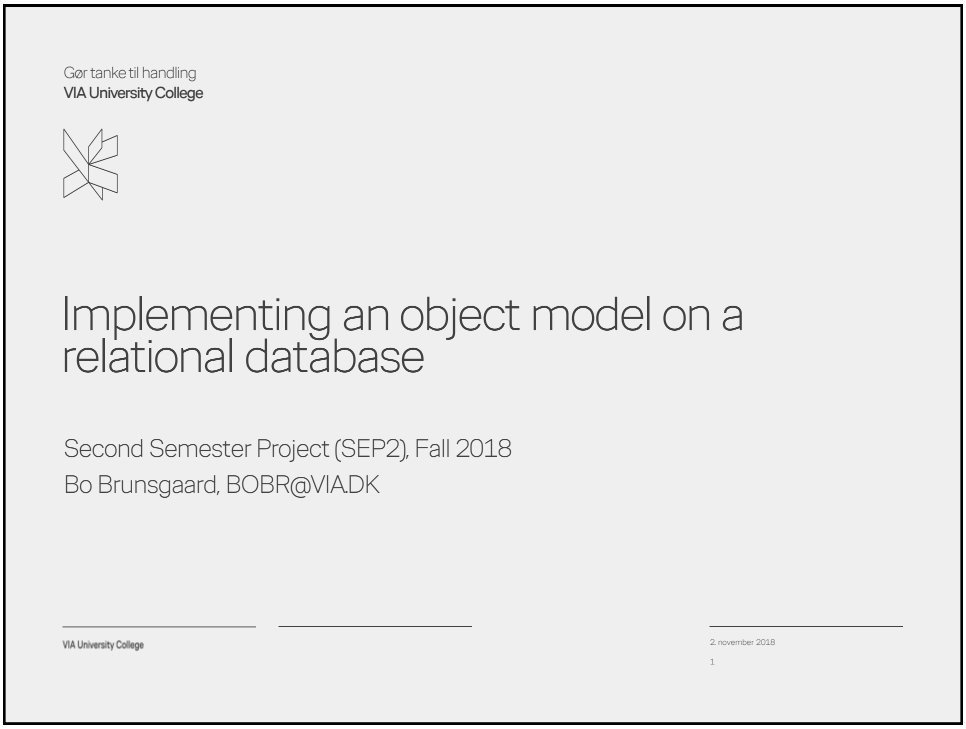
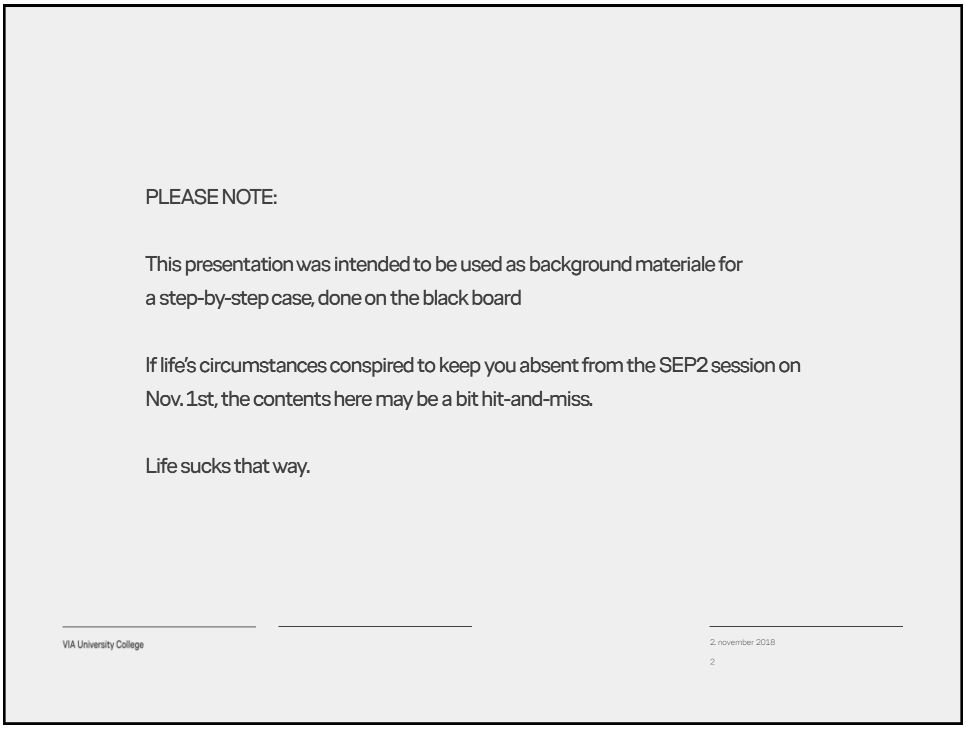


Gør tanke til handling
VIA University College

Implementing an object model on a relational database

Second Semester Project (SEP2), Fall 2018

Bo Brunsgaard, BOBR@VIA.DK



PLEASE NOTE:

This presentation was intended to be used as background materiale for a step-by-step case, done on the black board

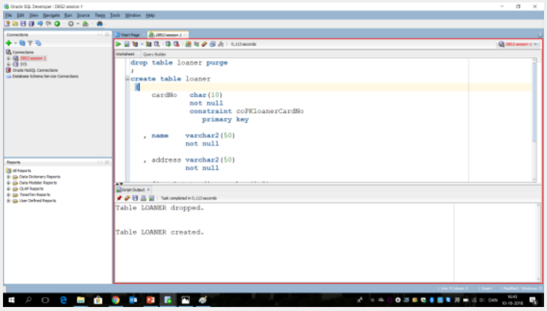
If life's circumstances conspired to keep you absent from the SEP2 session on Nov. 1st, the contents here may be a bit hit-and-miss.

Life sucks that way.

Agenda

- OO, ER and relational: same same, but different
- Case walk-through

A note on tools



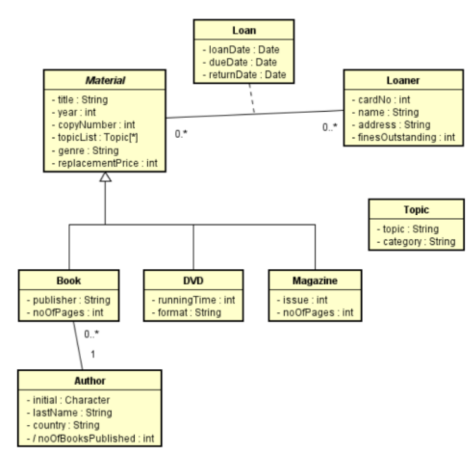
```
drop table loaner purge
;
create table loaner
(
    cardNo    char(10)
              not null
              constraint coPKloanerCardNo
                 primary key

  , name     varchar2(50)
             not null

  , address varchar2(50)
             not null
```

```
Table LOANER dropped.


Table LOANER created.
```

- You are using postgreSQL
- My examples are built using the Oracle RDBMS
- Tools *look* different
  - Mapping principles are exactly the same
  - SQL is (almost) the same
- I have certain habits – e.g. always naming my constraints
  - You may do things differently in DBS1; do as your DBS1 teacher says

A note on examples

- We will be using a small (analysis) class model for a minimal library system
- In some of the examples, I may make small deviations from from the model, either to demonstrate a point or to avoid clutter.



Loan
- loanDate : Date
- dueDate : Date
- returnDate : Date

Material
- title : String
- year : int
- copyNumber : int
- topicList : Topic[*]
- genre : String
- replacementPrice : int

Loaner
- cardNo : int
- name : String
- address : String
- finesOutstanding : int

Topic
- topic : String
- category : String

Book
- publisher : String
- noOfPages : int

DVD
- runningTime : int
- format : String

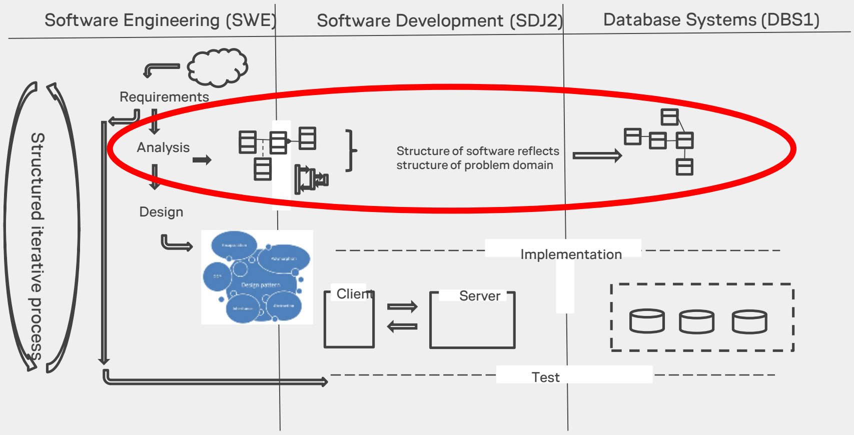
Magazine
- issue : int
- noOfPages : int

Author
- initial : Character
- lastName : String
- country : String
- / noOfBooksPublished : int

Bring ideas to life
VIA University College

First principles

Remember this?


Software Engineering (SWE)
Software Development (SDJ2)
Database Systems (DBS1)
Structured iterative process
Requirements
Analysis
Design
Design pattern
Structure of software reflects structure of problem domain
Implementation
Clien
Server
Test

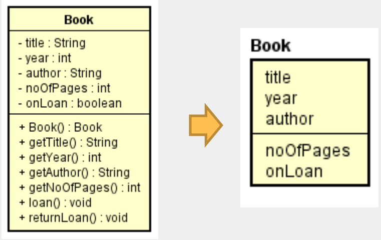
First principles

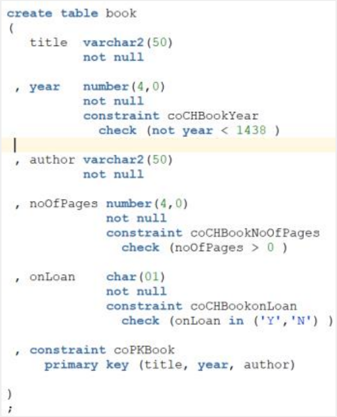
1. The structure of both the software and the database reflect the structure of the problem domain
2. The structure of the software and the database should, ideally, be the same
   - For technical reasons they cannot be exactly the same (impedance mismatch)
3. We capture the problem domain structure in a domain (analysis) model

Collary:

1. The database model & structure is <u>DERIVED</u> from the (object-oriented) domain model
2. Analysis/ domain modelling and database design are NOT disjoined activities!

Concepts





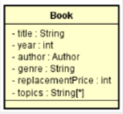
```
create table book
(
   title  varchar2(50)
          not null

 , year   number(4,0)
          not null
          constraint coCHBookYear
            check (not year < 1438 )

 , author varchar2(50)
          not null

 , noOfPages number(4,0)
             not null
             constraint coCHBookNoOfPages
               check (noOfPages > 0 )

 , onLoan    char(01)
             not null
             constraint coCHBookonLoan
               check (onLoan in ('Y','N') )

 , constraint coPKBook
     primary key (title, year, author)

)
;
```

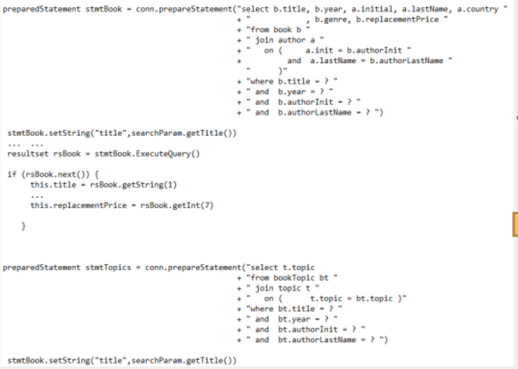
- Conceptually, class, entity and table are closely related.
  - An entity is the class without operations
  - The table is an implementation of an entity

First-cut mapping

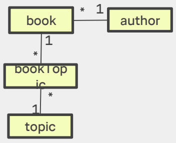
- A class becomes an entity / table
  - Collary: an object becomes a row of said table
- An attribute of a class becomes a column of a table
- An association between two classes becomes a primary/foreign key relationship
  - Collary: we need to identify which attributes of a class make up the foreign key!
- An association class becomes a table, holding the PKs of the tables at either end
- Specialized classes (can..) become separate entities/tables
  - Each table must then have all columns inherited from the generalized class
- Abstract classes?
  - Abstract classes? We don't have no stinking abstract classes!

Object-relational mapping





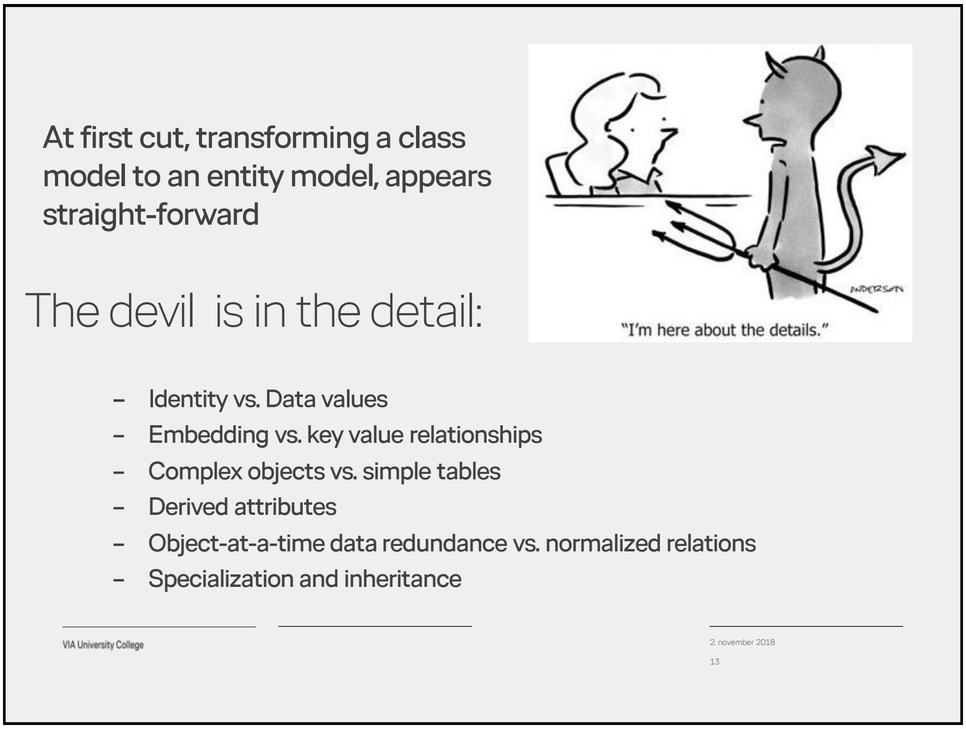
```
preparedStatement stmtBook = conn.prepareStatement("select b.title, b.year, a.initial, a.lastName, a.country "
                                                  + "           , b.genre, b.replacementPrice "
                                                  + "from book b "
                                                  + " join author a "
                                                  + "   on (     a.init = b.authorInit "
                                                  +          and  a.lastName = b.authorLastName "
                                                    "        )"
                                                  + "where b.title = ? "
                                                  + " and  b.year = ? "
                                                  + " and  b.authorInit = ? "
                                                  + " and  b.authorLastName = ? ")

 stmtBook.setString("title",searchParam.getTitle())
 ...  ...
 resultset rsBook = stmtBook.ExecuteQuery()

 if (rsBook.next()) {
      this.title = rsBook.getString(1)
      ...
      this.replacementPrice = rsBook.getInt(7)

    }


preparedStatement stmtTopics = conn.prepareStatement("select t.topic
                                                  + "from bookTopic bt "
                                                  + " join topic t "
                                                  + "   on (      t.topic = bt.topic )"
                                                  + "where bt.title = ? "
                                                  + " and  bt.year = ? "
                                                  + " and  bt.authorInit = ? "
                                                  + " and  bt.authorLastName = ? ")

 stmtBook.setString("title",searchParam.getTitle())
```



- You will need to code to:
  - translate from four tables into one object on reading a book from the database
  - Translate from one object into on four tables on inserting/updating/deleting

Bring ideas to life
VIA University College

Object-Oriented models, Entity-Relation models, and relational implementation



At first cut, transforming a class model to an entity model, appears straight-forward

The devil is in the detail:

"I'm here about the details."

- Identity vs. Data values
- Embedding vs. key value relationships
- Complex objects vs. simple tables
- Derived attributes
- Object-at-a-time data redundance vs. normalized relations
- Specialization and inheritance

VIA University College

2. november 2018

13

Gør tanke til handling
VIA University College

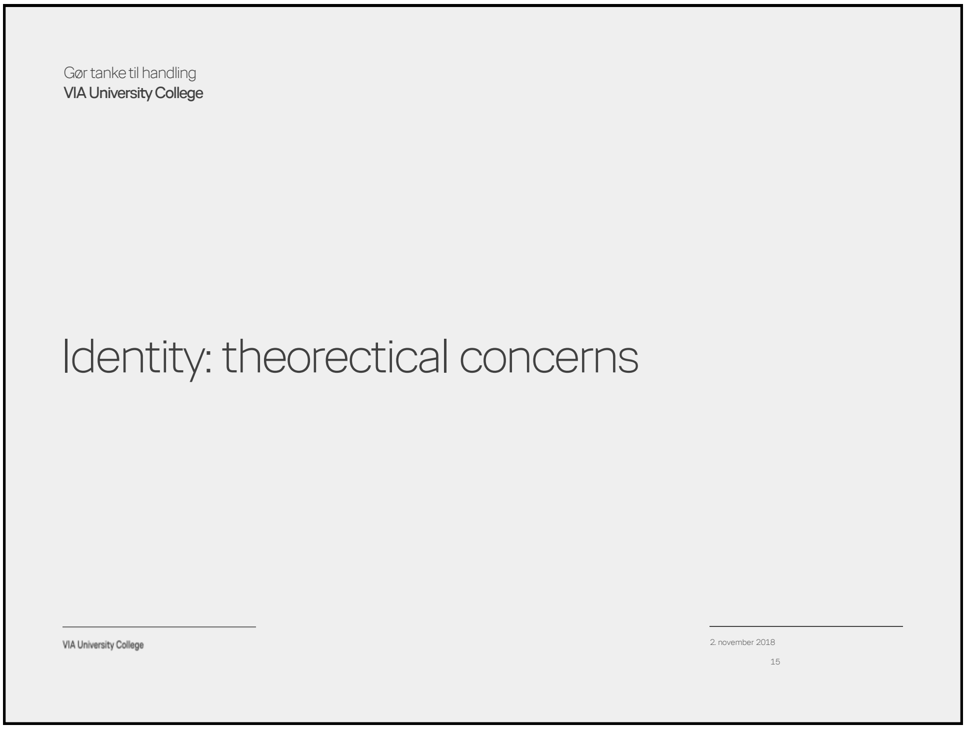
Identity

SHOW ME YOUR
ID

VIA University College

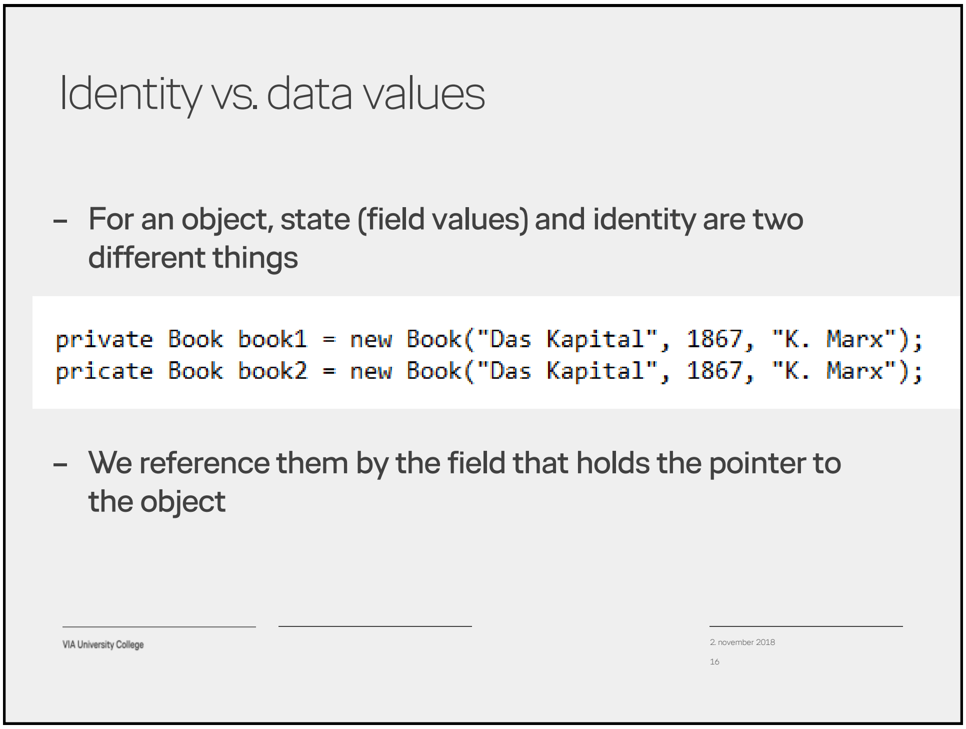
2. november 2018

14



Gør tanke til handling
VIA University College

Identity: theorectical concerns



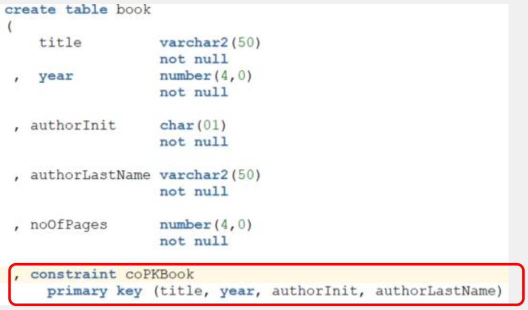
Identity vs. data values

- For an object, state (field values) and identity are two different things

```
private Book book1 = new Book("Das Kapital", 1867, "K. Marx");
pricate Book book2 = new Book("Das Kapital", 1867, "K. Marx");
```

- We reference them by the field that holds the pointer to the object

Identity vs. data values

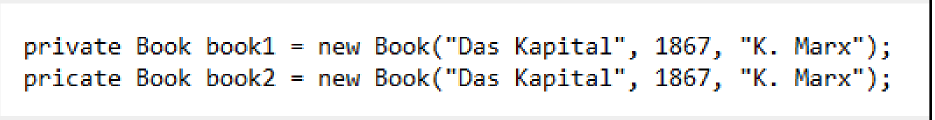


```
create table book
(
    title         varchar2(50)
                  not null
 ,  year          number(4,0)
                  not null

 , authorInit     char(01)
                  not null

 , authorLastName varchar2(50)
                  not null

 , noOfPages      number(4,0)
                  not null

 , constraint coPKBook
     primary key (title, year, authorInit, authorLastName)
```

- For a row in a table, identity is given by state
- We must identify the set of fields from the class definition that will uniquely identify an object/row -> `primary key`

Identity vs. data values

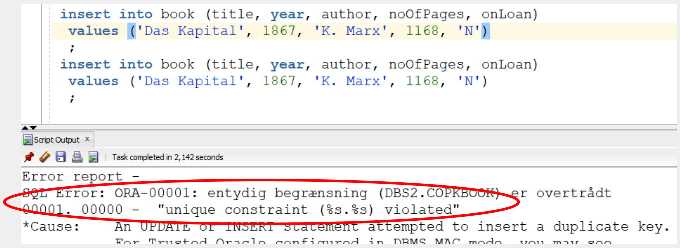
- An object has identity and state
- Two objects with exactly the same data will be two separate objects



```
private Book book1 = new Book("Das Kapital", 1867, "K. Marx");
pricate Book book2 = new Book("Das Kapital", 1867, "K. Marx");
```

Identity vs. data values

- An object has identity and state,
- A row in a table has only state: identity is given by state
- Two rows in a table with the same data will be duplicates:



```
insert into book (title, year, author, noOfPages, onLoan)
 values ('Das Kapital', 1867, 'K. Marx', 1168, 'N')
 ;
insert into book (title, year, author, noOfPages, onLoan)
 values ('Das Kapital', 1867, 'K. Marx', 1168, 'N')
 ;
```

```
Error report -
SQL Error: ORA-00001: entydig begrænsning (DBS2.COPKBOOK) er overtrådt
00001. 00000 -  "unique constraint (%s.%s) violated"
*Cause:    An UPDATE or INSERT statement attempted to insert a duplicate key.
           For Trusted Oracle configured in DBMS MAC mode, you may see
```

Identities

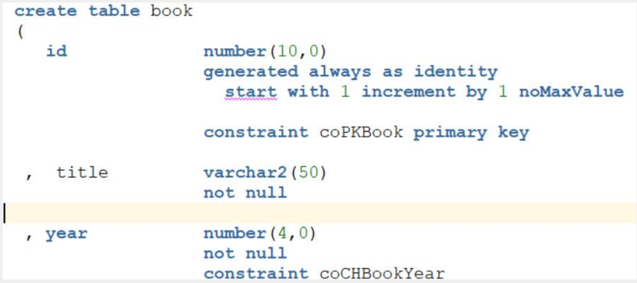
- Theoretically, this is a big deal. In practice, not so much
- Usually it indicates sloppy analysis and a bad domain modeling!
  - Often the result of mixing analysis and design classes
  - Real-world (analysis) objects can normally be told apart by their values
  - Design objects (e.g. connections, UI elements etc.) often cannot
- Think about it: in which system would we have two different students with the same studentID, social security number, name, gender, address, etc?

Gør tanke til handling
VIA University College

Identity: practical concerns

ID column

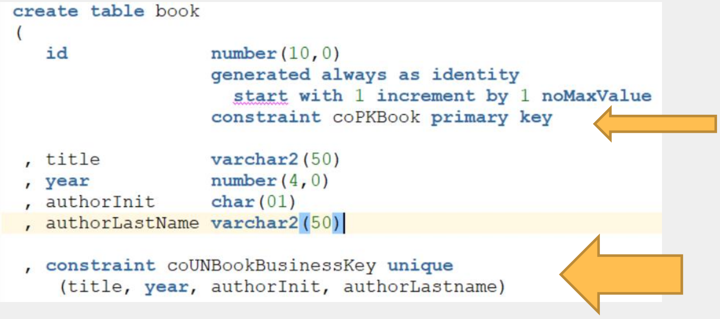
- IF the difference between state and identity is important, add an artificial key, and designate that as `primary key`



```
create table book
(
   id              number(10,0)
                   generated always as identity
                     start with 1 increment by 1 noMaxValue

                   constraint coPKBook primary key

 ,  title          varchar2(50)
                   not null

 , year            number(4,0)
                   not null
                   constraint coCHBookYear
```

ID and natural key
(one of my hobby horses. Rant warning!)

- If you add an artificial key, the business key uniqueness rule still exists and must be enforced!



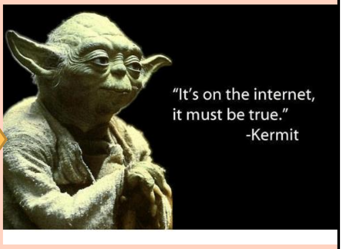
```
create table book
(
   id              number(10,0)
                   generated always as identity
                     start with 1 increment by 1 noMaxValue
                   constraint coPKBook primary key

 , title           varchar2(50)
 , year            number(4,0)
 , authorInit      char(01)
 , authorLastName  varchar2(50)

 , constraint coUNBookBusinessKey unique
     (title, year, authorInit, authorLastname)
```

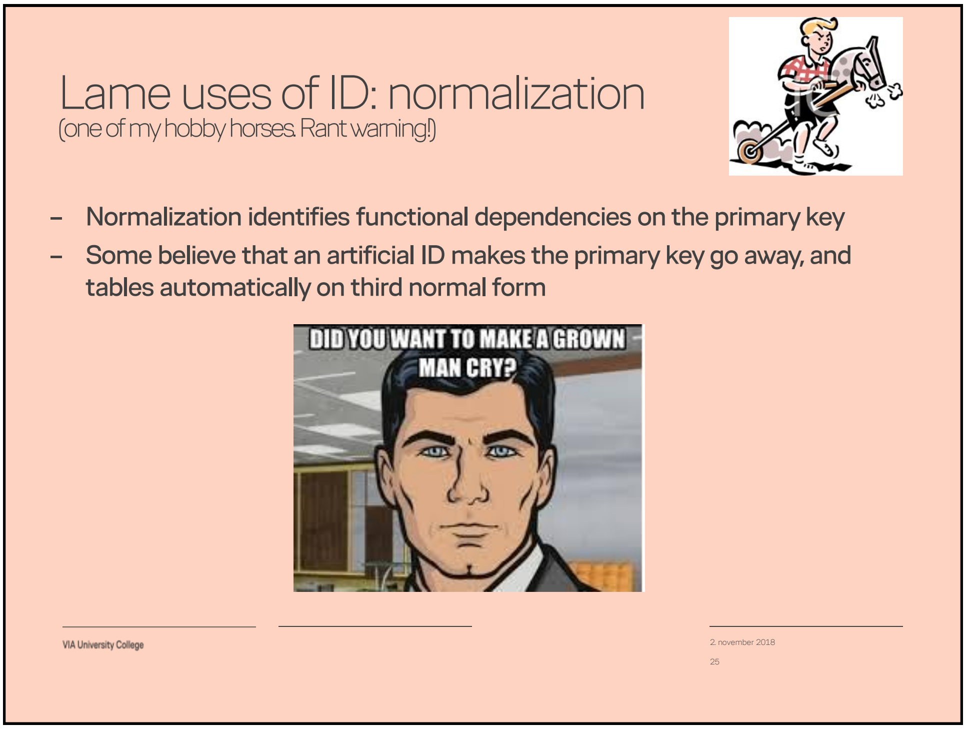
Lame uses of ID
(one of my hobby horses. Rant warning!)

- It has become customary always to add an artificial key to tables
  - (I am fighting a loosing battle against this)

But this does SO not fix any normalization issues

Even if:



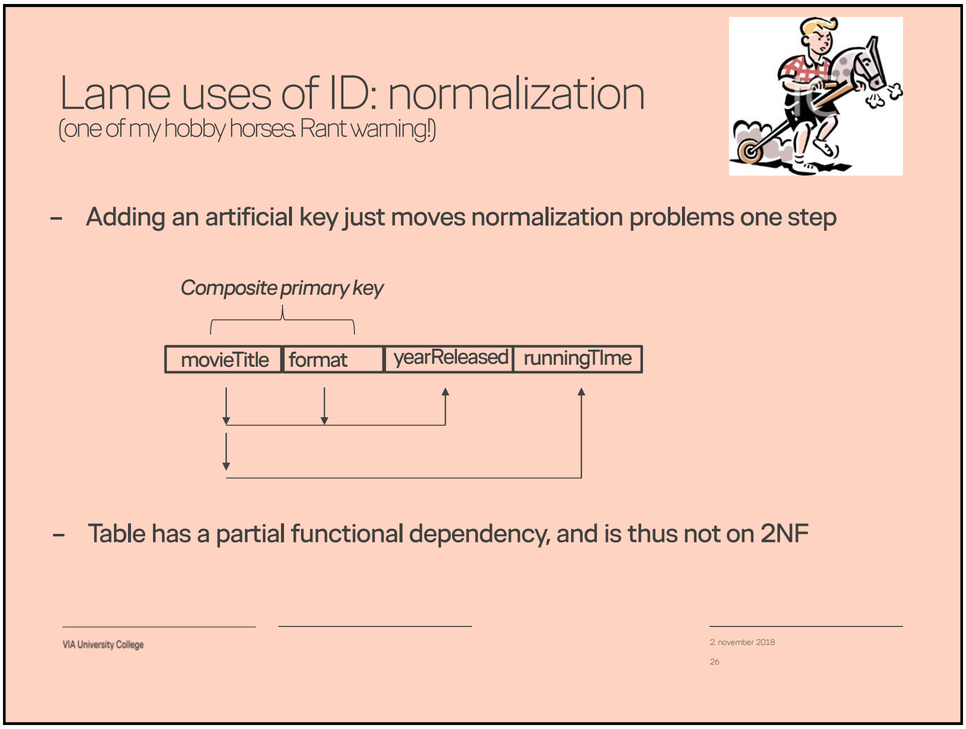


Lame uses of ID: normalization

(one of my hobby horses. Rant warning!)

- Normalization identifies functional dependencies on the primary key
- Some believe that an artificial ID makes the primary key go away, and tables automatically on third normal form

DID YOU WANT TO MAKE A GROWN MAN CRY?



Lame uses of ID: normalization

(one of my hobby horses. Rant warning!)

- Adding an artificial key just moves normalization problems one step

Composite primary key

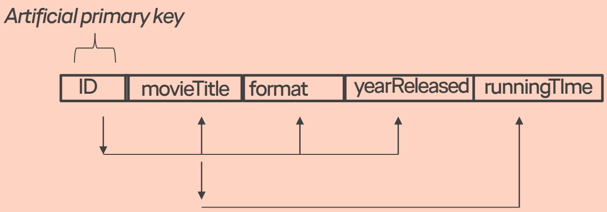
movieTitle	format	yearReleased	runningTIme

- Table has a partial functional dependency, and is thus not on 2NF

Lame uses of ID: normalization

(one of my hobby horses. Rant warning!)

- Adding an artificial key just moves normalization problems one step



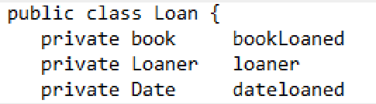
- Table now has a single-column PK, and is thus trivially on 2NF
  - A value cannot be determined by a sub-part of the PK if the PK has only one part
- But it now has a transitive dependency, ID -> movieTitle -> runningTime
- So we just exchanged a **1NF problem** for a **2NF problem**

Gør tanke til handling
VIA University College

Relationships

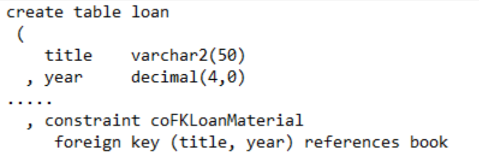
Pointers vs. Shared data values

- Objects reference other objects by pointers



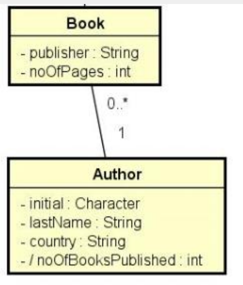
```
public class Loan {
   private book      bookLoaned
   private Loaner    loaner
   private Date      dateloaned
```

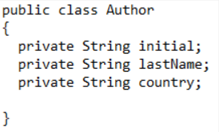
- Table rows reference other rows by keeping a copy of data values of key columns (primary/foreign key relationship)



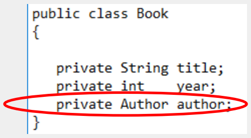
```
create table loan
 (
   title    varchar2(50)
 , year     decimal(4,0)
.....
 , constraint coFKLoanMaterial
    foreign key (title, year) references book
```

From reference to shared values (1 of 2)



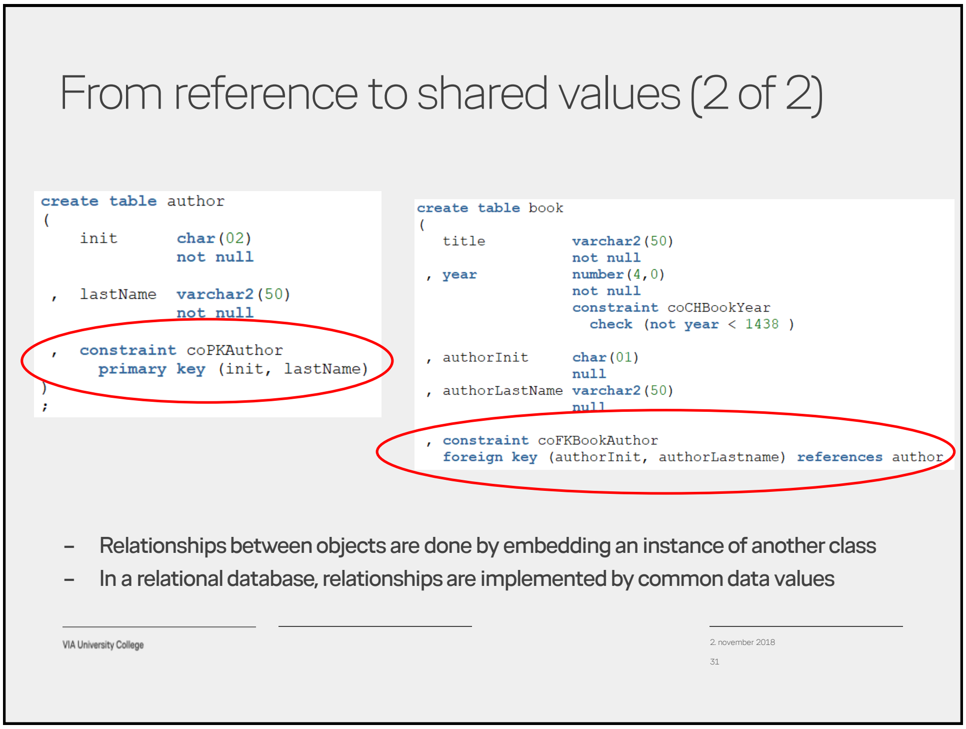


```
public class Author
{
  private String initial;
  private String lastName;
  private String country;

}
```



```
public class Book
{

   private String title;
   private int    year;
   private Author author;
}
```

- Relationships between objects are done by embedding a reference to an instance of another class
- In a relational database, relationships are implemented by common data values

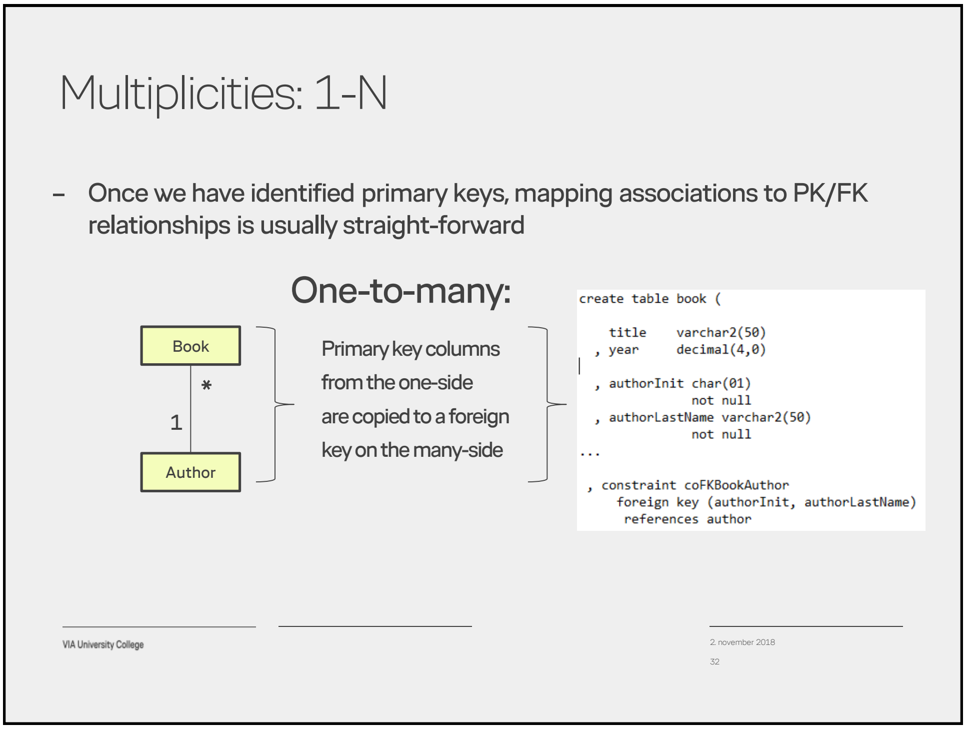


From reference to shared values (2 of 2)

```
create table author
(
   init       char(02)
              not null

 , lastName   varchar2(50)
              not null

 , constraint coPKAuthor
     primary key (init, lastName)
)
;
```

```
create table book
(
   title          varchar2(50)
                  not null
 , year           number(4,0)
                  not null
                  constraint coCHBookYear
                    check (not year < 1438 )

 , authorInit     char(01)
                  null
 , authorLastName varchar2(50)
                  null

 , constraint coFKBookAuthor
   foreign key (authorInit, authorLastname) references author
```

- Relationships between objects are done by embedding an instance of another class
- In a relational database, relationships are implemented by common data values



Multiplicities: 1-N

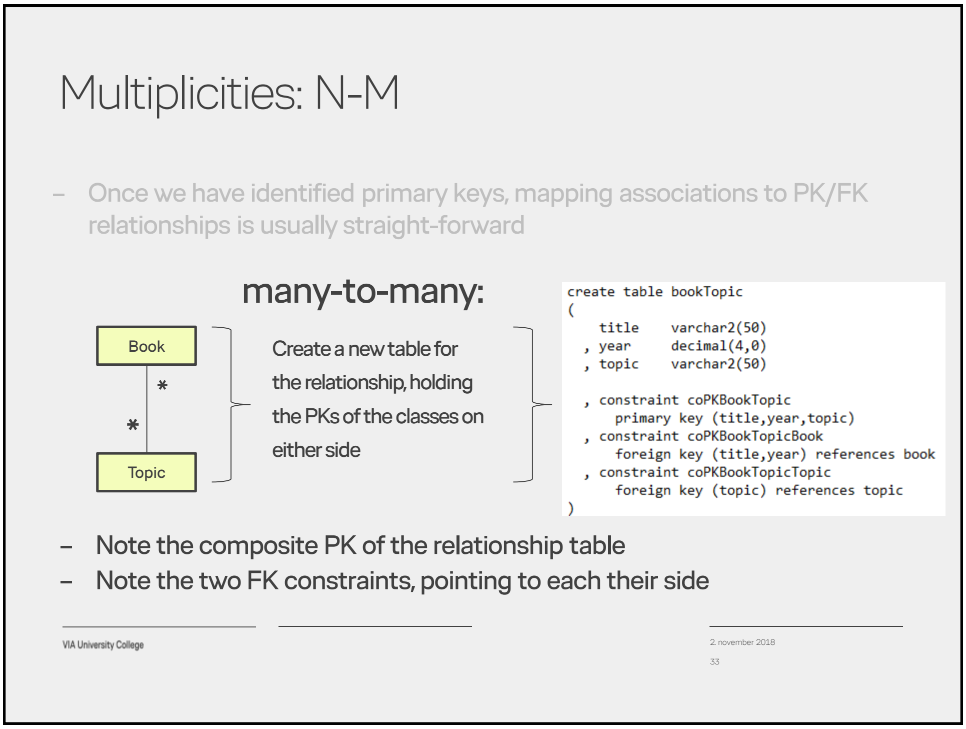
- Once we have identified primary keys, mapping associations to PK/FK relationships is usually straight-forward

One-to-many:

Book
*
1
Author

Primary key columns from the one-side are copied to a foreign key on the many-side

```
create table book (

   title    varchar2(50)
 , year     decimal(4,0)

 , authorInit char(01)
              not null
 , authorLastName varchar2(50)
              not null
...

 , constraint coFKBookAuthor
     foreign key (authorInit, authorLastName)
      references author
```



Multiplicities: N-M

- Once we have identified primary keys, mapping associations to PK/FK relationships is usually straight-forward

many-to-many:

Book

*

*

Topic

Create a new table for the relationship, holding the PKs of the classes on either side

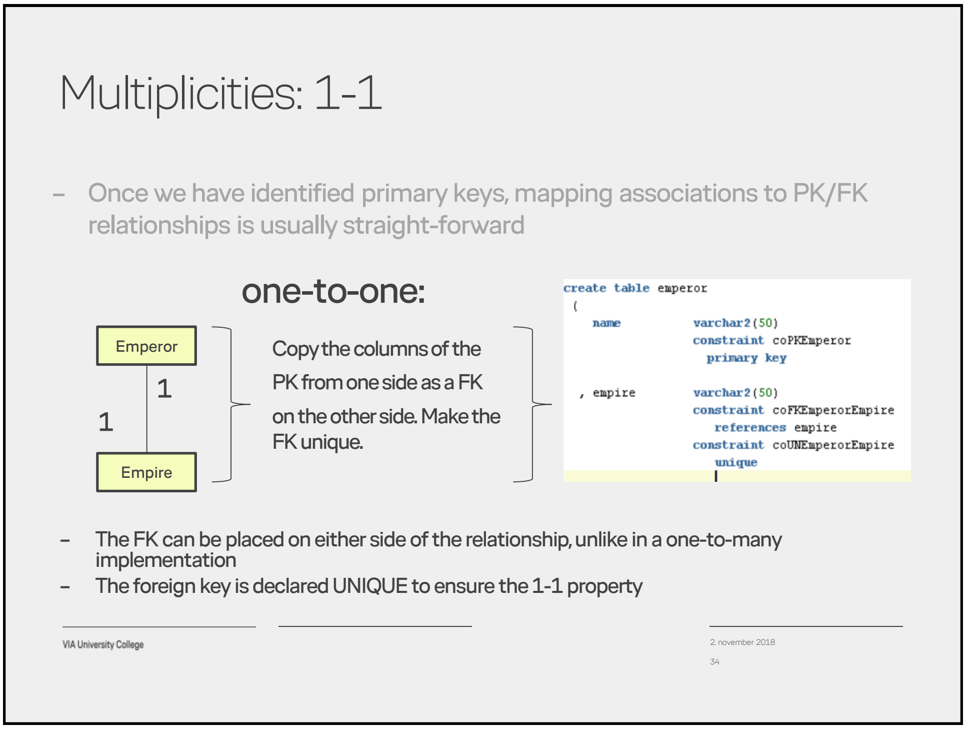
```
create table bookTopic
(
    title    varchar2(50)
  , year     decimal(4,0)
  , topic    varchar2(50)

  , constraint coPKBookTopic
      primary key (title,year,topic)
  , constraint coPKBookTopicBook
      foreign key (title,year) references book
  , constraint coPKBookTopicTopic
      foreign key (topic) references topic
)
```

- Note the composite PK of the relationship table
- Note the two FK constraints, pointing to each their side

VIA University College

2. november 2018

33



Multiplicities: 1-1

- Once we have identified primary keys, mapping associations to PK/FK relationships is usually straight-forward

one-to-one:

Emperor

1

1

Empire

Copy the columns of the PK from one side as a FK on the other side. Make the FK unique.

```
create table emperor
 (
   name              varchar2(50)
                     constraint coPKEmperor
                       primary key

  , empire           varchar2(50)
                     constraint coFKEmperorEmpire
                        references empire
                     constraint coUNEmperorEmpire
                        unique
```

- The FK can be placed on either side of the relationship, unlike in a one-to-many implementation
- The foreign key is declared UNIQUE to ensure the 1-1 property

VIA University College

2. november 2018

34

Gør tanke til handling
VIA University College

Relationships

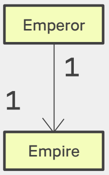
Subtleties

Direction of relationships (navigatibility)

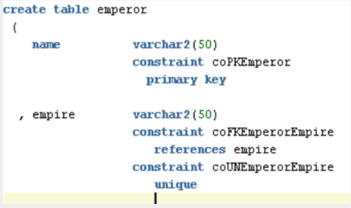
- Conceptually, associations are UNI-directional
- PK/FK relationships are always BI-directional
- Primarily a theoretical issue more than a practical one
- It can have an effect on:

Applies regardless of multiplicity of relationships

Directed one-to-one:



Copy the columns of the PK from the TO-side as a FK on the FROM side



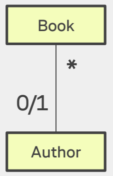
```
create table emperor
 (
   name          varchar2(50)
                 constraint coPKEmperor
                   primary key

  , empire       varchar2(50)
                 constraint coFKEmperorEmpire
                    references empire
                 constraint coUNEmperorEmpire
                    unique
```

- (it actually won't make any difference to the database (still bi-directional), but matches the code structure better)

Optional relationships

- An optional relationship is still implemented as a foreign key
- To make it optional, it is declared as NULLable
  - A foreign key constraint isn't checked if the actual value of the columns is/are NULL

Optional one-to-many



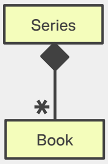
Copy the columns of the PK from the one side as a FK on the many side. Make the FK NULLable

```
create table book (

   title    varchar2(50)
 , year     decimal(4,0)

 , authorInit char(01)
             null
 , authorLastName varchar2(50)
             null
...

, constraint coFKBookAuthor
    foreign key (authorInit, authorLastName)
     references author
```

Aggregation

- In a composition, object the subordinate class MUST be related to an object of the superior class, AND the superior class is responsible for creating and deleting subordinate objects

Aggregation



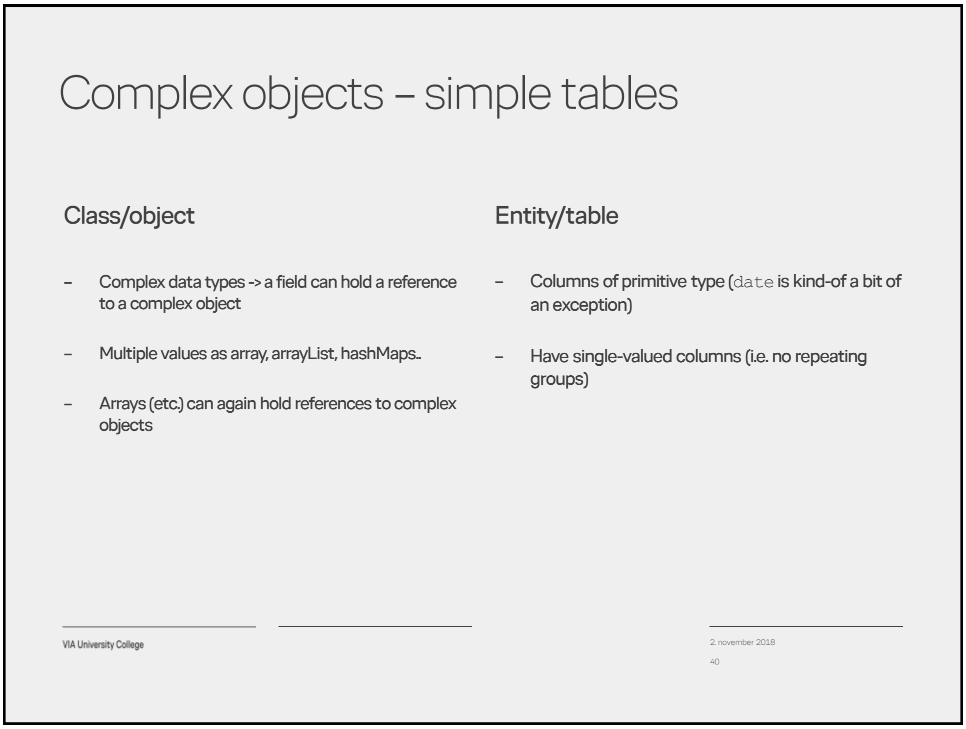
Implement as a one-to-many relationship with a DELETE CASCADE constraint

```
create table book (

   title    varchar2(50)
 , year     decimal(4,0)

 , seriesName varchar2(50)
             not null
             constraint coFKBookSeries
               references series
               on delete cascade
...
```

- Note that the foreign key is NOT NULL, since a book MUST be part of a series

Gør tanke til handling
VIA University College

Complex objects and simple tables



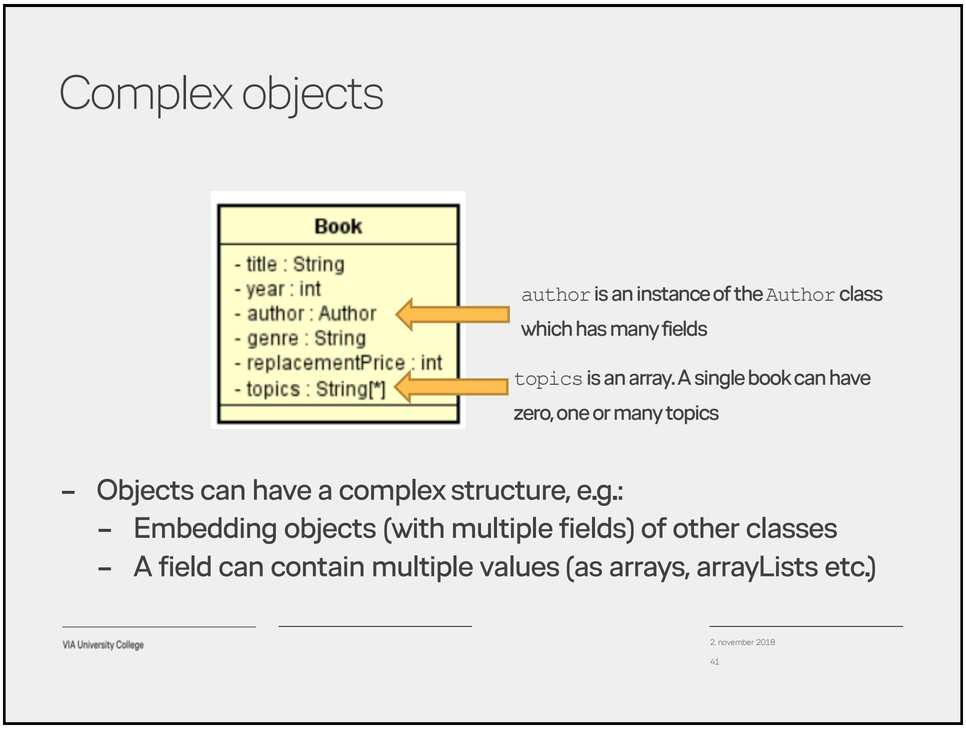
Complex objects – simple tables

Class/object

- Complex data types -> a field can hold a reference to a complex object
- Multiple values as array, arrayList, hashMaps..
- Arrays (etc.) can again hold references to complex objects

Entity/table

- Columns of primitive type (`date` is kind-of a bit of an exception)
- Have single-valued columns (i.e. no repeating groups)

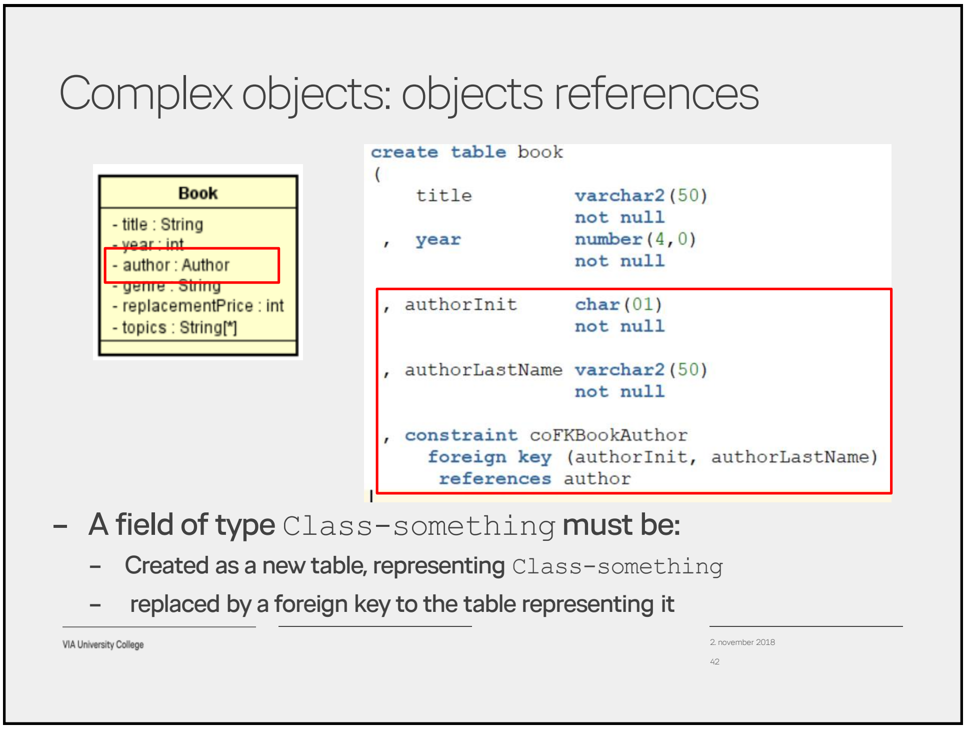


Complex objects

`author` is an instance of the `Author` class which has many fields

`topics` is an array. A single book can have zero, one or many topics

- Objects can have a complex structure, e.g.:
  - Embedding objects (with multiple fields) of other classes
  - A field can contain multiple values (as arrays, arrayLists etc.)



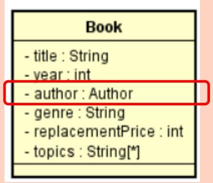
Complex objects: objects references

```
create table book
(
    title          varchar2(50)
                   not null
 ,  year           number(4,0)
                   not null

 , authorInit      char(01)
                   not null

 , authorLastName  varchar2(50)
                   not null

 , constraint coFKBookAuthor
     foreign key (authorInit, authorLastName)
      references author
```

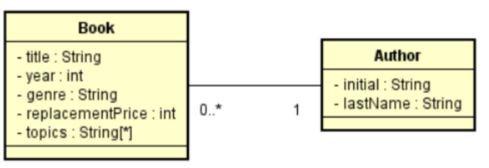
- A field of type `Class-something` must be:
  - Created as a new table, representing `Class-something`
  - replaced by a foreign key to the table representing it

Object references
(minor rant)

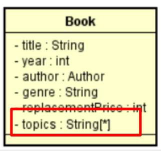
- Anyway, to me this is sloppy documentation practice

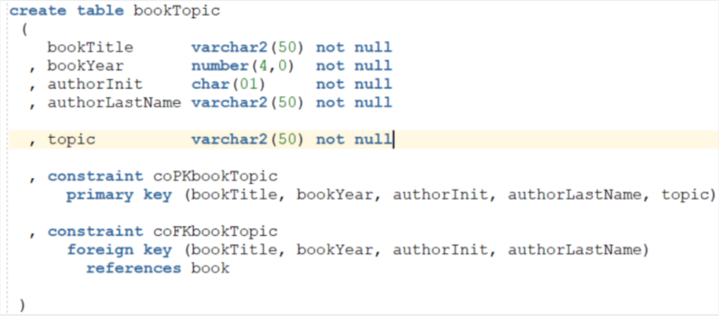


- It means exactly the same as this
- Much easier to understand at a glance



Complex objects: arrays

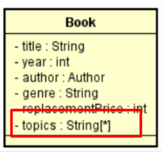


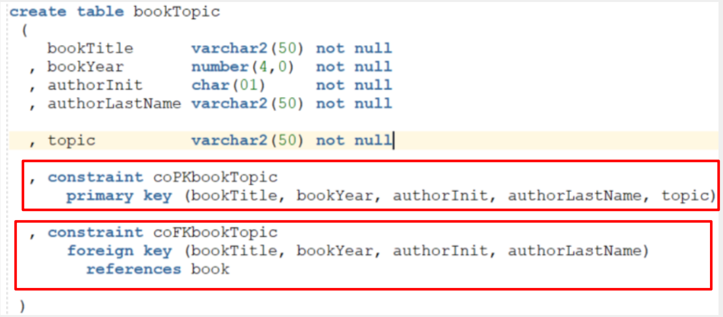


```
create table bookTopic
 (
    bookTitle      varchar2(50) not null
  , bookYear       number(4,0)  not null
  , authorInit     char(01)     not null
  , authorLastName varchar2(50) not null

  , topic          varchar2(50) not null

  , constraint coPKbookTopic
      primary key (bookTitle, bookYear, authorInit, authorLastName, topic)

  , constraint coFKbookTopic
      foreign key (bookTitle, bookYear, authorInit, authorLastName)
        references book
 )
```

- A multi-valued field must be split into a separate table.
- This will represent a `1..*` relationship (which is what the array actually is - one book contains (has) many topics)

Complex objects: arrays



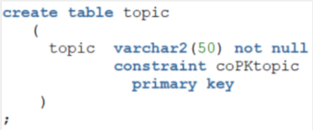


```
create table bookTopic
 (
    bookTitle      varchar2(50) not null
  , bookYear       number(4,0)  not null
  , authorInit     char(01)     not null
  , authorLastName varchar2(50) not null

  , topic          varchar2(50) not null

  , constraint coPKbookTopic
      primary key (bookTitle, bookYear, authorInit, authorLastName, topic)

  , constraint coFKbookTopic
      foreign key (bookTitle, bookYear, authorInit, authorLastName)
        references book

 )
```

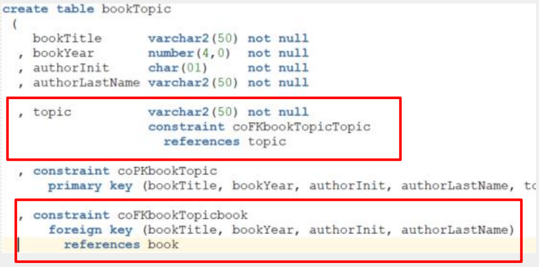
- The new table must have a foreign key to the original table (the new table is the many-side of the relationship)
- If the array was of a primitive type, PK = orginal table's PK + the value from the array

Complex objects: arrays
(a probably better solution in most cases)

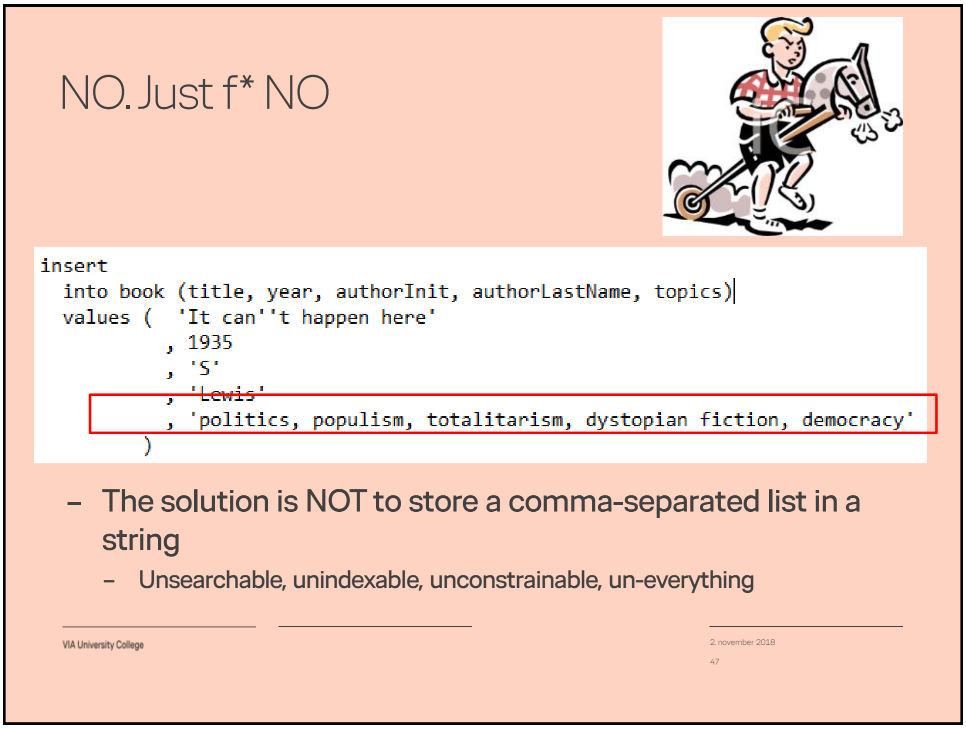
- Often better to create a table for the values that can appear in the array
  - All books now use same set of values
  - It now becomes a *:* relationship.



```
create table topic
    (
      topic  varchar2(50) not null
             constraint coPKtopic
               primary key
    )
;
```



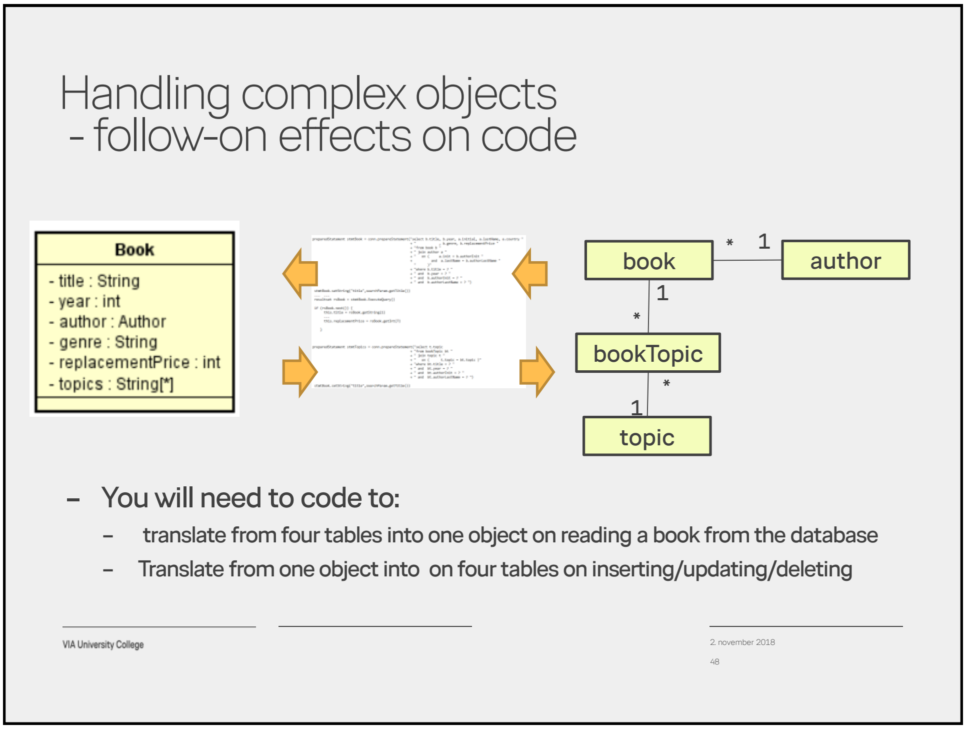
```
create table bookTopic
 (
    bookTitle      varchar2(50) not null
  , bookYear       number(4,0)  not null
  , authorInit     char(01)     not null
  , authorLastName varchar2(50) not null

  , topic          varchar2(50) not null
                   constraint coFKbookTopicTopic
                     references topic

  , constraint coPKbookTopic
      primary key (bookTitle, bookYear, authorInit, authorLastName, to

  , constraint coFKbookTopicbook
      foreign key (bookTitle, bookYear, authorInit, authorLastName)
        references book
```



NO. Just f* NO

```
insert
  into book (title, year, authorInit, authorLastName, topics)
  values (  'It can''t happen here'
           , 1935
           , 'S'
           , 'Lewis'
           , 'politics, populism, totalitarism, dystopian fiction, democracy'
         )
```

- The solution is NOT to store a comma-separated list in a string
  - Unsearchable, unindexable, unconstrainable, un-everything

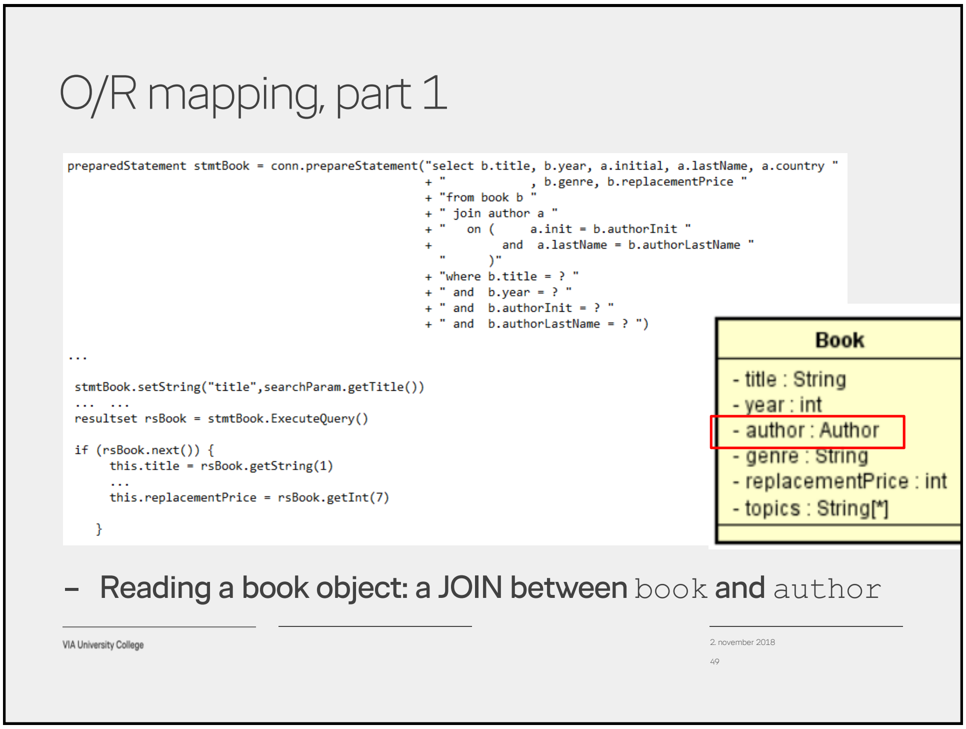


Handling complex objects - follow-on effects on code

Book

- title : String
- year : int
- author : Author
- genre : String
- replacementPrice : int
- topics : String[*]

- You will need to code to:
  - translate from four tables into one object on reading a book from the database
  - Translate from one object into on four tables on inserting/updating/deleting



O/R mapping, part 1

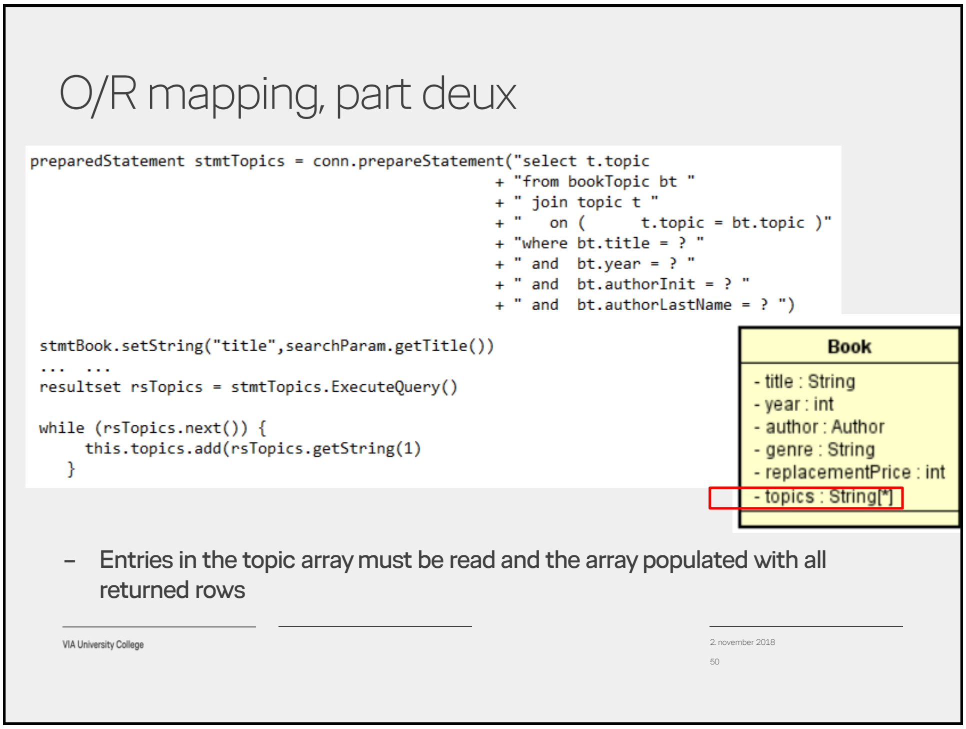
```
preparedStatement stmtBook = conn.prepareStatement("select b.title, b.year, a.initial, a.lastName, a.country "
                                                 + "              , b.genre, b.replacementPrice "
                                                 + "from book b "
                                                 + " join author a "
                                                 + "   on (     a.init = b.authorInit "
                                                 +           and  a.lastName = b.authorLastName "
                                                   "      )"
                                                 + "where b.title = ? "
                                                 + " and  b.year = ? "
                                                 + " and  b.authorInit = ? "
                                                 + " and  b.authorLastName = ? ")

...

 stmtBook.setString("title",searchParam.getTitle())
 ...  ...
 resultset rsBook = stmtBook.ExecuteQuery()

 if (rsBook.next()) {
      this.title = rsBook.getString(1)
      ...
      this.replacementPrice = rsBook.getInt(7)

    }
```

- Reading a book object: a JOIN between `book` and `author`

VIA University College

2. november 2018

49



O/R mapping, part deux

```
preparedStatement stmtTopics = conn.prepareStatement("select t.topic
                                                   + "from bookTopic bt "
                                                   + " join topic t "
                                                   + "   on (      t.topic = bt.topic )"
                                                   + "where bt.title = ? "
                                                   + " and  bt.year = ? "
                                                   + " and  bt.authorInit = ? "
                                                   + " and  bt.authorLastName = ? ")

 stmtBook.setString("title",searchParam.getTitle())
 ...  ...
 resultset rsTopics = stmtTopics.ExecuteQuery()

 while (rsTopics.next()) {
      this.topics.add(rsTopics.getString(1)
   }
```

- Entries in the topic array must be read and the array populated with all returned rows

VIA University College

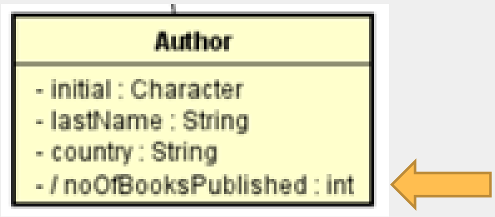
2. november 2018

50

Gør tanke til handling
VIA University College

Derived attributes

When an attribute really isn't there?



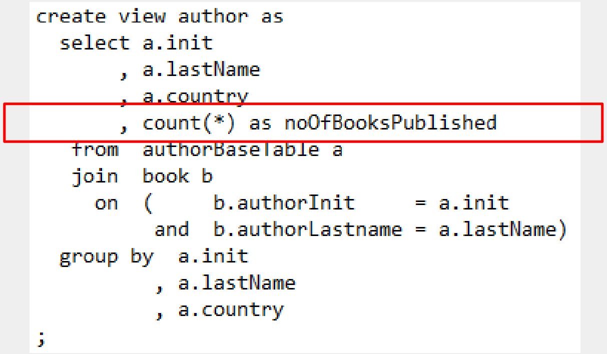
- `noOfBooksPublished` is not a field *per se*:
- It is derived from other information
  - In this case from the cardinality of the association with the `Book` class

- (In the design model it might very well be transformed into an operation (method) instead)

Derived attributes

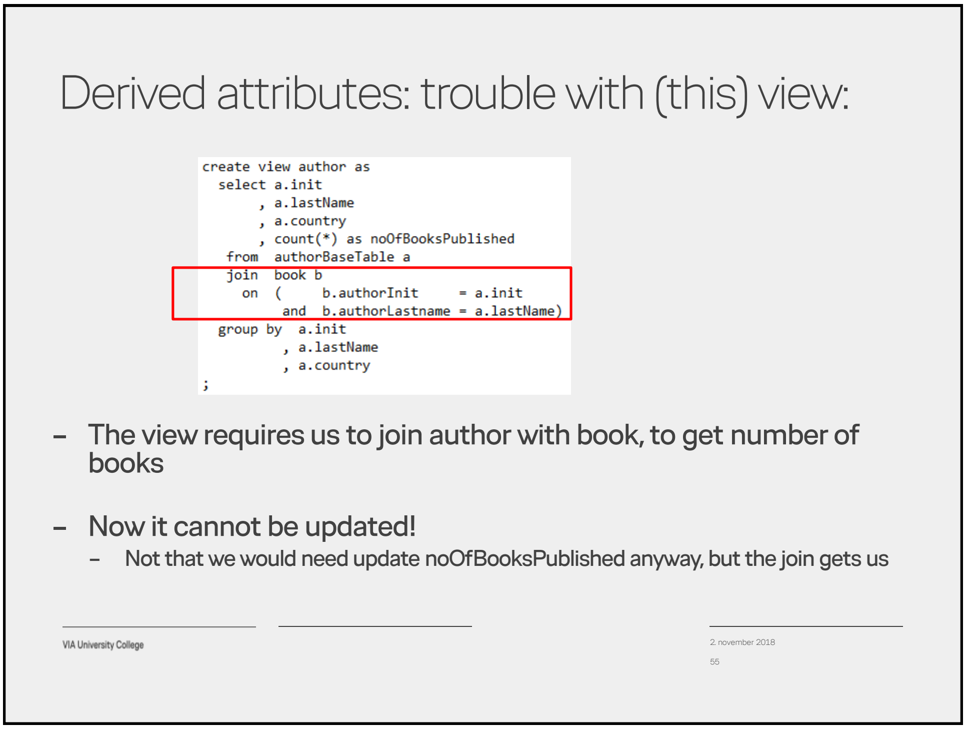
- Storing the derived value negates the idea of a derived field
- Leading to problems keeping it up-to-date
  - When a new book is added, we must remember to update the author
  - When a book is deleted, we must remember to update the author
  - When a book is updated, worst case, we must remember to update TWO authors
- So: derived attributes should NOT be mapped to columns of a table

Derived attributes: views:



```
create view author as
  select a.init
       , a.lastName
       , a.country
       , count(*) as noOfBooksPublished
   from  authorBaseTable a
   join  book b
     on  (     b.authorInit     = a.init
          and  b.authorLastname = a.lastName)
  group by  a.init
          , a.lastName
          , a.country
;
```

- We can create a view that calculates (i.e. derives) the value on-the-fly



Derived attributes: trouble with (this) view:

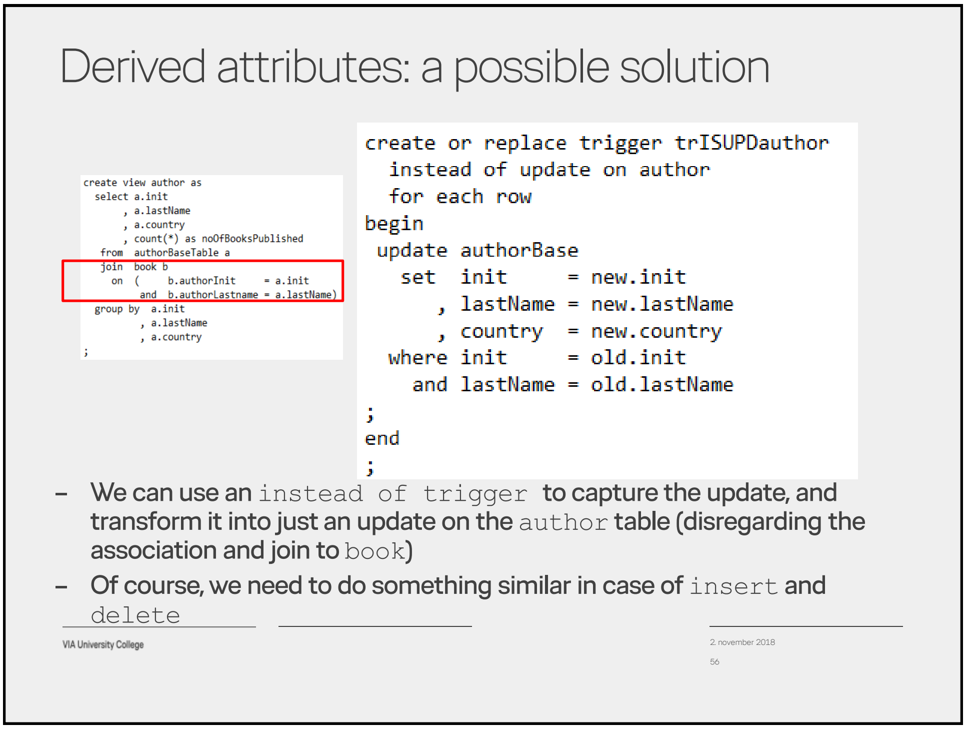
```
create view author as
  select a.init
       , a.lastName
       , a.country
       , count(*) as noOfBooksPublished
   from  authorBaseTable a
   join  book b
     on  (     b.authorInit     = a.init
          and  b.authorLastname = a.lastName)
  group by  a.init
          , a.lastName
          , a.country
;
```

- The view requires us to join author with book, to get number of books
- Now it cannot be updated!
  - Not that we would need update noOfBooksPublished anyway, but the join gets us

VIA University College

2. november 2018

55



Derived attributes: a possible solution

```
create view author as
  select a.init
       , a.lastName
       , a.country
       , count(*) as noOfBooksPublished
   from  authorBaseTable a
   join  book b
     on  (     b.authorInit     = a.init
          and  b.authorLastname = a.lastName)
  group by  a.init
          , a.lastName
          , a.country
;
```

```
create or replace trigger trISUPDauthor
  instead of update on author
  for each row
begin
 update authorBase
   set  init     = new.init
      , lastName = new.lastName
      , country  = new.country
  where init     = old.init
    and lastName = old.lastName
;
end
;
```

- We can use an `instead of trigger` to capture the update, and transform it into just an update on the `author` table (disregarding the association and join to `book`)
- Of course, we need to do something similar in case of `insert` and `delete`

VIA University College

2. november 2018

56

Derived attributes
(minor rant)

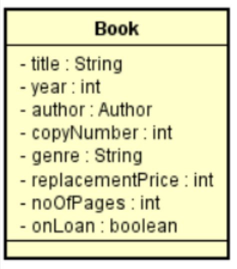
- Dealing with views and instead of triggers can become very convoluted, with little benefit
- Usually, I wouldn't bother handling derived attributes in the database
  - Let the application code handle it – calulate it in the `getNoOfBooksPublished()` method on the `book` object
- Exceptions may apply with the database is used by other applications and access through other interfaces

Gør tanke til handling
VIA University College

Normalization

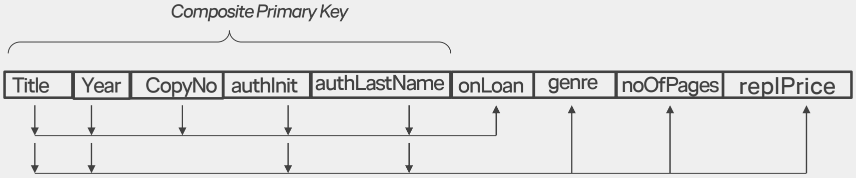
Object-at-a-time vs. Normalization

(my take, others may disagree!)



- Classes are often designed with an eye towards the specific application's needs
- Applications deal with individual objects, and don't consider objects that they are not dealing with
- So, it makes sense that if I look at a particular copy of a book, I get all the information about it (as above)

Table design from class:



- Table is only on 1NF (it has a partial functional dependency)
- `onLoan` depends on the entire primary key. You borrow a specific copy of a book
- `noOfPages`, `genre` and `replacementPrice` however, depend only on part of the primary key. They are the same for all copies of a book

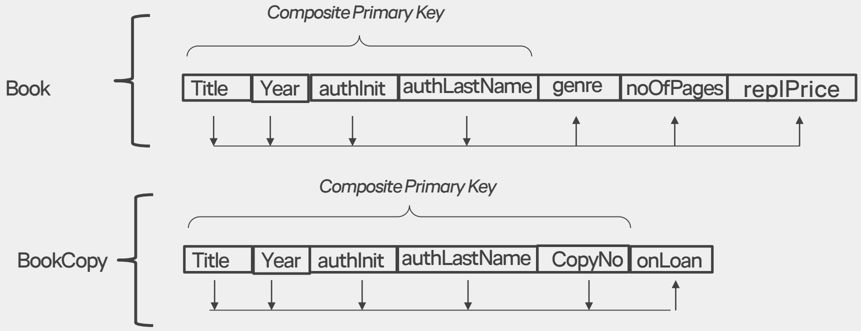
Table design from class:

- We have both INSERT, UPDATE, and DELETION anormalities
  - Your friendly neighborhood DBS1 teacher will supply all the details on this
- For now, suffices to say

Object-at-a-time vs. Normalization

(my take, others may disagree!)

- We need to split the table into two


Book

Composite Primary Key

Title	Year	authInit	authLastName	genre	noOfPages	replPrice

BookCopy

Composite Primary Key

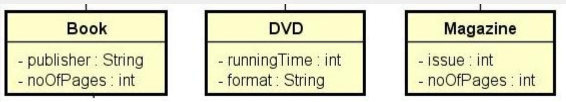
Title	Year	authInit	authLastName	CopyNo	onLoan

Gør tanke til handling
VIA University College

Specialization and inheritance

Specialization and inheritance

- Specialization and inheritance are elegant ways of handling common/separate properties



Book
- publisher : String
- noOfPages : int

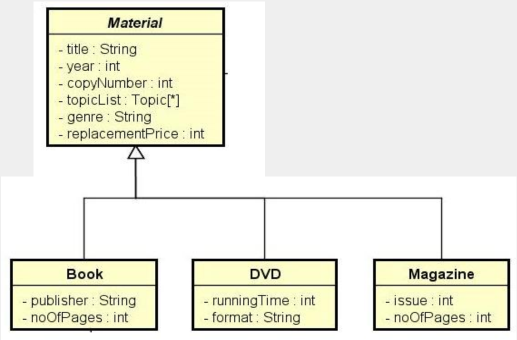
DVD
- runningTime : int
- format : String

Magazine
- issue : int
- noOfPages : int

- In the library, we have books, DVDs and magazines
- They are all different, with each their attributes

Specialization and inheritance

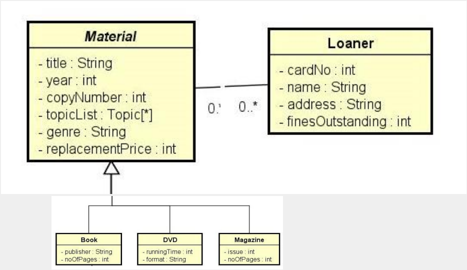
- But books, DVDs and Magazines have attributes in common
- We model these as a common ancestor, and books. DVDs and magazines inherit attributes from this

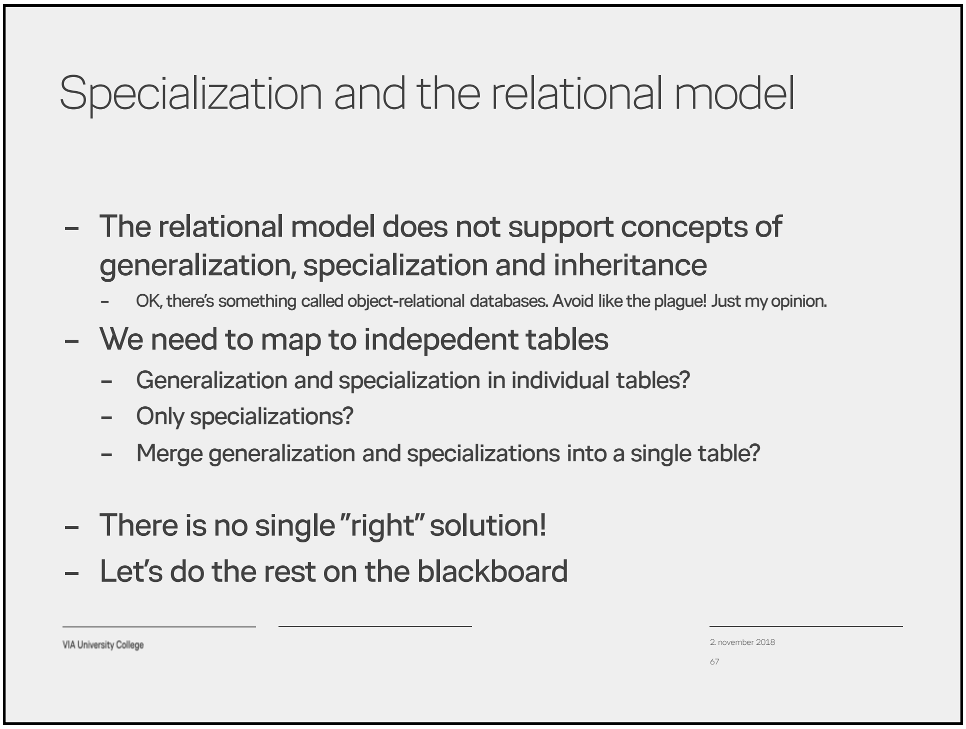


Specialization and inheritance

- Even better: shared relationships can be made to the super-class
- I simply loan a material which is then a book, a DVD or a magazine

- When I need it to be a material, it will be a material
- When I need it to be a book, it will be a book





Specialization and the relational model

- The relational model does not support concepts of generalization, specialization and inheritance
  - OK, there's something called object-relational databases. Avoid like the plague! Just my opinion.
- We need to map to indepedent tables
  - Generalization and specialization in individual tables?
  - Only specializations?
  - Merge generalization and specializations into a single table?
- There is no single "right" solution!
- Let's do the rest on the blackboard