Database Course Summary Review

CS-A1153: Databases

Prof. Nitin Sawhney and Dr. Lukas Ahrenberg

Aalto University
School of Science

Unified Modeling Language (UML) Review

CS-A1153: Databases

Prof. Nitin Sawhney

Aalto University
School of Science

Final Exam: September 3, 2020, 09:00-12:00

Expected to Understand:

- Most concepts introduced in the lectures and textbook (chapters 4.9-4.10, 5.3-5.6, 8.5, and 9-12).
- Constructing UML Diagrams with Relations & Attributes.
- Writing SQL Queries for a Database Schema.
- Functional Dependencies and Decomposition into BCNF.
- Transactions and ACID properties of transactions.
- Principle of Triggers and the distinction between row level and statement level triggers.
- Estimate which Indexes the database should have.

Not Expected to demonstrate in exam:

- How to decompose relations to Fourth Normal Form.
- Writing embedded SQL or using SQL queries in Python.
- How to write triggers.

Review

- Creating a UML Diagram
  - Class and Attributes
  - Associations and Multiplicity
  - Self-Associations and Association Class
  - Attributes as Keys
- Converting a UML diagram into Relations

UML Modeling

- How can one start UML modeling?

- Simple, but often a good working approach:
  1. Write a description of the database being modeled.
  2. Underline all the nouns.
  3. From nouns, find candidates for classes and attributes.
  4. Some nouns won't become either.
  5. When the classes and attributes are done, think, what kind of relations there might be between the objects of the classes. Make them the associations.

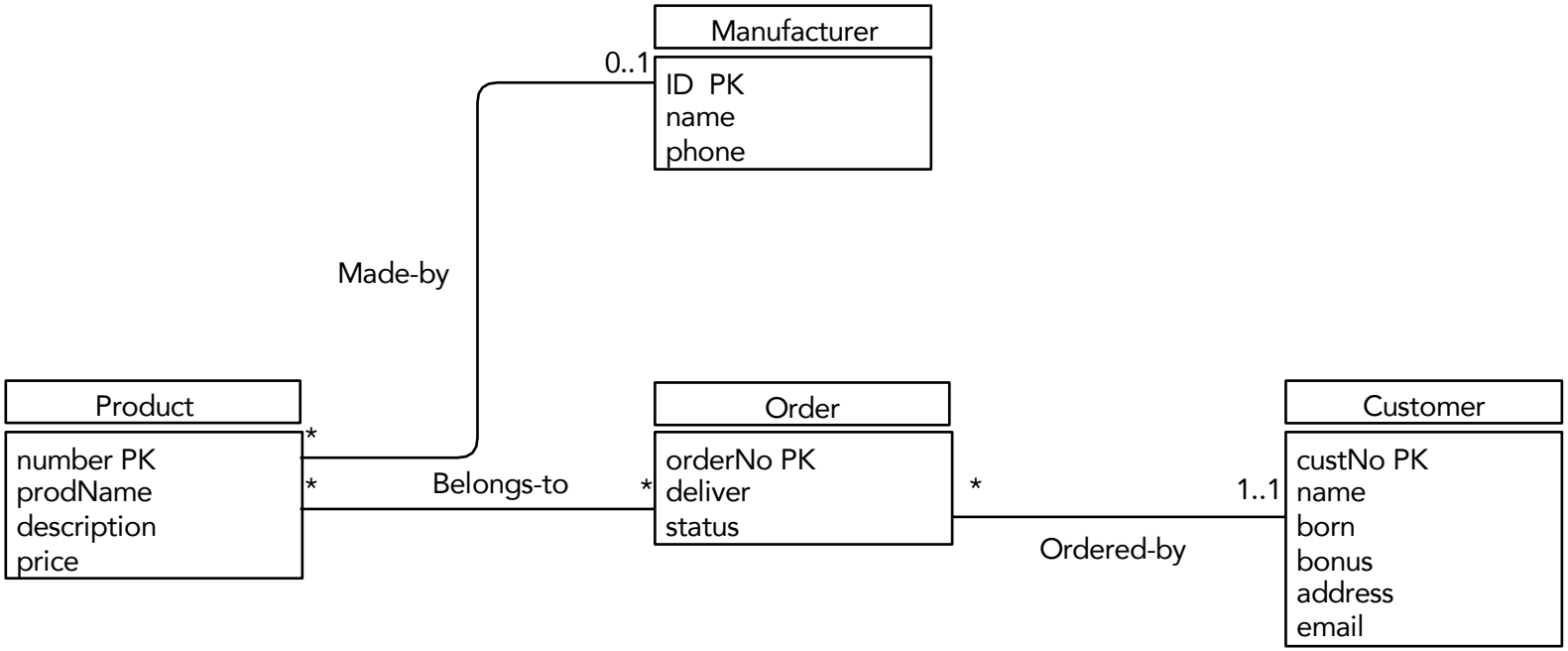
Example: Description of a Web Store

- Create a database for a web store that has products and customers. Customers can make orders which can include multiple products. Products have product number, name, description, price and manufacturer. Manufacturers have ID, name and phone number. Customers have customer ID, name, year of birth, bonus points, address and email address. Every order has unique order number. Orders also have shipping method, state, products included in the order and the customer who made the order.

Example: underline classes and attributes

- Create a database for a web store that has products and customers. Customers can make orders which can include multiple products. Products have product number, name, description, price and manufacturer. Manufacturers have ID, name and phone number. Customers have customer ID, name, year of birth, bonus points, address and email address. Every order has unique order number. Orders also have shipping method, state, products included in the order and the customer who made the order.

Creating a UML Diagram for the Web Store



Associations and Multiplicity

- Associations create a connection between two classes (no more than two in UML diagrams).

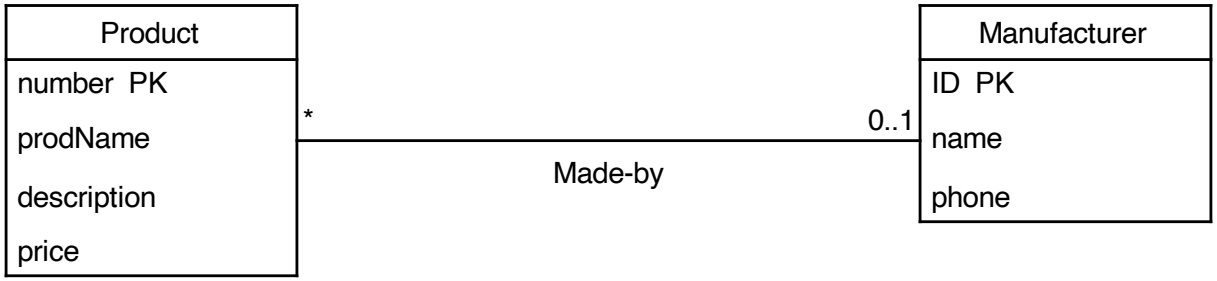
- In this example, there can be 0 or 1 *Manufacturer*-objects for each *Product*-object.
- Each *Manufacturer*-object can connect to arbitrarily many *Product*-objects.
- **Pay attention to which ends the labels are on!**

Multiplicity of Association: Precise Notations

- In an association between classes *A* and *B*, above or below the line, the label m..n **at class *B*'s end** means, that for each object in the class *A* we have at least *m* and at most *n* objects of class *B* connected to it.
- Examples:
  - *0..1* at most 1, but possibly none
  - *0..5* at most 5, but possibly none
  - *1..2* at least 1, at most 2
  - *1..1* exactly 1 (can be labeled with *1*)
  - *1..** at least one 1, but otherwise arbitrarily many
  - *0..** arbitrary amount (can be labeled simply with *).
- When it comes to database modeling, a missing label is equivalent to 1..1.

Marking the Attributes as Keys

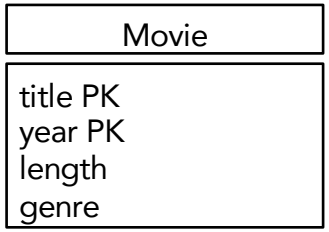
- A product is identified by its product number (attribute *number*), and a manufacturer by its ID (attribute *ID*).



Another Example of Keys

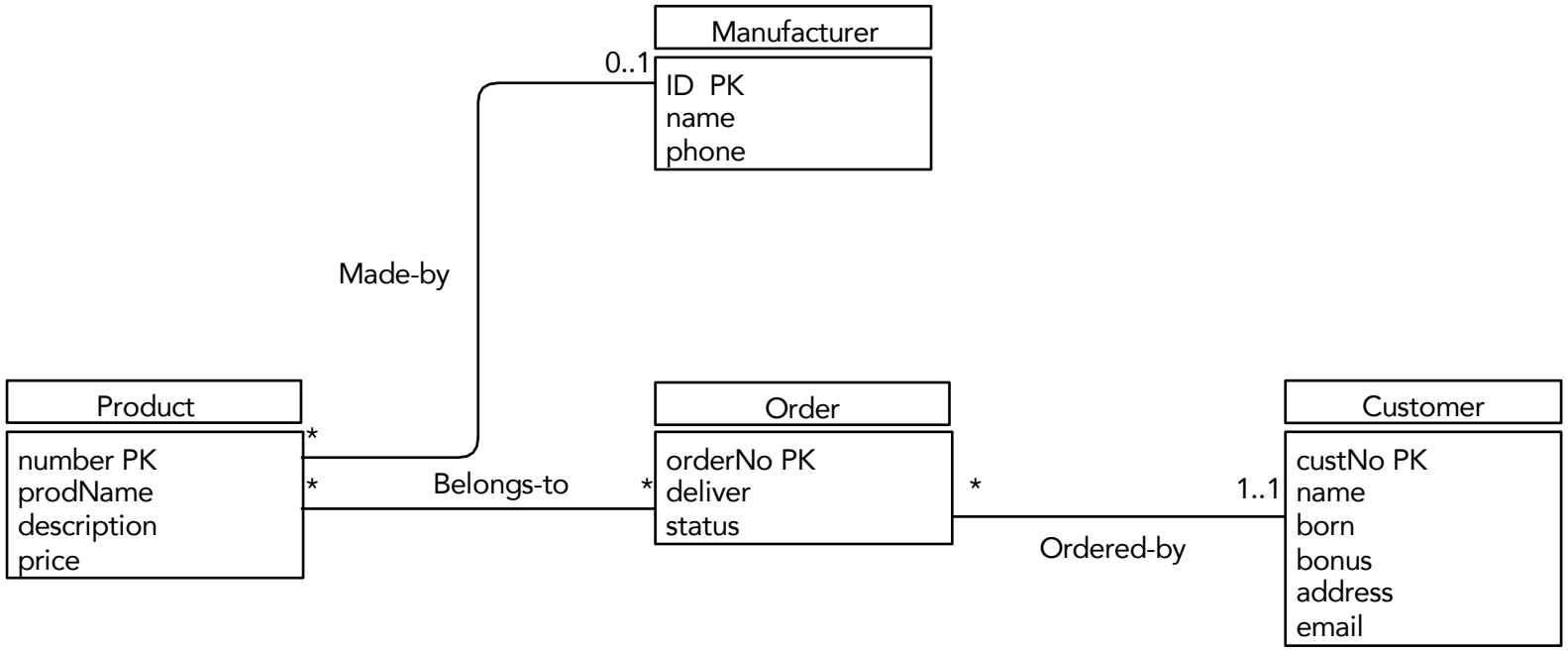
- In the textbook, the keys for the class *Movie* are the name of the movie and the year, together.

The idea is, that no film studio wants to produce a movie with the same name as another movie from a competing studio in the same year. However, it's possible that later someone wants to create a new version of the movie with the same name.



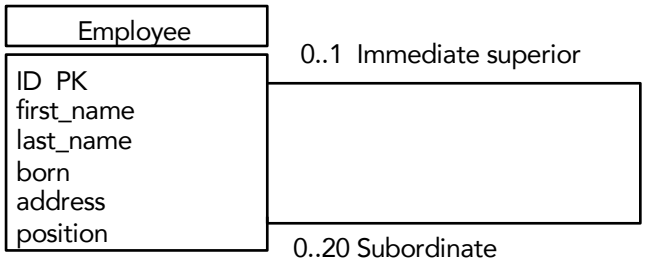
Example Diagram

- This model allows us to add products with no manufacturer in the database. On the contrary we can't add orders with no customer.



Self-Association

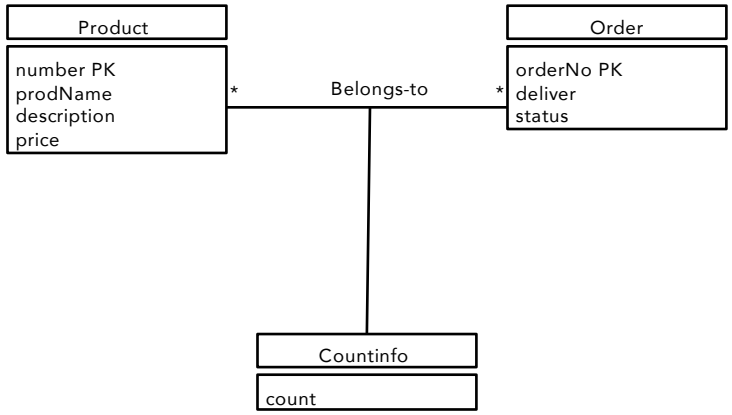
- *Problem:* How can we represent associations where the same class occurs twice?
- Example: let's define, that one employee can be an immediate superior for another employee.



- Both roles are written in the diagram. The multiplicity of the association is indicated at the end corresponding to the roles (here one employee can have at most one immediate superior, and 0–20 subordinates).

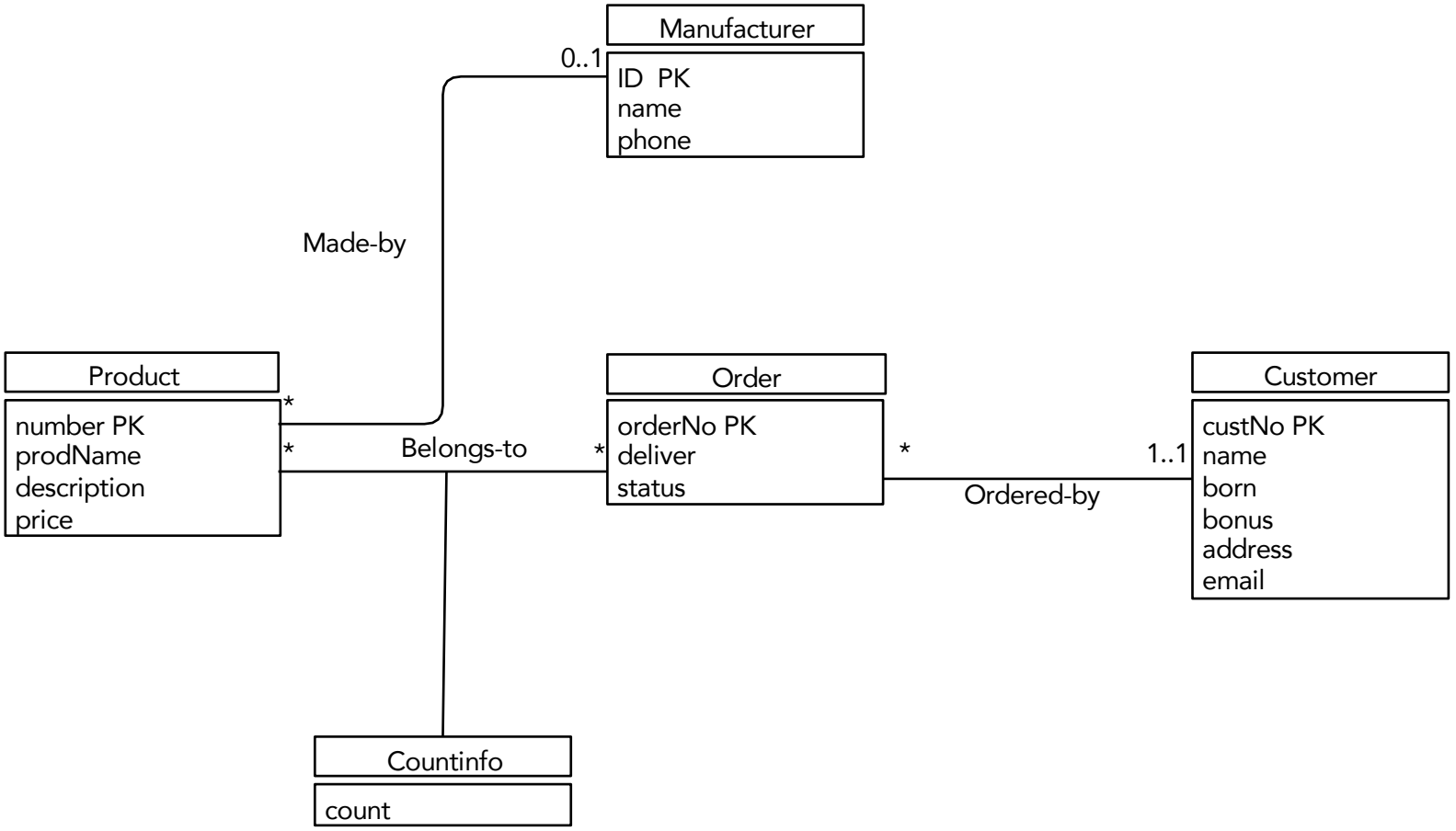
Association Class

- Example: a customer can add many items of the same product to one order.
- Where should information about number of items be attached?
- Add to association *Belongs-to* an association class, and to it's attribute we'll add information about number of items.



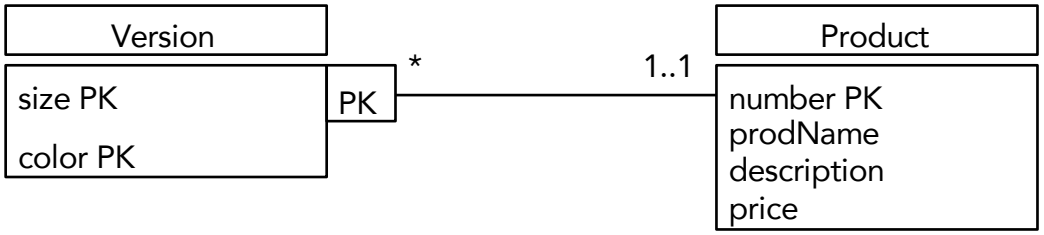
- An association class has no key attributes and multiplicity is never indicated on the line leading to the association class.

Web Store Example with Association Class



When Attributes are not enough for the Key

- Assume that products can have different versions, e.g. the same clothing in different colors and sizes.
- Define a class *Version* with size and color as attributes. Even together they are not enough to uniquely identify an object of *Version* (therefore they don't form a key).
- Solution: form a key of class *Version* by using attributes of class *Version* and the key attribute of class *Product*. The class *Version* is marked with only it's own attributes.



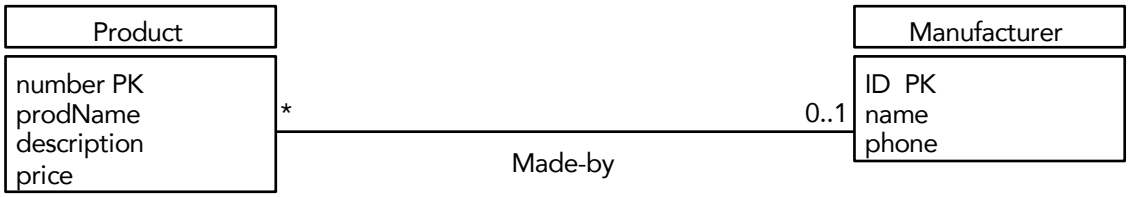
Design Principles

What should be considered when making a UML model of a real-life system?

- Faithfulness
- Avoiding Redundancy
- Simplicity Counts
- Choosing the Right Relationships
- Picking the Right Kind of Element

Using the Right Elements

- When should some element be described with a class or with an attribute?
- Let's examine the model below. Could class *Manufacturer* and association *Made-by* be replaced by adding all its attributes to the class *Products*?



- But then the name and phone number of a certain manufacturer would be repeated in different products.

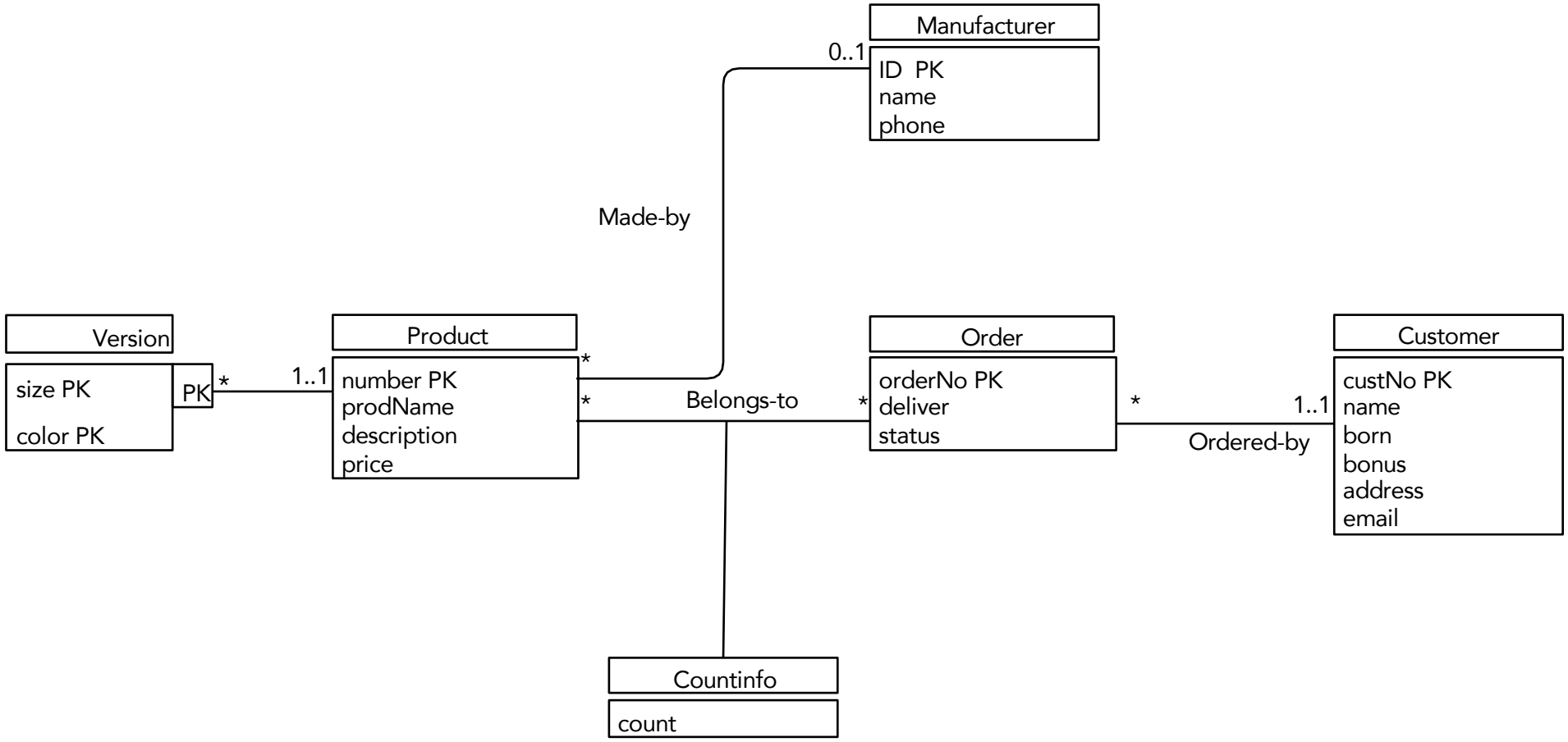
Using the Right Elements, Continued

- When to use a regular class vs. an association class? Association class is good for situations where the information being described is related exactly to a pair formed by two objects and it does not have its own key.

- In the web store example *Order* couldn't be an association class, but is has to be regular class, because:
  1. Order is not necessarily a relation between exactly one product and one customer, but one order can include several products.
  2. The class *Order* has it's own key attribute.

Only one of these reasonings would be enough.

Complete Web Store Example



Converting UML Diagram to Database Schema

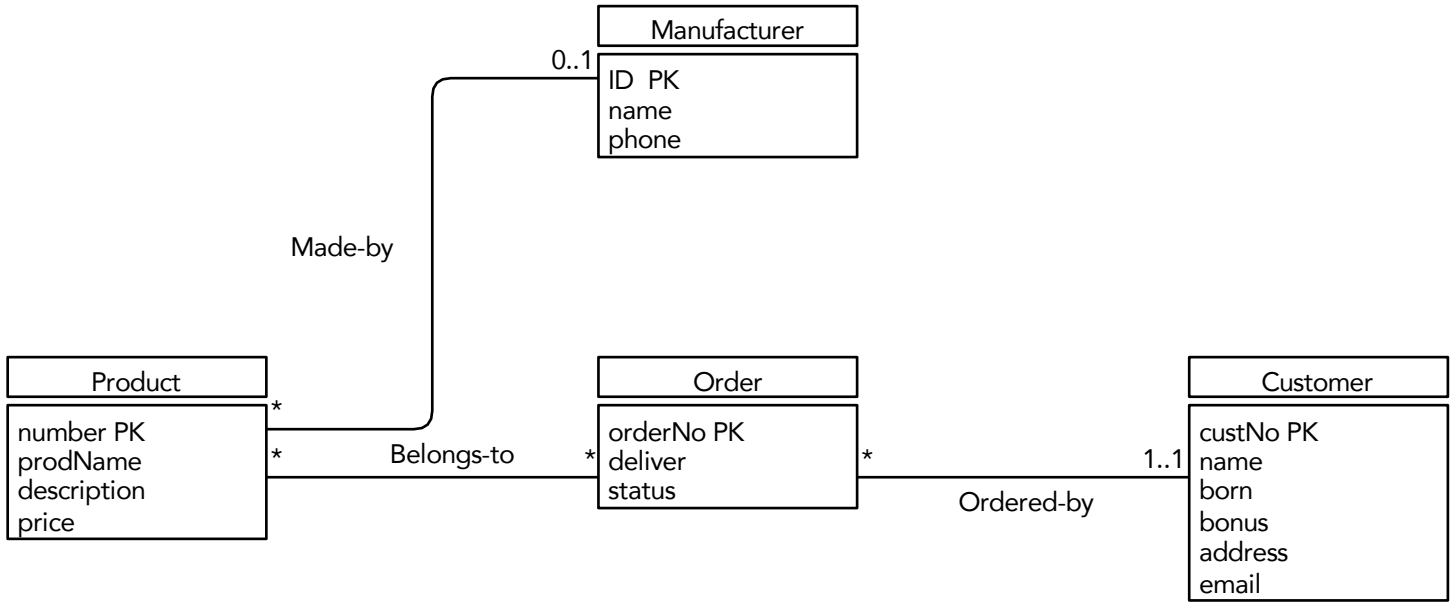
Basics:

- Each class gets its own relation, which has the same attributes as the class.
- For each many-many association we create a relation, whose attributes are the keys from both of the relations the association connects, and the attributes of the association class, if there is one.
- For each many-one association we can either create a relation in the same manner as with many-many associations or we can add attributes to the relation that are on the many-side of the association.
- Rename the attributes in the relations, if needed, to make them clear and distinct.

Example: Defining Relations for UML Classes

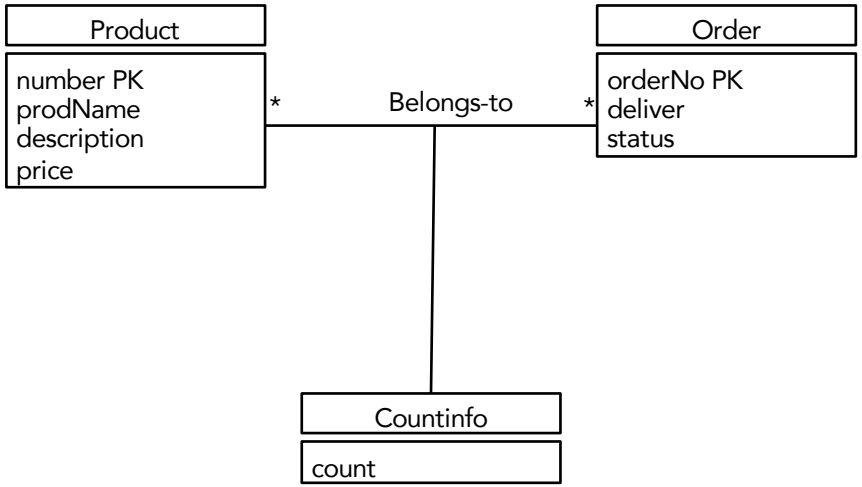
- For the classses in the UML diagram of the online store we can define relations:

  Customers(custNo, name, born, bonus, address, email)
  Products(number, name, description, price)
  Manufacturers(ID, name, phone)
  Orders(orderNo, deliver, status)



Example: Defining Relations for Associations

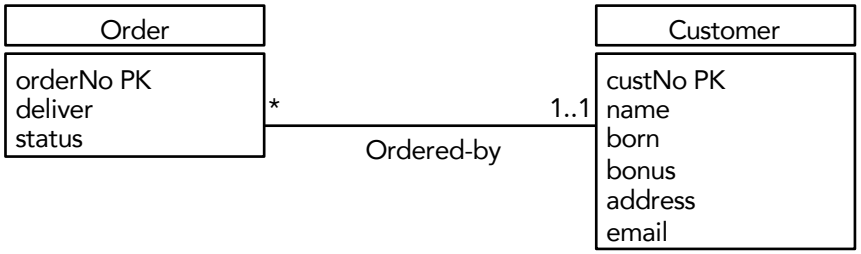
- For the associations we define relations:
  MadeBy(number, ID)
  OrderedBy(orderNo, custNo)
  BelongsTo(orderNo, number, count)



Note: No relation needs to be created for the Countinfo class.

About Keys of Relations

- A relation defined based on a class has the same key attributes as the class.
- Relations based on many-many associations inherit the key attributes from both of the classes connected by the association.
- Relations based on many-one associations only inherit the key attributes from the class in the many-side.

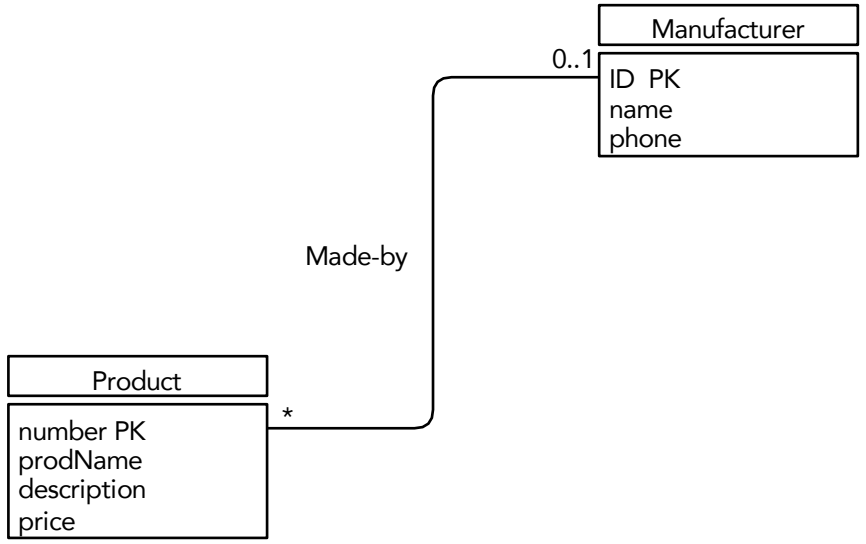


For example, OrderedBy(orderNo, custNo)

About Keys of Relations, Continued

- It's important not to define too many key attributes, as with the key attributes we make sure, that two tuples with the same values in the key attributes cannot be added.

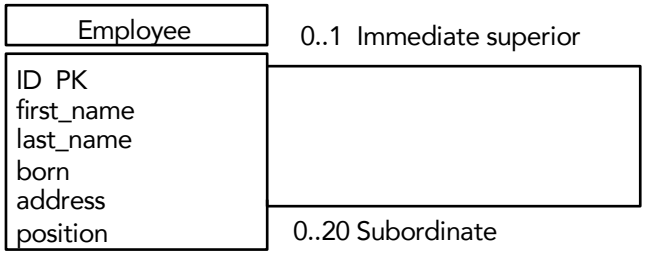
For instance, if the relation *MadeBy* would also have *ID* as its key attribute, we could add multiple tuples with the same product number, but with a different manufacturer.



E.g. MadeBy(number, ID)

Example: Relations for Self-Associations

- Here the same class occurs in the association twice:

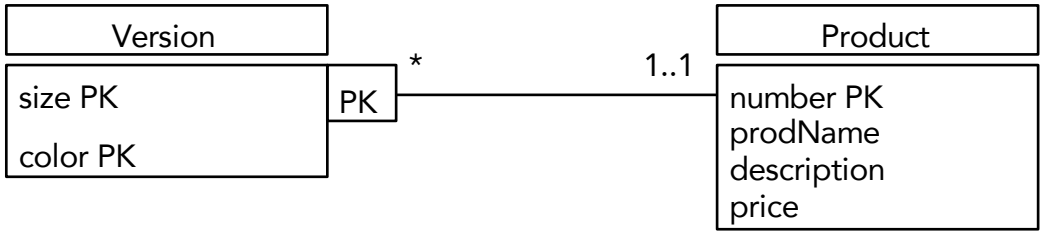


The key attributes will be added twice to describe both sides of the association.

- Hence, we define the relation:

Manages(subordinateID, superiorID)

Attributes of the class not sufficient for a key



- In addition to the key attributes of the class *Version*, we add the key attributes from the class *Product*.
- We should never create a relation for the association between *Version* and *Product*.
- If the class *Version* is part of some other association, the relation created based on this association should include the key attributes from both of the classes *Version* and *Product*.
- Relations:
  Products(number, name, description, price)
  Versions(prodNo, size, color)

REVIEW

CS-A1153 - Databases (Summer 2020)

LUKAS AHRENBERG

Aalto University
School of Science

BCNF

- Always show the steps you take to reach BCNF
- Compute the closure at each step to show if FDs are in BCNF or not
- Also when you have reached the point of no BCNF violation (show that there is none)

BCNF PROCEDURE (REPETITION)

1. Pick any non-trivial FD violating BCNF for `R`
    - (If none is found `R` is on BCNF)
    - **Here:** Show closures, and note violations (even if you are sure)
    - *This is the only point at which you can terminate the procedure*
2. Use it to create two (partially overlapping) sets of attributes
    - Set 1 : The closure of the determinants (the left side) of the violating FD
    - Set 2 : The union of the set of determinants with any attributes in `R` not already in Set 1.
    - **Here:** Show how you get the two sets
3. These two sets are the attributes of two new relations: `R1`, `R2`
    - **Here:** Remember to note which FDs are valid in the new relation
4. Apply the same procedure to `R1` and `R2`.

BCNF

`M(title,year,studioName,president,presAddr)`

Schema: $\{title, year, studioName, president, presAddr\}$

Functional Dependencies:

- $title\ year \rightarrow studioName$
- $studioName \rightarrow president$
- $president \rightarrow presAddr$

Is this in BCNF?

- *Always show closures* (If it is a complicated case, also show how you have gotten them.
- $\{title, year\}^+ = \{title, year, studioName, president, presAddr\}$ [OK]
- $\{studioName\}^+ = \{studioName, president, presAddr\}$ [Violation]
- $\{president\}^+ = \{president, presAddr\}$ [Violation]

BCNF

`M(title,year,studioName,president,presAddr)`

Schema: $\{title, year, studioName, president, presAddr\}$

1. Pick non-trivial violating

   $studioName \rightarrow president$

2. Set 1: The closure

   $\{studioName, president, presAddr\}$

3. Set 2: The union of the determinants and attributes not in Set 1

   $\{studioName\} \cup \{title, year\} = \{title, year, studioName\}$

4. Create two new relations

   Need to check these!

PROJECTION OF FDS (REPETITION)

What happens to an FD when the relation it is defined for is decomposed?

- The original FD's are not necessarily valid
- New FD's may result due to the projection of the original set

Let $\mathtt{R}$ *be a relation, decomposed into the relation* $\mathtt{R1}$ *(and some other relation).*

Let $\mathcal{S}$ *be the set of FD's for* $\mathtt{R}$*.*

Then valid FD's for $\mathtt{R1}$ *can be determined as*

For each possible subset of attributes $\mathcal{A}$ *of* $\mathtt{R1}$*, and some attribute* B *of* $\mathtt{R1}$*,* $\mathcal{A} \rightarrow B$ *is an FD, if the following conditions hold*

1. B *is included in* $\mathcal{A}^+$ *(with respect to* $\mathcal{S}$*)*
2. B *is not included in* $\mathcal{A}$

BCNF

Schema: $\{title, year, studioName\}$

1. Which FDs are present in the new relation?
2. Create all possible attribute subsets:

$$\{title\}, \{year\}, \{studioName\}, \{title, studioName\}, \{title, year\}, \{year, studioName\}$$

3. Use the FDs for M and check if there are any non-trivial FDs
4. Starting with the singletons we find $title\ year \rightarrow studioName$
5. Take the closure : $\{title, year\}^+ = \{title, year, studioName\}$
6. Remark that this is in BCNF and the only FD

7. Now you are done with this particular relation

Note: In complicated cases you might not be able to 'see' which FDs project

BCNF

Schema: $\{studioName, president, presAddr\}$

1. Which FDs are present in the new relation?
2. Create all possible attribute subsets:

$$\{studioName\}, \{president\}, \{presAddr\},$$
$$\{studioName, presAddr\}, \{studioName, president\},$$
$$\{president, presAddr\}$$

3. Use the FDs for M and check if there are any non-trivial FDs
4. Starting with the singletons we find
    - $studioName \rightarrow president$
    - $president \rightarrow presAddr$
5. **And so on…** (Follow the rest of the procedure.)

SQL - ROUND 4, PROBLEM 11

Using the database schema in Problem 1, write the following query in SQL: For each study program, list the number of the students who have completed at least 10.0 credits (list only those study programs which have at least one student who has completed at least 10.0 credits).

SQL - WHERE CAN I GET THE INFORMATION?

Students(ID, name, program, year)

Courses(code, name, credits)

Grades(studentID, courseCode, date, grade)

- Take your time; think about what you might need:
  - Program names can be gotten from `Students`
  - Credits for courses from `Courses`
  - Need `Grades` to link these via `studentID` and `courseCode`
- Need to find the programs of students with more than 10 credits
- Need to count those for each program

STUDENTS WHO HAS COMPLETED 10 CREDITS - 1

- Need to involve all three tables

```
SELECT ID
FROM Students, Grades, Courses
WHERE ID = StudentID AND code = courseCode;
```

STUDENTS WHO HAS COMPLETED 10 CREDITS - 2

- Need to group by ID

```
SELECT ID
FROM Students, Grades, Courses
WHERE ID = StudentID AND code = courseCode
GROUP BY ID;
```

- But still not taking credits into account

STUDENTS WHO HAS COMPLETED 10 CREDITS - 3

- Need HAVING clause

```
SELECT ID
FROM Students, Grades, Courses
WHERE ID = StudentID AND code = courseCode
GROUP BY ID
HAVING SUM(credits) >= 10;
```

ID
112233
212434
218311
224411
433511
442255
512434
512443
553311
987202

NOW WE CAN LOOK UP PROGRAMS WITH STUDENTS IN THIS SET

```
SELECT program, COUNT(ID)
FROM Students
WHERE ID IN
(SELECT ID
FROM Students, Grades, Courses
WHERE ID = StudentID AND code = courseCode
GROUP BY ID
HAVING SUM(credits) >= 10
)
GROUP BY program;
```

program	COUNT(ID)
AUT	2
BIZ	2
KEM	1
TFM	1
TIK	3
TUTA	1