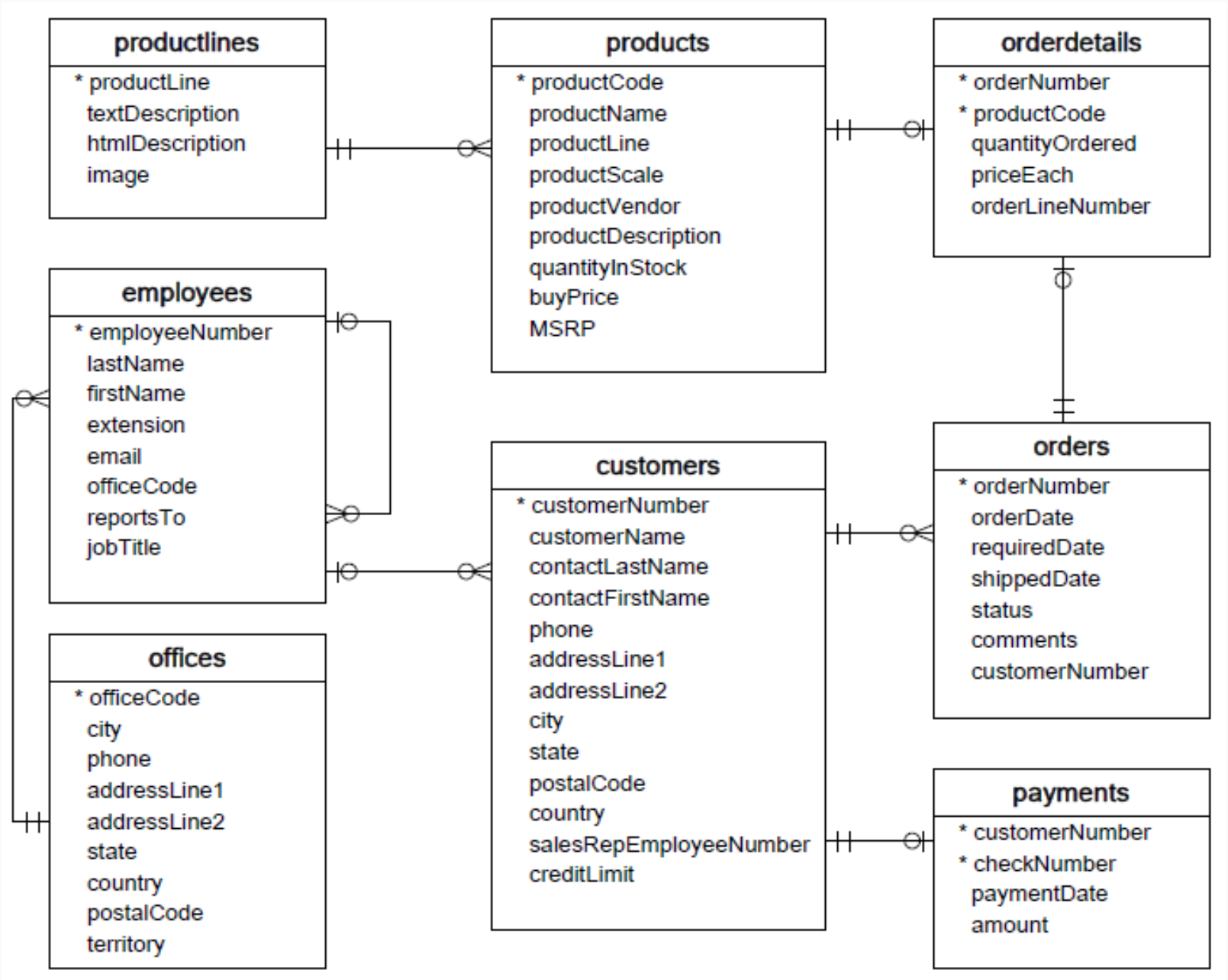
Modeling - Classic Models problem

Another thing missing is the data types in each attribute.

But that's easy.

Let's have a look at the final product, and try to understand what's going on.

Modeling - Classic Models problem



Modeling - Classic Models problem

Try to understand diagramme in figure 18

- Interpret the self relationship in employee
- Interpret the relationship between payments and customers
- Interpret the relationship between orders and customers

Now we are ready to write a SQL script that will create the database we want.

SQL - create script

SQL - create script

First, open pgadmin4, right click on your localhost server and create a new database with the name *classicmodels*.

It's in this database we'll create the tables.

SQL - create script

We'll now write the script that will create the tables and relationships.

We should start by writting the tables that are the least dependent of other tables. That is, we start with the tables that do not have foreign keys. Then we create the tables with one foreign key, etc.

So let's start with offices.

SQL - table offices

```
1  /*
2          This is a multi-line comment.
3          As you can see, postgreSQL is not case sensitve.
4          That is, This is the same of this and the same as tHis.
5  */
6  drop table if exists public.offices;
7
8  create table public.offices (
9          -- note that the primary key here is a string with 10 characters.
10         -- Is this a good idea?
11         officecode varchar(10) primary key,
12         city varchar(50) not null,
13         phone varchar(50) not null,
14         addressline1 varchar(50) not null,
15         addressline2 varchar(50) default null,
16         state varchar(50) default null,
17         country varchar(50) not null,
18         postalcode varchar(15) not null,
19         territory varchar(10) not null
20 );
```

SQL - table employees

```
1  drop table if exists public.employees;
2
3  create table public.employees (
4          employeenumber bigint primary key,
5          lastname character varying(50) not null,
6          firstname character varying(50) not null,
7          extension character varying(10) not null,
8          email character varying(100) not null,
9          officecode character varying(10) not null
10                 references public.offices(officecode)
11                 on delete cascade,
12         -- This is how we create a foreign key, but not the only way.
13         -- There is a way using the alter table command,
14         -- but this is a compact way.
15         reportsto bigint references
16                 public.employees(employeenumber)
17                 on delete cascade,
18         jobtitle character varying(50) not null
19 );
```

SQL - table customers

```sql
drop table if exists public.customers;

create table public.customers (
    customernumber bigint primary key,
    customername varchar(50) not null,
    contactlastname varchar(50) not null,
    contactfirstname varchar(50) not null,
    phone varchar(50) not null,
    addressline1 varchar(50) not null,
    addressline2 varchar(50) default null,
    city varchar(50) not null,
    state varchar(50) default null,
    postalcode varchar(15) default null,
    country varchar(50) not null,
    salesrepemployeenumber bigint
                references public.employees(employeenumber)
                on delete cascade,
        -- can the credit be negative? no
    creditlimit double precision check(creditlimit >= 0)
);
```

SQL - table payments

```sql
drop table if exists public.payments;

create table public.payments (
        customernumber bigint not null
                references public.customers(customernumber)
                on delete cascade,
        checknumber character varying(50) not null,
        paymentdate date not null,
        amount double precision not null,
        -- this is a way to define a composite primary key
        primary key(customernumber, checknumber)
);
```

SQL - table product lines

```
1  drop table if exists public.productlines;
2
3  create table public.productlines (
4      productline varchar(50) primary key,
5          -- an alternative is to change the data type to 'text';
6          -- text data type is practically ilimitated.
7      textdescription varchar(4000) default null,
8      htmldescription text,
9          -- an alternative to this is to save the path in the server
10         -- where the image is saved; we save space this way
11     image bytea
12 );
```

SQL - table products

```sql
drop table if exists public.products;

create table public.products (
    productcode varchar(15) primary key,
    productname varchar(70) not null,
    productline varchar(50) not null
                references public.productlines(productline)
                on delete cascade,
    productscale varchar(10) not null,
    productvendor varchar(50) not null,
    productdescription text not null,
    quantityinstock smallint not null,
    buyprice double precision not null,
    msrp double precision not null
);
```

SQL - table orders

```
1  drop table if exists public.orders;
2
3  create table public.orders (
4          ordernumber bigint primary key,
5          orderdate date not null,
6          requireddate date not null,
7          shippeddate date,
8          status varchar(15) not null,
9          comments text,
10         customernumber bigint not null
11                 references public.customers(customernumber)
12                 on delete cascade
13 );
```

SQL - table orderdetails

```
1  drop table if exists public.orderdetails;
2
3  create table public.orderdetails (
4      ordernumber bigint
5                  references public.orders(ordernumber)
6                  on delete cascade,
7      productcode varchar(15)
8                  references public.products(productcode)
9                  on delete cascade,
10     quantityordered bigint not null,
11     priceeach double precision not null,
12     orderlinenumber smallint not null,
13         primary key(ordernumber, productcode)
14 );
```

SQL - populate the tables

Typically once the database is set, we can populate the data base either gradually using a dedicated system or in block. We'll take the second route.

We'll insert data directly in the database. Because it's quite time consuming to write an script to insert all the data we need, I have provided the script already prepared to run.

Go to moodle and download the script *"classicmodels_insert.sql"*.

SQL - populate the tables

In pgadmin4, select the classicmodels database and then click in *Tools* > *Query Tool*.

Open the downloaded script. You will see lots of insert commands.

Select all, and run the script. If the database is correct, you should have no error.

Final section

The final section

We've covered alot of terrain and some of the concepts maybe a bit fuzzy at this point. However, we'll have the chance to practice and you'll have the change also to apply then in your project.

But you may ask "Where is the spatial component in all this database thing?"

A geometry is just an attribute in a relational database, that is saved in byte format. All these details will be clear next week. At this point, what's important is that you have all these concepts in mind.

Next class we'll be talking about the select query.