cphbusiness

COPENHAGEN BUSINESS ACADEMY

- From Business modeling to database design

Today's Topics

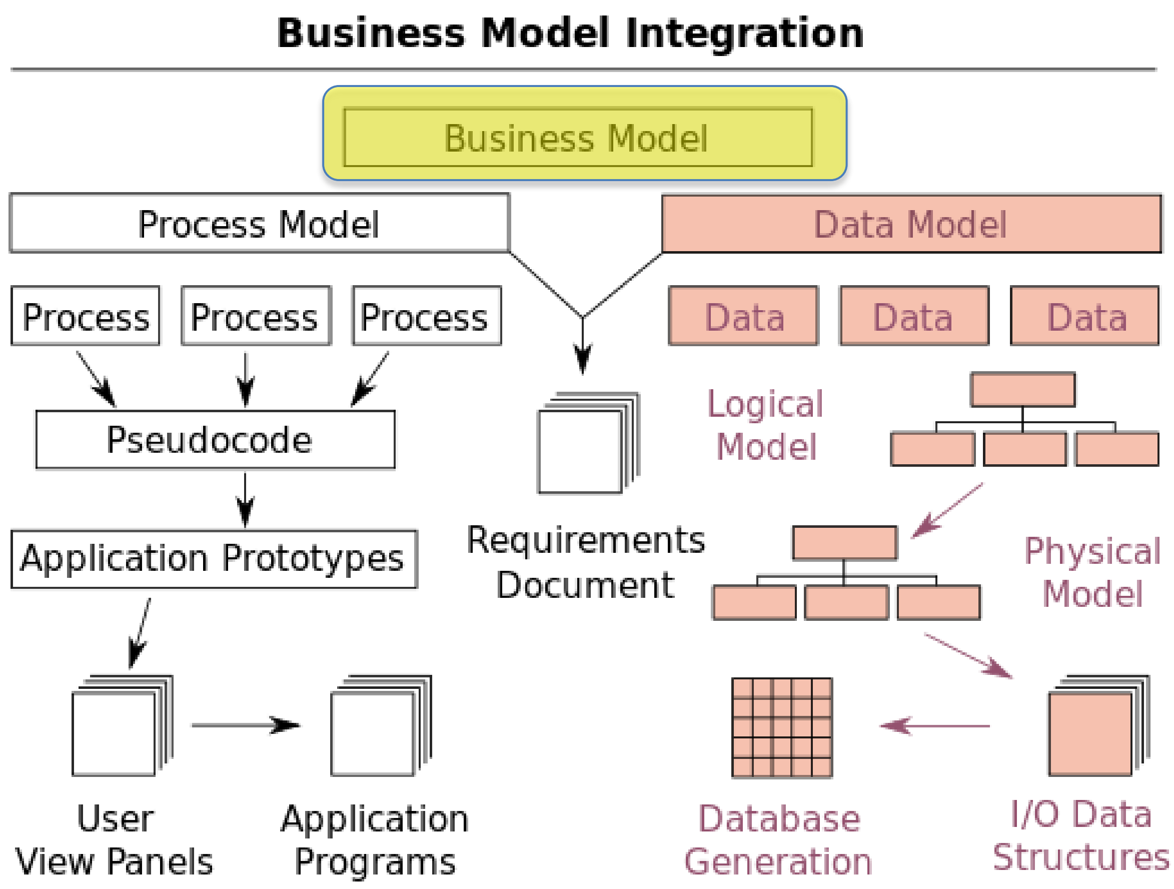
- We will look at different types of **data models** and how logical data models are transformed into physical data models.
- Focus will be on design **principles** and central **properties** for database tables.
- You will learn how to **forward engineer** a database from graphical database modeling in MySQL Workbench.

Database design – it starts with business modeling

Let's go back to your January activity with Palle for a moment …

Domain model

Use cases (user stories)



E/R model

Relational model

Tables / SQL

Data models at several levels

Data models transform from logical into physical form:

- **Conceptual model**
  - Presentation of data in end user terminology and context
  - Technology free
  - Domain model (business focus), **E/R model** (persistence focus)

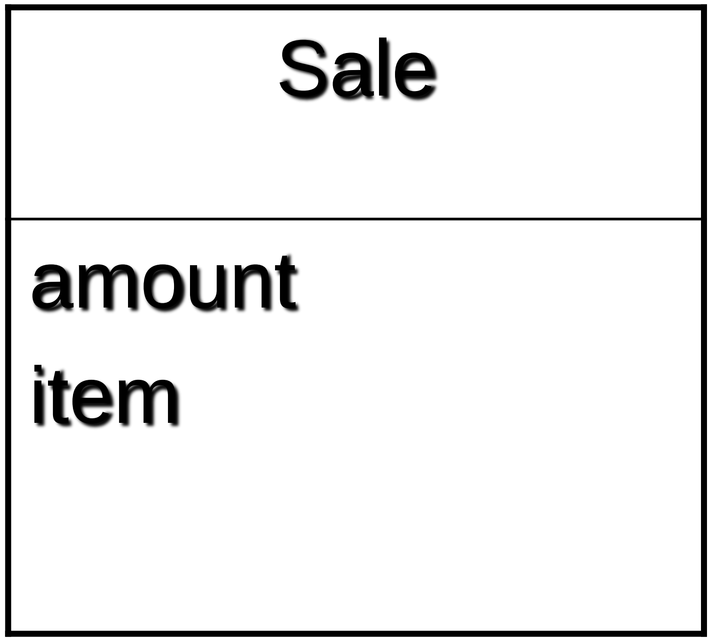
- **Logical model**
  - Can partly be understood by end-users
  - Relies on technology, but doesn't show it directly
  - **Relational model**, network, hierarchical, OO models

Physical model
  - Close to the physical representation of the database
  - Technology specific, e.g. **MySQL (SQL scripts)**
  - Disk usage, security, etc.

From Palle's slides

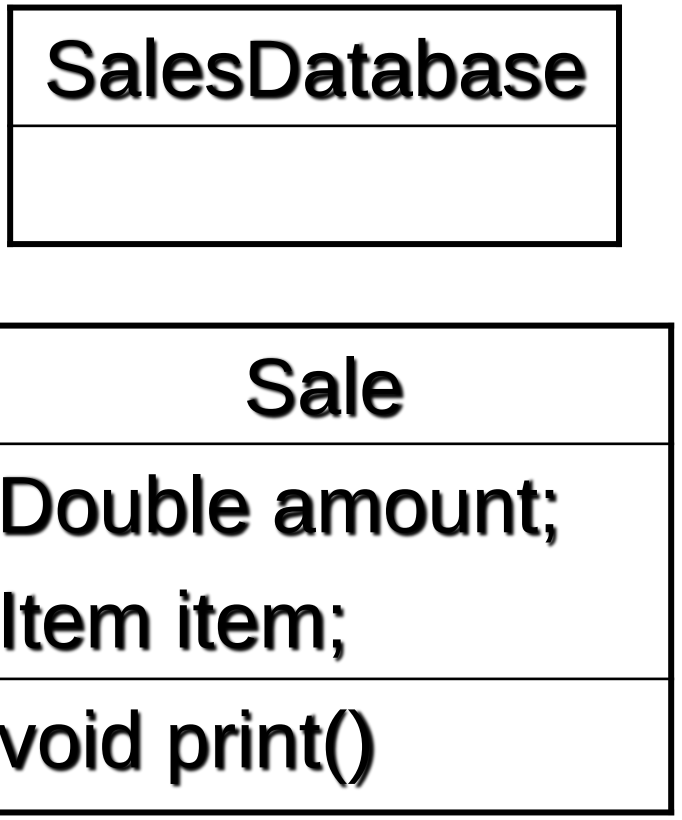
A Domain Model is Conceptual, not a Software Artifact

Conceptual Class:



vs.

Software Artifacts:



From Palle's slides

Let's try!

- This school has many students, distributed into classes, each student goes in one class. Each class has many hours in various subjects, and these subjects are held in many different rooms. This school has several teachers who teach more classes ………

RED - possible domain classes
GREEN – associations between classes
PURPLE – could tell about multiplicity

Exercise

Think of your local Pizza shop.

They offer a lot of products, chosen from a menu. Typically, you can then select additional accessories, eg. extra cheese, extra ham, pineapples or other.

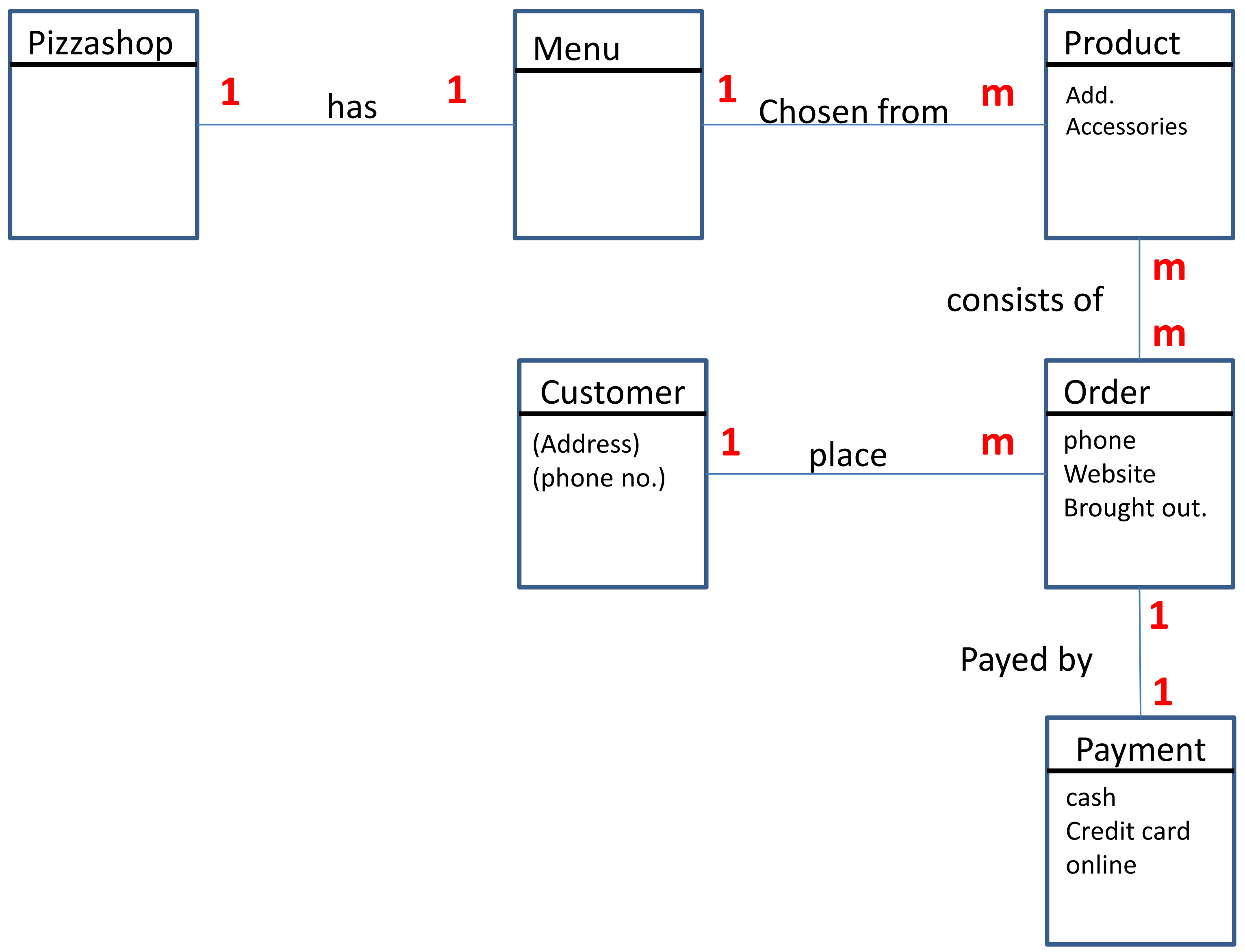
Ordering is done either by phone or on a website.

The customer can select the food picked up or to be brought out. Payment is made either online when ordering or with Credit Card or cash on pickup.

Create a domain model

From Palle's slides

Domain Model Pizzashop



Database design

Later this semester, you will work in project groups of 4-5 students.

Make a suggestion of

- which database tables you will need to represent the project groups
- which columns these tables should contain

Demo –MySQL Workbench modeling

Forward engineering:

Start with model -> use tool to **generate** database

Instructions at:

https://dev.mysql.com/doc/workbench/en/wb-getting-started-tutorial-creating-a-model.html

Naming Principles

Table names are like class names:

- Upper case first letter
- **Singular**
- no_ - use LargeandSmallLetters

Column names are:

- Lower case first letter
- **Singular**
- no_ - use smallAndLargeLetters

Simple Data Types

Numeric data types :

- INTEGER, **INT**, SMALLINT, TINYINT, MEDIUMINT, BIGINT
- FLOAT, **DOUBLE**

Number in parenthesis “INT(11)” indicates display size when printed in Workbench

Text comes in two fundamental types:

- CHAR and VARCHAR. **Use VARCHAR!**

Number in parenthesis “VARCHAR(20)” indicates maximum length– if nothing is specified it can be up till 65535 characters.

- VARCHAR uses storage according to concrete content
- CHAR uses storage as indicated in parenthesis and can be up till 255 characters

Date Types

The **DATE** type is used for values with a date part, but no time part. MySQL retrieves and displays DATE values in 'YYYY-MM-DD' format.

The supported range is '1000-01-01' to '9999-12-31'.

The **DATETIME** type is used for values that contain both date and time parts. MySQL retrieves and displays DATETIME values in 'YYYY-MM-DD HH:MM:SS' format.

The supported range is '1000-01-01 00:00:00' to '9999-12-31 23:59:59'.

The **TIMESTAMP** data type is used for values that contain both date and time parts.

TIMESTAMP has a range of '1970-01-01 00:00:01' UTC to '2038-01-19 03:14:07' UTC

Unique Columns

If you declare a column unique, there cannot be two rows with the same value

- Emails and phone numbers are often unique

Notice– even though a column is declared as unique there can be null values in several rows

Not Null & Default Value

A column can be declared to never to contain any null values in any rows

You can specify a default value for a column

You can combine not null with a default value

Key

A key is a unique column that cannot be null

The databasecreates an **index** for keys which make it much faster to retrieve a row based on its key.

Example:

```
select *
from student
where email= “cph-rg54@cphbusiness.dk”
```

Sometimes a key is composite

Example:

```
CREATE TABLE orderdetails (
  orderNumber int(11) NOT NULL,
  productCode varchar(15) NOT NULL,
  quantityOrdered int(11) NOT NULL,
  priceEach decimal(10,2) NOT NULL,
  orderLineNumber smallint(6) NOT NULL,
  PRIMARY KEY (orderNumber,productCode)
)
```

Primary Key

A primary key is a key chosen for identification of rows in a table

You should not change a primary key over time.

This means that phone numbers and email adresses are poor primary keys

Foreign Key

A foreign key is a column that has a value which exists as primary keys in a (other) table

Example: The **office** table in the classicmodels database has primary key “officeCode”. In the table **employees**, there is a column which represents the office that the employee belongs to. This column is a foreign key.

Notice: Since more employees can belong to the same office, the foreign key is not unique in the **employees** table. Also, freelancers do not belong to an office and therefore the foreign key might also be null.

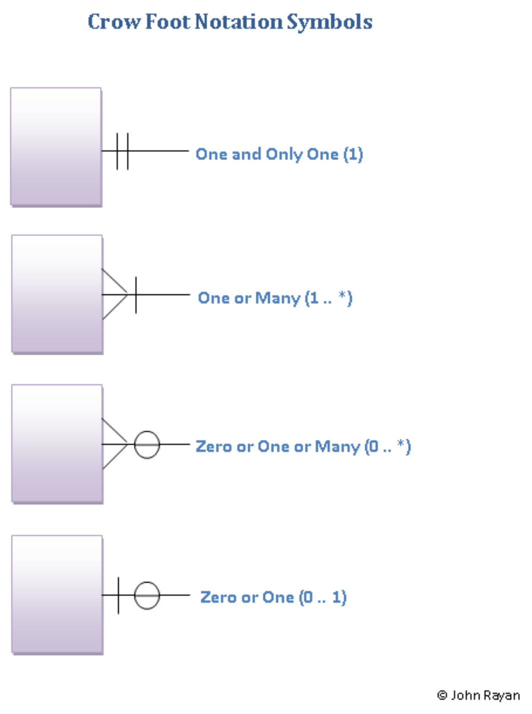
Relationship Between Tables

Primary and foreign keys are used to make **relations** between tables
There are different types of relationships:

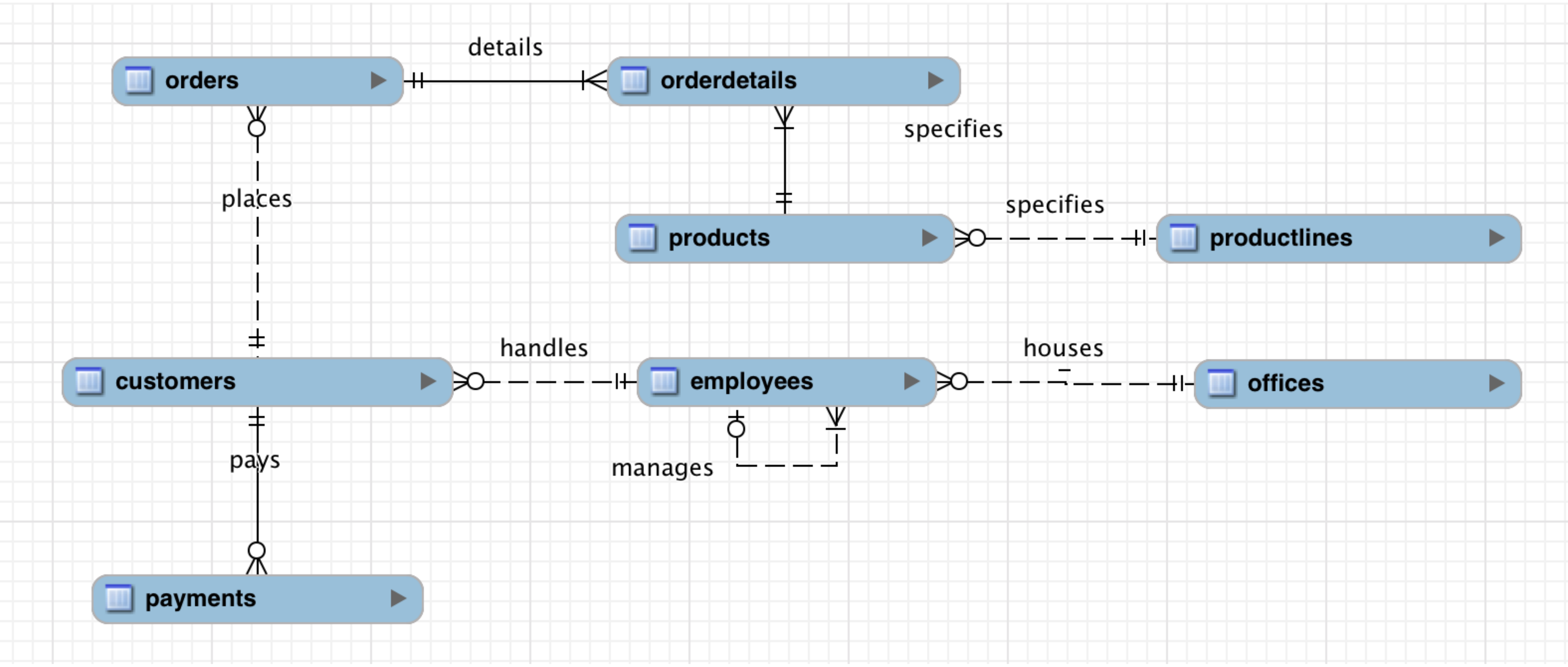
- Multiplicity / Cardinality

– 1-1	Example - marriage
– 1-Many	Example – ownership (**most frequent relation**)
– Many-Many	Example – project participation

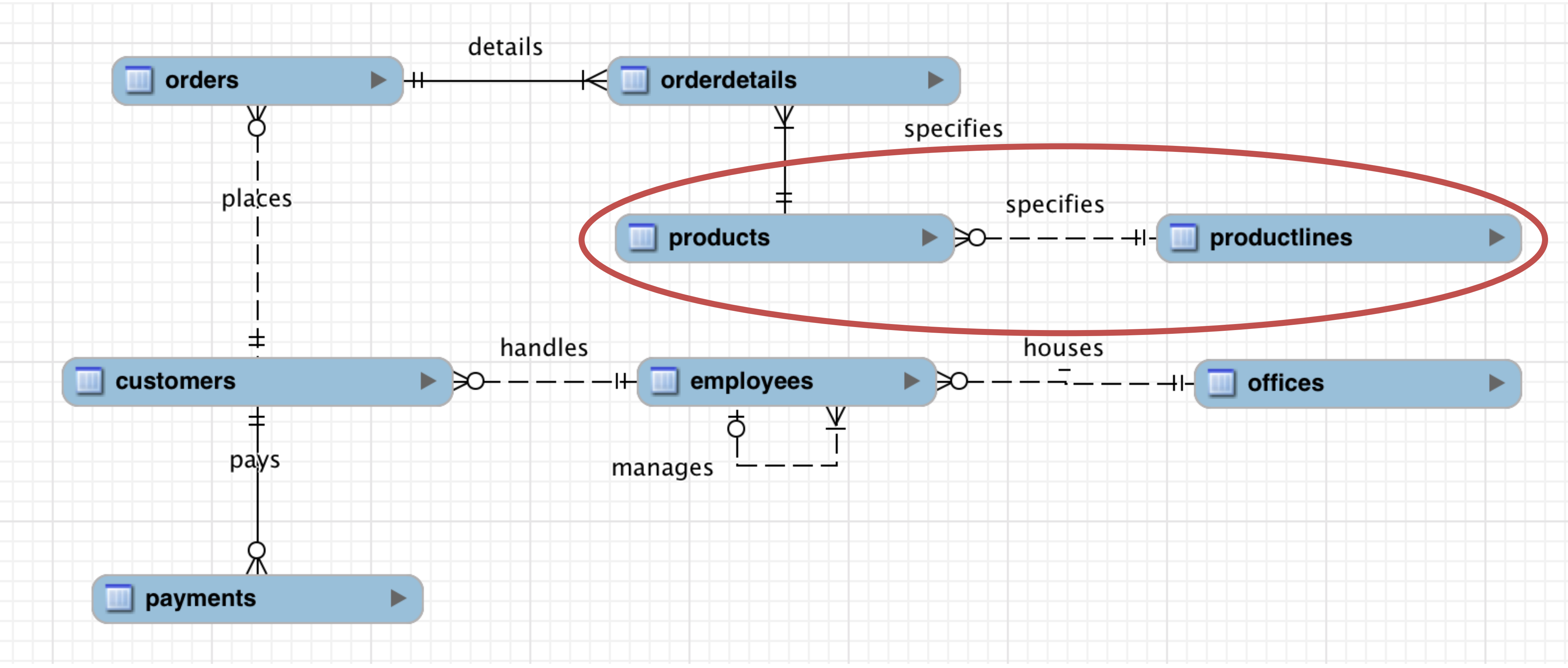
- Modality / Mandatory or optional
  - Can any of the”1” above be 0?



Relationship - 1



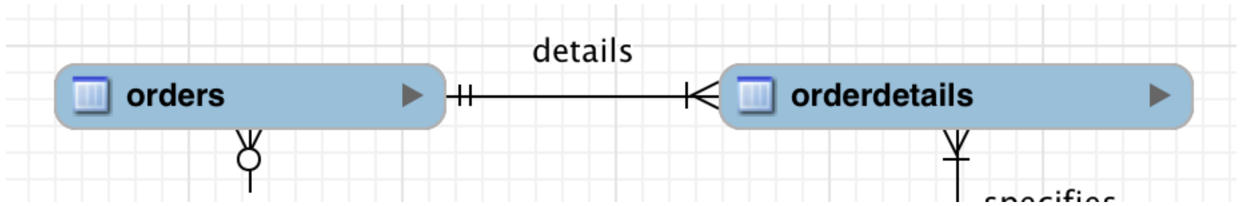
Relationship - 2



0-M relation reads:
Each product is specified by a product line
Each product line specifies more product (or none)

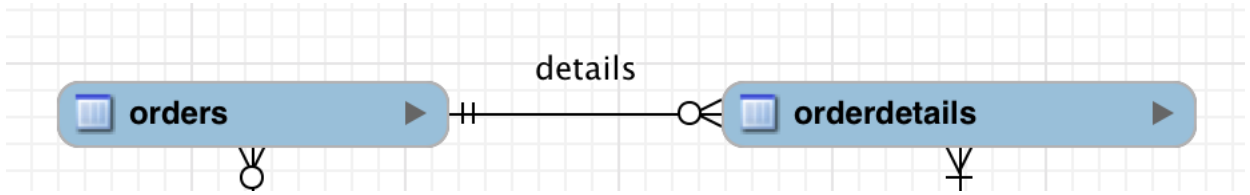
Relationship 3

Watch out for 1-M relationship compared to 0-M relationship when one is a weak entity



You need to create at least one detail when the order is created. Often the order is created first, and details afterwards .

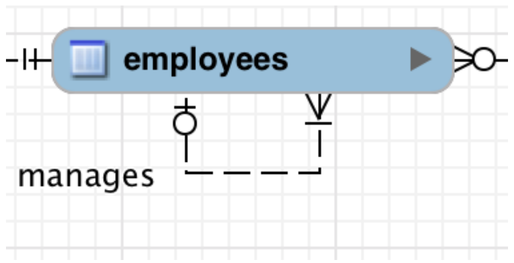
You could instead use this model instead in the database:



And let the application make sure that all orders have at least one order detail (transaction handling).

Relationship 4

In the model a relationship refers to the same table:



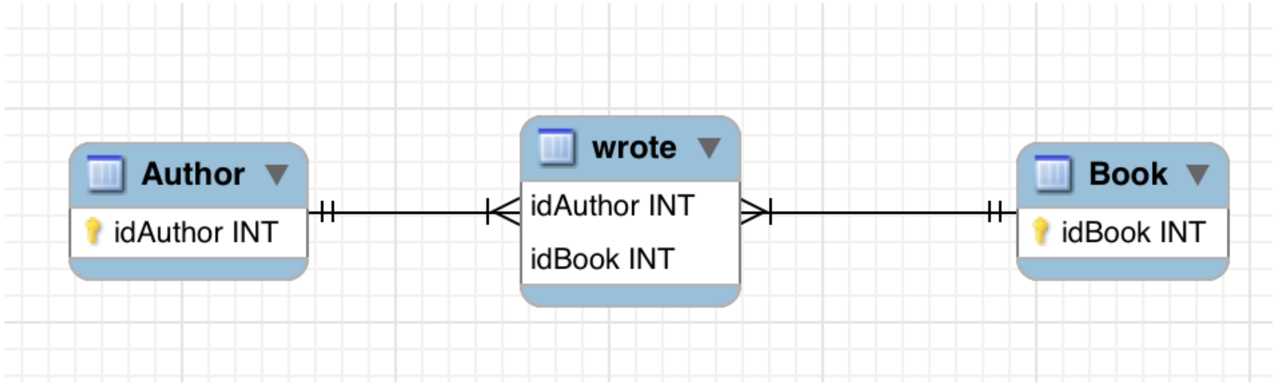
Is this correctly modelled?

Many to Many Relationship

A library database has a table for authors and a table for books.

An author can write many books and a book can be written by many authors.

The relationship “wrote” is a many-to-many multiplicity, but it cannot be done without an extra mapping table in the database:



Database Design Exercise (work in pairs)

Make a model of the library as specified by the file ”library-system.pdf”

The model must be drawn in MySQL workbench. and hereafter created as a database.

Insert a borrower and a few materials into the database. Let the borrower borrow one material and let him/her make two reservations.

Make a SQL statement that can show a borrower’s name and the material title of all his/hers reservations

Extra for Red students: Reverse Engineering Exercise

Diagrams can be made based on an existing database.

Useful when:

- You don’t have the model already
- You have changed the database, but not the model

Be careful with 0/1 – M relationships – it is not certain that it is done properly when you reverse engineer a database into a model

Exercise

- Make a reverse engineering of classicmodels database
  - See: https://dev.mysql.com/doc/workbench/en/wb-reverse-engineer-live.html
- Adjust the diagram so it makes the most sense (= because a useful artifact)