## **4.0 Relational Database Implementation**

*4.1 Purpose*

The issue of this activity is to implement the design in a Relational DataBase (RDB). The Before going into details one has to decide between two streams:

We have chosen to us a directly transformation from design class diagrams to relations/tables as the design is very detailed, thus making EE/R-Diagramming superfluous.

*4.2 Activities*

In the next sections we shall describe and do the following activities:

* Choice of RDB
* Review of OOD model finding any needed changes
* Categorise the classes and transform into relations
* Transform associations and generalisation structures into relations
* Define and describe the relations, attributes, keys and domains
* Create a Database Model Diagram
* Normalise the tables

*4.3 Choice of RDB*

The definitive choice of tools (Postgress, Ingress, Oracle, MySQL, SQLServer, ODBC), does not have to be taken now, but we have decided that we will use the Relational Database MySQL (ref. …) and that the connection to MySQL (ref. …) shall be handled by a JDBC-driver provided by Oracle (ref…).

The reason is that:

* MySQL provides the necessary facilities also for a later web-applications
* MySQL is quite fast and can handle many users at the same time
* MySQL is free software

*4.3 Review of OOD model diagrams.*

When we look at the diagram (cf. App. A). All the classes in the model data class category with persistent data can directly be taken as candidates to relations, giving the following preliminary list of relations:

**Relation List**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fantasy Library** | **No.1** | **Version 1.0** | **Date**02.11.2011 | **Initials**<MCG 3> |
| **Relation name** | **Definition/Description** | **Aliases** | **Remarks/****Occurences** |
| AdultBorrower | Borrower older than 15 years (inclusive) | Adult | many |
| Book | Book for lending/reservation from library |  | many |
| Borrower | Person who has a borrower card to the library |  | many |
| ChildBorrower | Borrower younger than 15 years (exclusieve) | Child | many |
| Item | Item for lending/reservation/using from library | Book,CD,DVD |  |
| Lend | Lend of Item by a Borrower |  | 0 to many |

Besides minor printing errors (e.g. tille is title, ) there are a few suspicious issues to be looked at:

* Lent date
* The naming of Book
* The one-many Item-Lend
* The Author attribute

and some very suspicious issues:

* Number of item-exemplars, many of each
* Result of Use case “Find Late lends” is missing

Lent date

Why is the date of a Lend not registered ? Maybe the future date of return is enough maybe not?

Better is to introduce the lentDate.

Book class

There are probably other classes like CD, DVD, Video. It is evident that the way Book is used in our diagram is the same as it would have been for the other items. Probably the correct Generalisation Specialisation connections should be a super class Item and 3 sub classes: Book, CD, Video.

However as the use case leading to the diagram focuses on Lending we don’t introduce this extra Generalisation-Specialisation structure now. But right now we just keep the Book with Item.

Number of item exemplars

In the present diagram it was for simplicity assumed that there is only exemplar of each item (book, video, CD) to be lent. If this was not the case there would be an extra class ItemDecription holding the common information for these item-exemplars. This I think is the correct solution but this also implies a big change in our diagram. In order not to confuse you (the students) we will not take this into account right now. In appendix B (not given in this note) you will find the correct diagram and the correct database table diagram. I am quite sure that if you follow the instructions in UP such a big mistake will not be made. So make your use-cases and sequence diagrams with great care. Follow Lars’ instructions!!

Book-Lend association

This is a one-many association although a Book can only be lent to a Borrower one at a time. The question is: Should only present actual lends be registered or should historic information be possible. To insure the future use we keep the association as a one-many association. This means that Lend must have an extra attribute for when the item was returned: returnedDate.

Book-Author attribute/class

The attribute author is a little strange as a book can have many authors and one author can be author to many books. Thus it would be tempting to introduce a new class: Author and a Book-Athor association.

Find late lends

The use case “Find late lends” and the sequence diagram actually lead to a method *checkLateLends*, but this method is missing in LendCollection, so we just add this method This is outrageous and a big mistake. Again it shows how careful one must follow UP and remember everything.

With these remarks we shall now proceed and the new diagram is shown below.

 *4.4 Class categorisation*



We categorised the classes according to their responsibility into the following categories:

 Presentation classes, i.e GUI

 LibraryGUI ItemCatalogGUI, BorrowerRegisterGUI, LendCollectionGUI,

 Interface classes to technical devices and or other systems.

 None

Model-Holder/Collection class with the responsibility of holding objects of other classes.

 BorrowerRegister, ItemCatalog, LendCollection, AuthorCollection

Model/Data class holding persistent data/information

 Borrower, AdultBorrower, ChildBorrower, Item, Lend, Author

Utility classes

None for the moment but there might be Property-classes accessing flat files and there might be added DAO-classes for accessing the database, however this not of interest right now.

The classes in the Model-Collection class category are the classes to be transformed into DAO-classes handling the sql-queries/updates to the database.

The classes in the Model-Data class category are holding the persistent information to be put into the RDB and will be the classes to be transformed directly to relations/tables.

The other classes hopefully need very few changes.

*4.5 Transformation into relations/tables*

All the classes in the data class category with persistent data are directly taken as candidates to relations, giving the following preliminary list of relations:

**Relation List**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fantasy Library** | **No.1** | **Version 1.0** | **Date**02.11.2011 | **Initials**<MCG 3> |
| **Relation name** | **Definition/Description** | **Aliases** | **Remarks/****Occurences** |
| AdultBorrower | Borrower older than 15 years (inclusive) | Adult | many |
| Author | Writer of one/more books registered in library | Writer, Artist | many |
| AuthorBook | Relationship between Book and Author |  | many |
| Book | Book for lending/reservation from library |  | many |
| Borrower | Person who has a borrower card to the library |  | many |
| ChildBorrower | Borrower younger than 15 years (exclusieve) | Child | many |
| Item | Item for lending/reservation/using from library | Book,CD,DVD |  |
| Lend | Lend of Item by a Borrower |  | 0 to many |

Based on this we will now discuss the various possibilities of relationships/associations and possible changes in the interpretations of the design diagram.

*4.6 Transformation of aggregation/association structures.*

One-one (1-1) structures

There are no 1-1 structures.

One-many (1-\*) structures

Here we have Borrower-Lend, Item-Lend and AdultBorrower-ChildBorrower. These one-many aggregations/associations are transformed into relations but during this phase foreign keys are inserted into the weaker part also known as the many part relation participating in the association. Thus the connection is now represented by the foreign keys.

A special problem is the AdultBorrower-ChildBorrower where the connection is represented in OOD by the reference-attribute, parent, in ChildBorrower. Such a memory reference changes whole time according to use of memory and thus it should not be registered in a RDB. Therefore the parent attribute can just replaced by the foreign key parentCprNo. Alternatively parentCprNo might be added as an extra attribute if other parts of the program needs the parent-attribute.

Many-many (\*-\*) structures

Here we have Author-Book. Transform the many association into two one-many relations. Thus an extra relation/table AuthorBook is added. In AuthorBook is added foreign keys: authorId and itemNo. Thus the connection is now represented by this extra table and the foreign keys.

Author is always an author of something and therefore has mandatory participation (1..1) whereas item has optional participation (there might exist books where the authors are unknown).

##### 4.7 Transformation of Generalization-Specialization structure

Decide whether the sub types in each inheritance is not disjoint (overlapping) or disjoint, whether the participation is mandatory (forced) or optional.

We have Borrower-AdultBorrower/ChidBorrower which is considered as a disjoint structure as AdultBorrower and ChildBorrower exclude each other. Furthermore one cannot create an instance of only Borrower, meaning Borrower cannot stand alone; i.e. Borrower has mandatory participation in the structure.

Two possibilities thus exist:

1. Collapse the super type into all sub types. One table per sub type.
2. Introduce super type foreign key (cprNo in Borrower) in each subtype.

Both solutions will be correct. The first will give a simple inheritance structure whereas the second will give a more complex inheritance structure for the sub types.

The first possibility, to collapse Borrower with the sub types into the two tables: ChildBorrower and Adultborrower and will lead to changes in Lend and the use of Lend. Either we now need two Cpr. Numbers in Lend: childCprNo and adultCprNo and all sql-queries must check whether it is an adult borrower or a child borrower. Or we need two Lend tables: LendAdult and LendChild leading to various controls, too.

Therefore I decided to choose the second possibility. The reason for this decision is that the overall borrowing system will be much simpler.

Concerning Item-Book inheritance structure the discussion is similar as authors are related only to books and not CD’s.

*4.8 Database relations and model diagram*

The above discussion has clarified the relations and relationships/associations giving the following list of relations:

**Relationship/Association List**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fantasy Library****Lend-System** | **No. 1** | **Version 1.0** | **Date**2011-11-07 | **Initials**MC |
| **Relation**  | **Multiplicity** | **Association** | **Multiplicity** | **Relation**  | **Participation** |
| AdultBorrower | 1..1  | Has children | 0..\* | ChildBorrower | O : M |
| Author | 1..1 | Wrote an Item | 1..\* | AuthorBook | M : M |
| Item  | 1..1 | Is a Book | 0..1 | Book | O : M |
| Borrower | 1..1  | Has a Lend | 0..\* | Lend | O : M |
| Borrower | 1..1 | Is an adult | 0..1 | AdutlBorrower | O : M |
| Borrower | 1..1 | Is a child | 0..1 | ChildBorrower | O : M |
| Item | 1..1 | Has a Lend | 0..\* | Lend | O : M |

Shorts used as follows:

|  |
| --- |
| Participation |
| M: MandatoryO: Optional |

The table is to be read as follows:

Borrower might be involved in zero or many lends and a Lend belongs exactly to one Borrower.

Lend has mandatory participation in this association; i.e can not be created without a Borrower. Borrower has optional participation in this association; i.e. a Bottower can be created and exist without any Lend.

*Database diagram*

The class diagram was used as a starting point and based on the relations/relationships we can now draw the database model diagram:



*4.9 Attributes, primary keys and foreign keys*

We have the following relations/tables and attributes:

**Relation Attribute List**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fantasy Library** | **No.1** | **Version 1.0** | **Date**02.11.2011 | **Initials**<MCG 3> |
| **Relation name** | **Attributes**  |
| AdultBorrower | parentCprNo(FK), address, email, phone |
| Author | authorId, name, nationality, birthday |
| Borrower | cprNo, name, bType, regYear, state |
| Book | itemNo(PK), publisher isbn |
| ChildBorrower | childCprNo(FK), sex, parentCprNo(FK) |
| Item | itemNo, title, pubYear, itemType, state |
| AuthorBook | itemNo(FK), authorId(FK) |
| Lend | itemNo(FK), cprNo(FK), returnDate |

Most of the attributes were just taken directly from the originals but the following changes and extensions were made in accordance with our former discussion:

AdultBorrower: parentCprNo is both a primary key and foreign key to Borrower.

Borrower: registration year regYear instead of year, which is a reserved word in sql.

ChildBorrower: childCprNo is both a primary key and foreign key to Borrower.

Item: publishing year regYear instead of year, which is a reserved word in sql.

The primary keys are marked as underlined and most like cprNo, itemNo are AuthorId are evident.

In ItemAuthor and Lend we decided to use non pure implementation (using combined foreign keys as primary keys). In AuthorBook the composition of the two foreign keys (itemNo, authorId) uniquely identifies the row.

In Lend we could have had have two possible candidate keys:

 itemNo(FK), cprNo(FK), returnDate

or itemNo(FK), cprNo(FK), returnedDate

or itemNo(FK), cprNo(FK), lentDate

if a returnedDate (date of when the item was returned) should be registered. However a primary key can not have any attributes which might become null and returnedDate is unknown until it is returned whereas returnDate and lentDate is known when the Lend is created. Using a faked returnedDate will complicate searching and therefore the last one is chosen as primary key.

*4.10 Normalisation*

A table where the primary key is just one attribute and this attribute uniquely identifies the row is automatically on 2NF and if no attributes determines other attributes the table will be on 3NF. Thus Borrower, ChildBorrower, Item, Author are all on 3NF.

All tables where all the attributes also are part of the primary key are on 3NF. Thus AuthorBook is on 3NF.

AdultBorrower is on 2NF as the address is a composite attribute consisting of street, no, postNo and City. As postNo uniquely identifies City in Denmark it is not on 3NF but on 2NF.

Lend is on 3NF.

This needs more explanation and some step by step explanations but I don’t have more time for this.

*4.11 Specification of attributes and domains*

We can now describe and specify attributes and domains

**Relation Attribute Description List**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fantasy Library****Lend-System**  | **No. 1** | **Version 1.0** | **Date**2011-11-07 | **Initials**<MC> |
| **Relation** | **Attribute** | Key | Description | **Default Value** | **Allowed****NULL** | **Composite** |
| AdultBorrower | parentCprNoaddresse-mailphone | PK,FK | Refers to Borrower |  | NoNoYesNoNo | NoYes (critical)NoNoNo |
| Author | authorIDnamenationalitybirthDay | PK  | Uniquely identifies |  | NoNoNoNo | NoNoNoYes |
| AuthorBook | itemNoauthorID | PK,FKPK,FK | Referers to BookReferent to Author |  | NoNo | NoNo |
| Book | itemNopublisherisbn | PK, FK | Uniquely identifies |  | NoNoNoNo | NoNoNoNo |
| Borrower | cprNonamebTyperegYearstate | PK | Uniquely identifies  | Active | NoNoNoNoNo | NoNoNoNoNo |
| ChildBorrower | childCprNoparentCprNosex | PK,FK FK | Refers to BorrowerRefers AdultBorrower | Null | NoNoYes | NoNoNo |
| Item | itemNotitle itemTyperegYearstate | PK | Uniquely identifies | Registered | NoNoNoNoNo | NoNoNoNoNo |
| Lend | itemNocprNoreturnDatereturnedDate | PK,FK PK,FKPK  | Refers to ItemRefers to Borrower | Null | NoNoNoYes | NoNoNo Yes |

*Remark*

Primary keys (PK) and foreign keys (FK) are just marked but discussed in the text.

**Attribute Domains**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fantasy Library****Lend-System**  | **No.** | **Version** | **Date**xxxx-xx-xx | **Initials**<NN> |
| **Relation** | **Attribute** | **Data Type** | **Set of Value** | **Format** |
| AdultBorrower | parentcprNoAddresse-mailphone | VarChar(10)VarChar(60)Varchar(30)Numeric(12) | 0-9digits and cpr-rulesAnyAny0-9 digits | ddmmyy-xxxxnonenonexxxxxxxxxxxx |
| Author | authorIDnamenationalitybirthDay | VarChar(10)VarChar(30)VarChar(18)Date | AnyAnyAny0-9digits | NoneNoneNonexx-xx-xxxx |
| AuthorBook | itemNoauthorID | Numeric(20)VarChar(10) | 0-9digitsAny | NoneNone |
| Book | itemNoisbnpublisher | Numeric(20)VarChar(15)VarChar(30) | 0-9digits 0-9digits |  |
| Borrower | cprNonamebTyperegYearstate | VarChar(10)VarChar(30)VarChar(6)Numeric(4)VarChar(7) | 0-9digits and cpr-rulesAnyFemale and male 0-9digitsActive, Passive, Illegal | ddmmyy-xxxxnonexxxx |
| ChildBorrower | childCprNoparentCprNosex | VarChar(10)VarChar(30)VarChar(6) | 0-9digits and cpr-rules0-9digits and cpr-rules Female and male | ddmmyy-xxxx ddmmyy-xxxx |
| Item | itemNotitle itemTyperegYearstate | Numeric(20)VarChar(20)VarChar(10)Numeric(4)Varchar(10) | 0-9digitsAny 0-9digitsAny Registered. Available, Reserved, Lent, Lost | NoneNonexxxxNone |
| Lend | itemNocprNoreturnDatereturnedDate | Numeric(20)VarChar(10)DatetimeDatetime | 0-9digits 0-9digitsDB data typeDB data type | Nonexxxxxx-xxxx |

*4.12. Transform the collection/holder objects. How are the relations identified ?*

This discussion is only important if your architecture didn’t inculde the DAO-layer from the beginning. These classes are holding/controlling important functionality in the Model layer and are not used as tables as the relations are automatically identified by the table names. *The classes should be used indirectly to set up the functionality for the database.* Therefore instead of changing the methods clone the method-signatures into new *DAO-classes and let the methods handle the SQL-sentences.*

This decision might include the choice of embedded SQL, 2- or 3- tier etc.

As a golden rule the methods can be categorized and placed on classes as follows:

DAO-Classes with the 4 CRUD-methods (Create, Read/Find/Search, Update, Delete)

One DAO-class per table and only when the methods involve only one table.

Read/Find/Search only for the “sql-select \*” only using the primary key

Delete only for the “sql-delete” only using the primary key

Update only for updating all data in one row and only using the primary key

DAO-QueryClasses with all other methods

One DAO-QueryClass per sql-sentence and only with one doIt-method

Or one DAO-QueryClass per use case.

In the first state of implementation just keep all collection-classes but be aware that the content, the number of classes and methods must be cloned to handle a RDB.

Furthermore a general Query class will be needed.

Whether or not the collection classes should be controllers of the DAO-Classes is an open question. Alternatively the controllers must have direct access both to the DAO-classes and the collection-classes.

*4.12 Conclusion*

Next step will be to look at the implementation of the tables and the controller classes…..Bla bla

**Appendix A**

