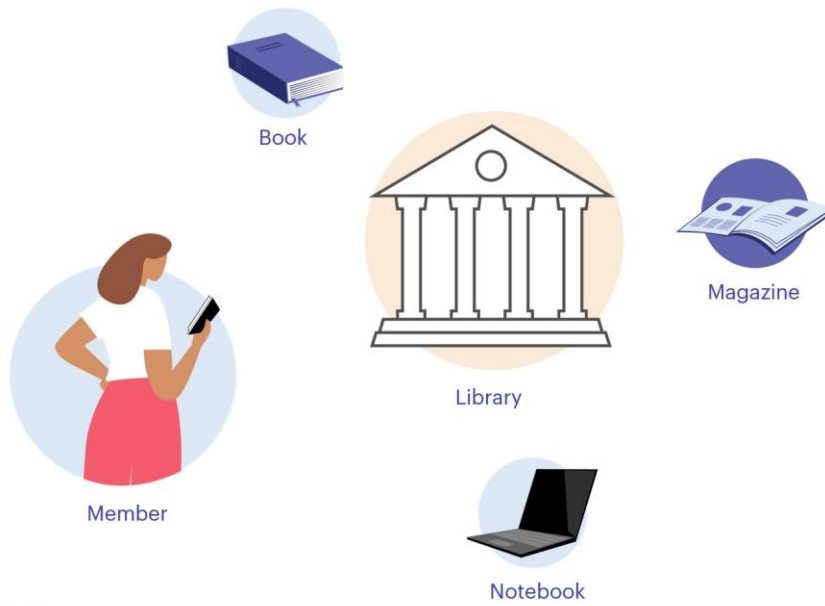


# Analysis of the Transaction Design Model

Library

**GeneXus**<sup>™</sup>

## Library



Throughout the previous course, we looked at everything we needed to correctly model a given reality in GeneXus. In this video, through the analysis of a limited reality, different options are analyzed to design transactions, using a set of essential resources and solving a series of real requirements.

Let's suppose that a GeneXus application needs to be designed for managing the tasks of a library related to the loans of books, magazines, and notebooks. It works with Members who can access different reading material, such as books and magazines, as well as computer equipment loaned by the Library.

## Library



- Id
- Name

Country



- Document
- Name
- Address
- Image
- Phone
- Over 20 years old

Member



- Id
- Name
- Image
- Country
- May have several books published

Author

The information to be recorded is as follows:

### Country

Each Country is registered with a unique identifier and its name.

### Member

Each Library Member is registered with his/her ID card, name, address, photo and a contact phone number.

Library members must be 20 years of age or older.

### Author

Each Author is registered with a unique identifier, name, photo and country of origin. An Author can have several books published

## Library



Country

- Id
- Name



Book

- Id
- Title
- Edition Date
- Copies
- Literary Genre
- Author
- Publishing House in charge of its publication



Member

- Document
- Name
- Address
- Image
- Phone
- Over 20 years old



Magazine

- Id
- Title
- Publication Date
- Image
- Copies



Author

- Id
- Name
- Image
- Country
- May have several books published



Notebook

- Id
- Image
- Description
- Status

**Book**

Each book is registered with a unique identifier, title, date of publication and number of copies available in the Library. It can be a Novel, an Essay, a Poetry book, etc.; therefore, a book belongs to a literary genre.

In addition, a Book has an Author and a Publishing House in charge of its publication.

**Magazine**

Each magazine is registered with a unique identifier, title, date of publication, cover image and the number of copies available.

**Notebook**

The Library's services include loans of notebooks to its Members, since many of them are writers and researchers. Each notebook is registered with a unique identifier, its image, and short description.

In addition, its status (available or checked out) is recorded.

## Library



Country

- Id
- Name



Book

- Id
- Title
- Edition Date
- Copies
- Literary Genre
- Author
- Publishing House in charge of its publication



Loan

- Id
- Date
- Member
- Return Date
- 15-day deadline
- may include a maximum of 3 books, 4 magazines, and may or may not include borrowing a laptop
- Comment



Member

- Document
- Name
- Address
- Image
- Phone
- Over 20 years old



Magazine

- Id
- Title
- Publication Date
- Image
- Copies



Book Request

- Id
- Date
- Publishing House
- Book
- Number of copies



Author

- Id
- Name
- Image
- Country
- May have several books published



Notebook

- Id
- Image
- Description
- Status

## Loan

The Library loans books, magazines, and notebooks to its Members.

Each loan is recorded with a unique identifier, the date of checkout, Member, and due date, which must be automatically determined.

All loans are for 15 days; they may include a maximum of 3 books, 4 magazines, and may or may not include the loan of a notebook.

Only one copy of each publication may be checked out, and it is possible to enter any comments considered necessary at the level of each copy's loan (e.g., damaged cover, missing pages, etc.).

In addition, the date on which a new loan is recorded is always the current date and it must not be possible to change it.

## Request for copies (BookRequest)

It often happens that certain books are in high demand, and the Library decides to request more copies.

To this end, it makes a Request to the corresponding Publishing House. The system must check that book copies are requested from the indicated Publishing House.

# Library



Country

- Id
- Name



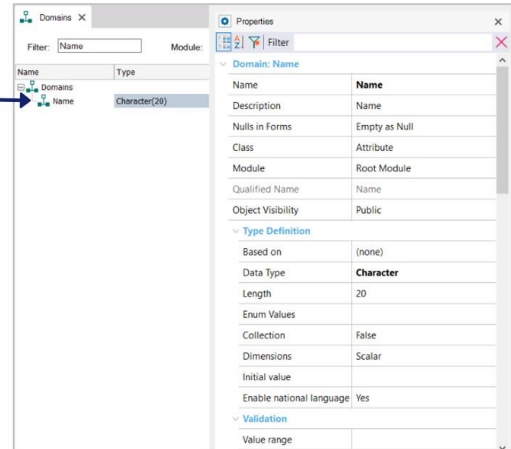
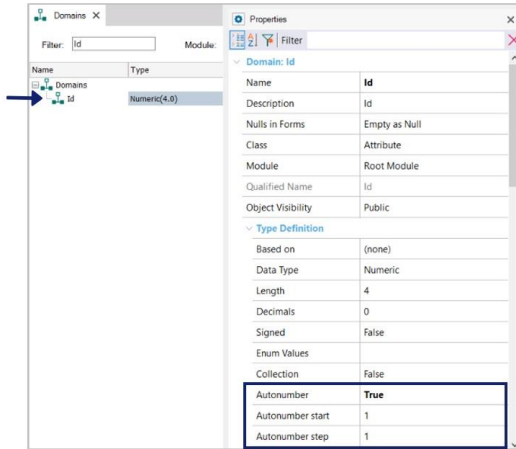
Member

- Document
- Name
- Address
- Image
- Phone
- Over 20 years old



Author

- Id
- Name
- Image
- Country
- May have several books published



Let's start to analyze this reality.

Initially, we can clearly distinguish certain simple entities that we can begin to define, such as, for example, Country, Member and Author.

But first, we should take into account that most of the entities are registered with an identifier that can be autoincremented, except for the Member Number, who as we mentioned before, is registered with his or her identity document. Therefore, we define the Domain ID, with the Autonumber property set to True.

We also define the Name domain, as a character of 20.

## Transaction definition: Country



- Id  
- Name

| Name        | Type    | Description  |
|-------------|---------|--------------|
| Country     | Country | Country      |
| CountryId   | Id      | Country Id   |
| CountryName | Name    | Country Name |

| Attribute       | Order       |
|-----------------|-------------|
| Country Indexes |             |
| ICountry        | Primary Key |
| • CountryId     | Ascending   |
| UCountry        | Unique      |
| • CountryName   | Ascending   |

Let's start by defining the Country transaction, with CountryId as primary key, and CountryName as secondary attribute.

CountryId is based on the ID domain, and to check that the name is not repeated, we define the corresponding unique index. This same definition is valid for all entities where it is necessary to check that the name is not repeated.

# Transaction definition: Country



- Id  
- Name

Weak 1-N

| Name        | Type    | Description  |
|-------------|---------|--------------|
| Country     | Country | Country      |
| CountryId   | Id      | Country Id   |
| CountryName | Name    | Country Name |
| City        | City    | City         |
| CityId      | Id      | City Id      |
| CityName    | Name    | City Name    |

| Attribute       | Order       |
|-----------------|-------------|
| Country Indexes |             |
| ICountry        | Primary Key |
| • CountryId     | Ascending   |
| UCountry        | Unique      |
| • CountryName   | Ascending   |

| Name                  | Type |
|-----------------------|------|
| CountryCity Structure |      |
| CountryId             | Id   |
| CityId                | Id   |
| • CityName            | Name |

If it were necessary to record the cities of each country, because we need to know the city of birth of the author, we could model it as a weak entity in relation to the country, since a city does not exist outside that context.

So a second level would be added to the Country transaction, but since City is considered a weak entity it will not exist as a transaction in itself.

This means that CityId will not exist as a primary key in any table, and therefore, in order to know, for example, the city of birth of an author, the pair made up of the attributes CountryId, CityId, which will be the primary key of the COUNTRYCITY table associated with the second level of the Country transaction, will be needed.

However, since the description of the reality we are analyzing does not include the city concept, we will model the Country as a simple transaction.



## Transaction definition: Member



- Document
- Name
- Address
- Image
- Phone
- Over 20 years old

| Name            | Type             | Description       | Formula                       |
|-----------------|------------------|-------------------|-------------------------------|
| Member          | Member           | Member            |                               |
| MemberDocument  | Numeric(8.0)     | Member Document   |                               |
| MemberName      | Name             | Member Name       |                               |
| MemberImage     | Image            | Member Image      |                               |
| MemberAddress   | Address, GeneXus | Member Address    |                               |
| MemberPhone     | Phone, GeneXus   | Member Phone      |                               |
| MemberBirthDate | Date             | Member Birth Date |                               |
| MemberAge       | Numeric(3.0)     | Member Age        | age(MemberBirthDate, today()) |

```

Member X
Structure | Web Layout | Win Form | Rules | Events | Variables | Help | Documentation
1 Error("The member must be over 20 years old")
2   if MemberAge <= 20;

```

Let's move on to Member. For this we define the Member transaction, with the following attributes:

- MemberDocument, which corresponds to the ID card and therefore is not autonumbered, so we define it as an 8-digit numeric value.
- MemberName, of Name type.
- MemberImage, of Image type.
- MemberAddress, based on the Address semantic domain.
- and MemberPhone, of Phone type.

But, in addition, reality tells us that members must be over 20 years of age, so we need their date of birth and age, which should be calculated automatically. So, to the transaction structure we add these attributes:

- MemberBirthDate, of Date type
- and MemberAge, a numeric value of 3 digits.

How do we calculate the **Member's** age? With the Age function, which calculates the age from the date of birth and the current date.

So, we define MemberAge as a calculated attribute that obtains its value from the following expression:

Age(MemberBirthDate, Today())

Why do we use the Today() function instead of the &today variable?

Because it's not possible to use variables in the formula declaration.

To check that every member is older than 20, it is enough to declare the following Error rule:

**Error("The member must be over 20 years old")** if MemberAge <= 20;

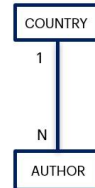
It should be noted that we are not considering in this analysis the declaration of rules for controlling basic data entry, such as, for example, controlling that a member is not entered without a name, etc.

# Transaction definition: Author



- Id
- Name
- Image
- Country
- May have several books published

| Name        | Type   | Description  |
|-------------|--------|--------------|
| Author      | Author | Author       |
| AuthorId    | Id     | Author Id    |
| AuthorName  | Name   | Author Name  |
| AuthorImage | Image  | Author Image |
| CountryId   | Id     | Country Id   |
| CountryName | Name   | Country Name |



Now **let's** consider the Authors. We need to create the Author transaction, with the following attributes:

- AuthorId
- AuthorName
- AuthorImage

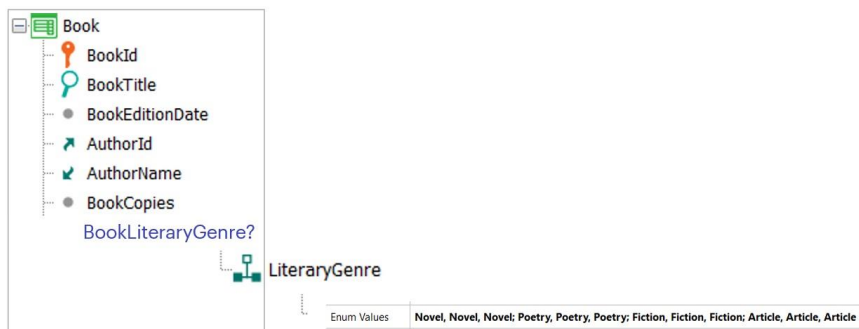
Also, we know that the country of the Author must be registered, since every Author has a country of birth; therefore, we add CountryId and CountryName, where CountryId is a foreign key and CountryName is an attribute that is inferred from that foreign key.

In this way, we represent a 1-N relationship between Country and Author.

## Transaction definition: Book



- Id
- Title
- Edition Date
- Copies
- Literary Genre
- Author
- Publishing House in charge of its publication



Now **let's** think of the concept of Book. It is a strong entity that is also identified with an autonumbered value, a title, a date of publication, an author, and the number of copies purchased by the Library.

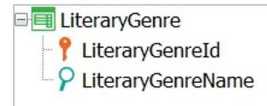
But, in addition, reality tells us that a book belongs to a literary genre, since it can be a Novel, an Essay, etc.

So, how do we model the concept of literary genre? If we think that these genres are finite, a first option we can consider is the creation of an enumerated domain.

## Transaction definition: Book



- Id
- Title
- Edition Date
- Copies
- Literary Genre
- Author
- Publishing House in charge of its publication



| Attribute             | Order       |
|-----------------------|-------------|
| LiteraryGenre Indexes |             |
| ILiteraryGenre        | Primary Key |
| LiteraryGenreId       | Ascending   |
| ULiteraryGenre        | Unique      |
| LiteraryGenreName     | Ascending   |



| Attribute               | Order       |
|-------------------------|-------------|
| PublishingHouse Indexes |             |
| IPublishingHouse        | Primary Key |
| PublishingHouseId       | Ascending   |
| UPublishingHouse        | Unique      |
| PublishingHouseName     | Ascending   |

But the categorization of literary genres has been changing over time and is likely to continue to do so, so the enumerated domain doesn't seem to be the best option, as changes would have to be made manually and it would not be scalable.

We can consider a LiteraryGenre transaction with these attributes:

- LiteraryGenreId
- and LiteraryGenreName.

It is a good decision to define a unique index on the name of the genre in order to avoid the registration of literary genres with the same name.

But the Book also requires registering the Author and the Publishing House responsible for its publication. The Author is already defined as an entity, but the Publishing House is not. And although it is not explicitly declared, in the analysis the need to model the Publishing House entity arises.

So, we define the PublishingHouse transaction with these attributes:

- PublishingHouseId
- and PublishingHouseName

We can also create the corresponding unique index on the name of the publisher.

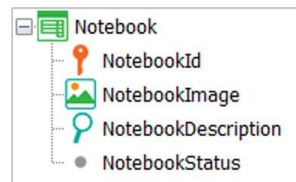
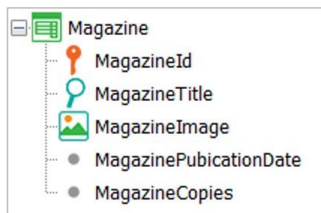
## Transaction definition: Magazine and Notebook



- Id
- Title
- Publication Date
- Image
- Copies



- Id
- Image
- Description
- Status



| Name    | Type          |
|---------|---------------|
| Domains |               |
| Status  | Character(20) |

|             |   |
|-------------|---|
| Enum Values | <b>OnLoan, On loan, OnLoan: Available, Available, Available</b> |
|-------------|---|

The Library also offers Magazines and Notebooks to its members, so we define the Magazine transaction, with the attributes:

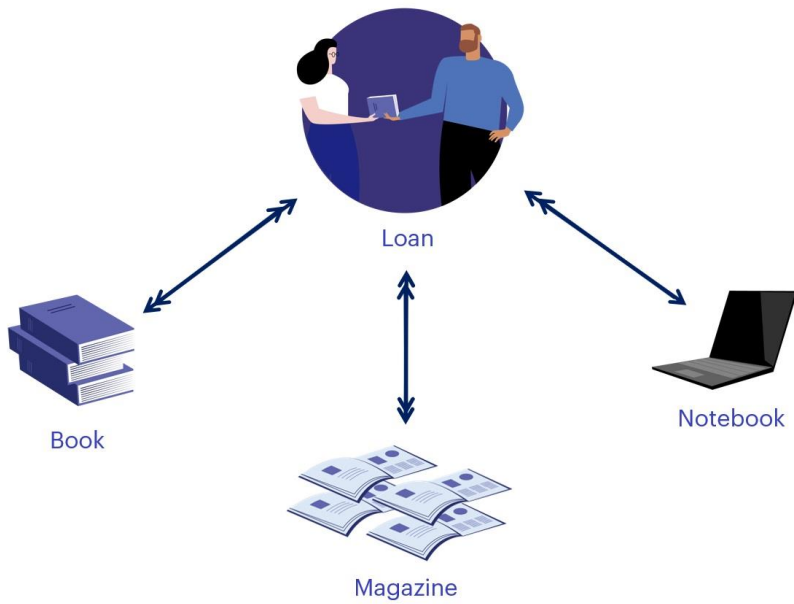
- MagazineId
- MagazineTitle
- MagazineImage
- MagazinePublicationDate
- and MagazineCopies to record the number of copies purchased.

In addition, there is the Notebook transaction with these attributes:

- NotebookId
- NotebookImage
- NotebookDescription of LongVarChar type
- and NotebookStatus.

A notebook is “Available” or “Checked out”, so we define the Status enumerated domain with these values.

## Transaction definition: Loan



Now **let's** analyze the concept of Loan, which is essential in this reality.

The Library allows a member to borrow a notebook, 3 books and 4 magazines for 15 days.

We then have a 1-N relationship between the entities Loan and Notebook, and an N-N relationship between Loan and Book, and between Loan and Magazine.

How can we model it?

## Transaction definition: Loan



| Name                | Type             | Formula              | Nullable |
|---------------------|------------------|----------------------|----------|
| Loan                | Loan             |                      |          |
| LoanId              | Id               |                      | No       |
| LoanDate            | Date             |                      | No       |
| MemberDocument      | Numeric(8.0)     |                      | No       |
| MemberName          | Name             |                      |          |
| MemberAddress       | Address, GeneXus |                      |          |
| LoanReturnDate      | Date             | LoanDate.adddays(15) |          |
| NotebookId          | Id               |                      | Yes      |
| NotebookDescription | Character(40)    |                      |          |

Let's analyze a first option:

We create the Loan transaction and define the following attributes:

- LoanId
- LoanDate
- MemberDocument
- MemberName
- MemberAddress
- LoanReturnDate

The return date is requested to be calculated automatically. A loan is made for 15 days, so we declare the following calculation associated with the attribute LoanReturnDate:

LoanDate.AddDays(15)

In addition, a loan may or may not include a notebook, so we add:

- NotebookId
- and NotebookDescription.

But in this Loan transaction, NotebookId is a non-mandatory foreign key; therefore, we set its Nullable property to Yes.



## Transaction definition: Loan



| Name                | Type             | Formula                | Nullable |
|---------------------|------------------|------------------------|----------|
| Loan                | Loan             |                        |          |
| LoanId              | Id               |                        | No       |
| LoanDate            | Date             |                        | No       |
| MemberDocument      | Numeric(8.0)     |                        | No       |
| MemberName          | Name             |                        |          |
| MemberAddress       | Address, GeneXus |                        |          |
| LoanReturnDate      | Date             | LoanDate.adddays(15)   |          |
| NotebookId          | Id               |                        | Yes      |
| NotebookDescription | Character(40)    |                        |          |
| LoanBooksQty        | Numeric(4.0)     | count(LoanBookComment) |          |
| Book                | Book             |                        |          |
| BookId              | Id               |                        |          |
| BookTitle           | Character(20)    |                        |          |
| LoanBookComment     | Character(40)    |                        |          |

```
Error("Only 3 books can be checked out")
if LoanBooksQty > 3;
```

A loan includes several books, with a maximum of 3. So we define a second level to record them, including a LoanBookComment attribute, to record any comments needed at the time of the loan.

How do we control that no more than 3 books are entered?

We can define a new LoanBooksQty attribute that counts that amount and then condition that value in an Error rule:

**Error("Only 3 books can be checked out")** if LoanBooksQty > 3;

# Transaction definition: Loan



| Name                | Type             | Formula                    | Nullable |
|---------------------|------------------|----------------------------|----------|
| <b>Loan</b>         | <b>Loan</b>      |                            |          |
| LoanId              | Id               |                            | No       |
| LoanDate            | Date             |                            | No       |
| MemberDocument      | Numeric(8.0)     |                            | No       |
| MemberName          | Name             |                            |          |
| MemberAddress       | Address, GeneXus |                            |          |
| LoanReturnDate      | Date             | LoanDate.adddays(15)       |          |
| NotebookId          | Id               |                            | Yes      |
| NotebookDescription | Character(40)    |                            |          |
| LoanBooksQty        | Numeric(4.0)     | count(LoanBookComment)     |          |
| LoanMagazinesQty    | Numeric(4.0)     | count(LoanMagazineComment) |          |
| <b>Book</b>         | <b>Book</b>      |                            |          |
| BookId              | Id               |                            |          |
| BookTitle           | Character(20)    |                            |          |
| LoanBookComment     | Character(40)    |                            |          |
| <b>Magazine</b>     | <b>Magazine</b>  |                            |          |
| MagazineId          | Id               |                            |          |
| MagazineTitle       | Character(20)    |                            |          |
| LoanMagazineComment | Character(40)    |                            |          |

```
Error("Only 3 books can be checked out")  
  if LoanBooksQty > 3;  
  
Error("Only 4 magazines can be checked out")  
  if LoanMagazinesQty > 4;
```

But the same loan can include up to 4 magazines, so we can define another second level, parallel to the book, to register the magazines. It is also possible to control that no more than 4 are checked out.

## Transaction definition: Loan



| Name                | Type             | Formula                    | Nullable |
|---------------------|------------------|----------------------------|----------|
| Loan                | Loan             |                            |          |
| LoanId              | Id               |                            | No       |
| LoanDate            | Date             |                            | No       |
| MemberDocument      | Numeric(8.0)     |                            | No       |
| MemberName          | Name             |                            |          |
| MemberAddress       | Address, GeneXus |                            |          |
| LoanReturnDate      | Date             | LoanDate.adddays(15)       |          |
| NotebookId          | Id               |                            | Yes      |
| NotebookDescription | Character(40)    |                            |          |
| LoanBooksQty        | Numeric(4.0)     | count(LoanBookComment)     |          |
| LoanMagazinesQty    | Numeric(4.0)     | count(LoanMagazineComment) |          |
| Book                | Book             |                            |          |
| BookId              | Id               |                            |          |
| BookTitle           | Character(20)    |                            |          |
| LoanBookComment     | Character(40)    |                            |          |
| Magazine            | Magazine         |                            |          |
| MagazineId          | Id               |                            |          |
| MagazineTitle       | Character(20)    |                            |          |
| LoanMagazineComment | Character(40)    |                            |          |

```

Error("Only 3 books can be checked out")
  if LoanBooksQty > 3;

Error("Only 4 magazines can be checked out")
  if LoanMagazinesQty > 4;

Default(LoanDate, today());
noaccept(LoanDate);

```

The reality clearly describes that the registration date of a loan must always be the current date, with no possibility of being modified.

To this end, we declare the following rules:

```

Default(LoanDate, today());
noaccept(LoanDate);

```

## Transaction definition: Loan



| Name                | Type             | Formula                    | Nullable |
|---------------------|------------------|----------------------------|----------|
| Loan                | Loan             |                            |          |
| LoanId              | Id               |                            | No       |
| LoanDate            | Date             |                            | No       |
| MemberDocument      | Numeric(8,0)     |                            | No       |
| MemberName          | Name             |                            |          |
| MemberAddress       | Address, GeneXus |                            |          |
| LoanReturnDate      | Date             | LoanDate.adddays(15)       |          |
| NotebookId          | Id               |                            | Yes      |
| LoanBooksQty        | Numeric(4,0)     | count(LoanBookComment)     |          |
| LoanMagazinesQty    | Numeric(4,0)     | count(LoanMagazineComment) |          |
| Book                | Book             |                            |          |
| BookId              | Id               |                            |          |
| BookTitle           | Character(20)    |                            |          |
| LoanBookComment     | Character(40)    |                            |          |
| Magazine            | Magazine         |                            |          |
| MagazineId          | Id               |                            |          |
| MagazineTitle       | Character(20)    |                            |          |
| LoanMagazineComment | Character(40)    |                            |          |

| Control Info            |                                    |
|-------------------------|------------------------------------|
| Control Type            | Dynamic Combo Box                  |
| Data Source From        | Attributes                         |
| Item Values             | NotebookId                         |
| Item Descriptions       | NotebookDescription                |
| Sort Descriptions       | True                               |
| Conditions              | NotebookStatus = Status.Available; |
| Instantiated Attributes |                                    |
| Empty Item              | True                               |



Although the objective of this video is to focus on analyzing the transaction design, we are going to suggest an implementation option so that, when registering a loan, only notebooks that have Available status are offered. This implementation can be done from the definition of the transaction itself and that is why we include it.

We are going to remove the NotebookDescription attribute, because we will “disguise” the identifier of a notebook with its description. We select NotebookId and define it as a Dynamic Combo Box, with NotebookId in the property Item Values, and NotebookDescription in the property Item Descriptions.

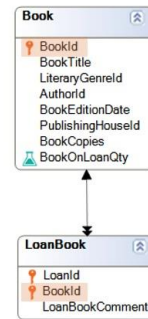
This will load all the notebooks registered. So that only those with available status are offered, we declare the following condition:  
 NotebookStatus = Status.Available;

Also, we must configure the Empty Item property to True, since NotebookId may not be chosen because it can be null.

Although this is not the only way to control that a notebook is available when registering a loan, it is an implementation that can be solved from the definition of the transaction itself.

## Available quantities of Books and Magazines

| Name                | Type          | Formula                |
|---------------------|---------------|------------------------|
| <b>Book</b>         | <b>Book</b>   |                        |
| BookId              | Id            |                        |
| BookTitle           | Character(20) |                        |
| BookEditionDate     | Date          |                        |
| AuthorId            | Id            |                        |
| AuthorName          | Name          |                        |
| BookCopies          | Numeric(4.0)  |                        |
| LiteraryGenreId     | Id            |                        |
| LiteraryGenreName   | Name          |                        |
| PublishingHouseId   | Id            |                        |
| PublishingHouseName | Name          |                        |
| BookOnLoanQty       | Numeric(4.0)  | count(LoanBookComment) |



| Name                    | Type            | Formula |
|-------------------------|-----------------|---------|
| <b>Magazine</b>         | <b>Magazine</b> |         |
| MagazineId              | Id              |         |
| MagazineTitle           | Character(20)   |         |
| MagazineImage           | Image           |         |
| MagazinePublicationDate | Date            |         |
| MagazineCopies          | Numeric(4.0)    |         |

Another necessary requirement is to know the availability of books or magazines in high demand.

We know the number of copies purchased by the Library for each book and magazine, but we don't know how many are still available.

Can we obtain these values in a simple way and based on the transaction design?

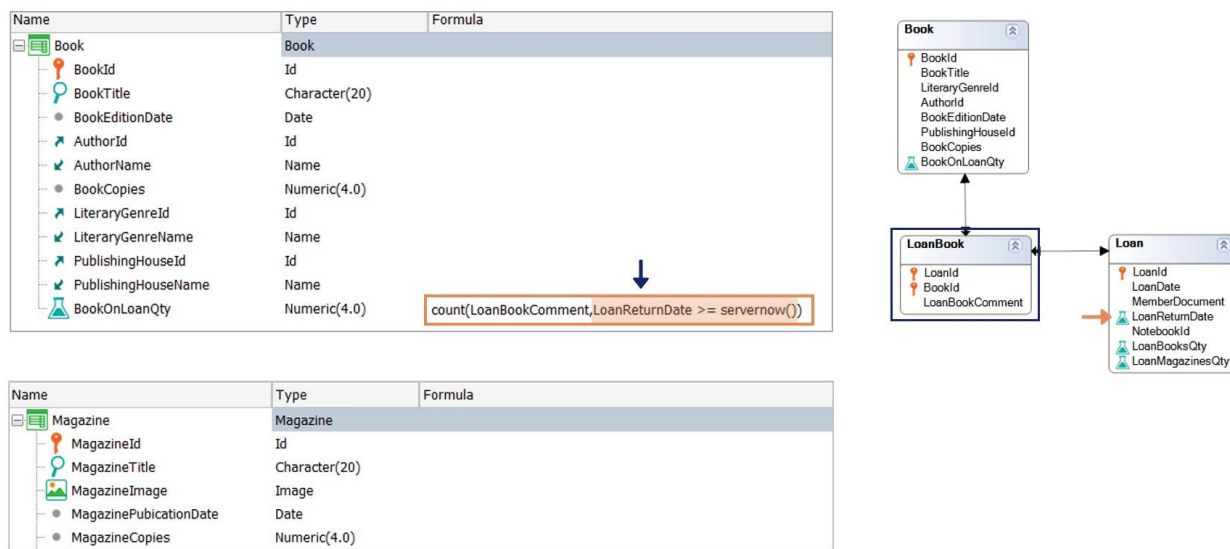
If in the Book transaction we define a new attribute, BookOnLoanQty, we can calculate the number of copies borrowed and therefore know the number of copies available.

What happens if we associate the Count(LoanBookComment) calculation with this new BookOnLoanQty attribute? Will it effectively sum the number of copies of that book that have been checked out?

The answer is YES, because the table associated with the transaction where we define the calculation is BOOK. And the table where the calculation is resolved is LOANBOOK, since it is determined by the LoanBookComment attribute.

Between both tables there is an attribute in common which is BookId, and therefore it is an implicit filter that GeneXus will apply; that is to say, that this calculation will return the total number of copies of that book that has been loaned.

## Available quantities of Books and Magazines



But we need to know the number of copies currently checked out, so we have to add a condition to the calculation that allows counting the number of copies currently checked out. For this purpose, we consider loans whose return date is greater than or equal to the current date.

We can state the following:

`Count(LoanBookComment, LoanReturnDate >= servernow())`

The `servernow()` function allows obtaining the server's current date and time. The `Today()` function could also be used.

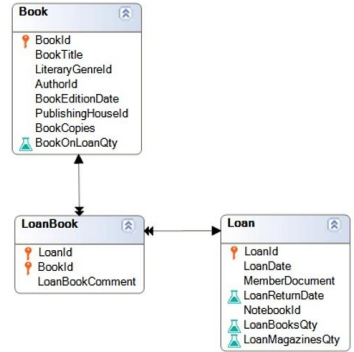
Can we declare this calculation condition? Yes, because the attribute involved (`LoanReturnDate`) belongs to the extended table of the table where the declared calculation is resolved, which is `LOANBOOK`.

## Available quantities of Books and Magazines

| Name                | Type          | Formula   |
|---------------------|---------------|---|
| <b>Book</b>         | <b>Book</b>   |   |
| BookId              | Id            |   |
| BookTitle           | Character(20) |   |
| BookEditionDate     | Date          |   |
| AuthorId            | Id            |   |
| AuthorName          | Name          |   |
| BookCopies          | Numeric(4,0)  |   |
| LiteraryGenreId     | Id            |   |
| LiteraryGenreName   | Name          |   |
| PublishingHouseId   | Id            |   |
| PublishingHouseName | Name          |   |
| BookOnLoanQty       | Numeric(4,0)  | count(LoanBookComment, LoanReturnDate >= servernow()) |
| BookAvailableQty    | Numeric(4,0)  | BookCopies - BookOnLoanQty                            |

| Name                    | Type            | Formula |
|-------------------------|-----------------|---------|
| <b>Magazine</b>         | <b>Magazine</b> |         |
| MagazineId              | Id              |         |
| MagazineTitle           | Character(20)   |         |
| MagazineImage           | Image           |         |
| MagazinePublicationDate | Date            |         |
| MagazineCopies          | Numeric(4,0)    |         |

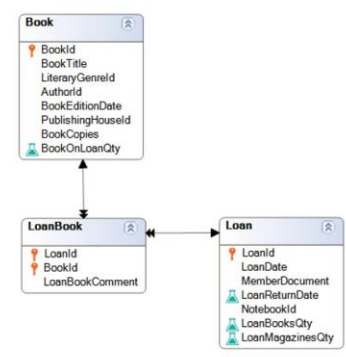


Therefore, if we know the total number of copies, and the number of copies currently checked out, then we can know the number of copies currently available.

So the structure of the Book transaction looks as follows:

# Available quantities of Books and Magazines

| Name                | Type          | Formula   |
|---------------------|---------------|---|
| <b>Book</b>         |               |   |
| BookId              | Id            |   |
| BookTitle           | Character(20) |   |
| BookEditionDate     | Date          |   |
| AuthorId            | Id            |   |
| AuthorName          | Name          |   |
| BookCopies          | Numeric(4.0)  |   |
| LiteraryGenreId     | Id            |   |
| LiteraryGenreName   | Name          |   |
| PublishingHouseId   | Id            |   |
| PublishingHouseName | Name          |   |
| BookOnLoanQty       | Numeric(4.0)  | count(LoanBookComment, LoanReturnDate >= servernow()) |
| BookAvailableQty    | Numeric(4.0)  | BookCopies - BookOnLoanQty                            |



| Name                    | Type          | Formula   |
|-------------------------|---------------|---|
| <b>Magazine</b>         |               |   |
| MagazineId              | Id            |   |
| MagazineTitle           | Character(20) |   |
| MagazineImage           | Image         |   |
| MagazinePublicationDate | Date          |   |
| MagazineCopies          | Numeric(4.0)  |   |
| MagazineOnLoanQty       | Numeric(4.0)  | count(LoanMagazineComment, LoanReturnDate >= servernow()) |
| MagazineAvailableQty    | Numeric(4.0)  | MagazineCopies - MagazineOnLoanQty                        |

In the same way, we can know the number of available copies of a given magazine:



## Transaction definition: Loan

| Name                 | Type             | Formula                            |
|----------------------|------------------|------------------------------------|
| Loan                 | Loan             |                                    |
| LoanId               | Id               |                                    |
| LoanDate             | Date             |                                    |
| MemberDocument       | Numeric(8.0)     |                                    |
| MemberName           | Name             |                                    |
| MemberAddress        | Address, GeneXus |                                    |
| LoanReturnDate       | Date             | LoanDate.adddays(15)               |
| NotebookId           | Id               |                                    |
| LoanBooksQty         | Numeric(4.0)     | count(LoanBookComment)             |
| LoanMagazinesQty     | Numeric(4.0)     | count(LoanMagazineComment)         |
| Book                 | Book             |                                    |
| BookId               | Id               |                                    |
| BookTitle            | Character(20)    |                                    |
| LoanBookComment      | Character(40)    |                                    |
| BookAvailableQty     | Numeric(4.0)     | BookCopies - BookOnLoanQty         |
| Magazine             | Magazine         |                                    |
| MagazineId           | Id               |                                    |
| MagazineTitle        | Character(20)    |                                    |
| LoanMagazineComment  | Character(40)    |                                    |
| MagazineAvailableQty | Numeric(4.0)     | MagazineCopies - MagazineOnLoanQty |

```

Error("Only 3 books can be checked out")
  if LoanBooksQty > 3;

Error("Only 4 magazines can be checked out")
  if LoanMagazinesQty > 4;

Default(LoanDate, today());

noaccept(LoanDate);

Error("There are no available copies of this book")
  if BookAvailableQty < 0;

Error("There are no available copies of this magazine")
  if MagazineAvailableQty < 0;

```

Now we have all the information we need to validate a loan, so let's add the necessary attributes to validate the availability of the copies to be borrowed:

- BookAvailableQty, at the book level, and
- MagazineAvailableQty, at the magazine level.

And we control with the following Error rules:

## Loan transaction: Form and Table structure

The image displays the GeneXus design tool interface for a Loan transaction. On the left, a form is shown with fields for: Id (LoanId), Date (LoanDate), Member Document (MemberDocument), Member Name (MemberName), Member Address (MemberAddress), Return Date (LoanReturnDate), Notebook Description (NotebookId), Books Qty (LoanBooksQty), and Magazines Qty (LoanMagazinesQty). Below these fields are two parallel grids: 'Book' and 'Magazine'. The 'Book' grid has columns for Book Id (BookId), Book Title (BookTitle), Book Comment (LoanBookComment), and Book Available Qty (BookAvailableQty). The 'Magazine' grid has columns for Magazine Id (MagazineId), Magazine Title (MagazineTitle), Magazine Comment (LoanMagazineComment), and Magazine Available Qty (MagazineAvailableQty). On the right, a tree view shows the 'Loan' entity with 'Associated Tables' including Loan, LoanBook, and LoanMagazine. Below the tree, three table diagrams are shown: 'Loan' with fields LoanId, LoanDate, MemberDocument, LoanReturnDate, NotebookId, LoanBooksQty, and LoanMagazinesQty; 'LoanBook' with fields LoanId, BookId, and LoanBookComment; and 'LoanMagazine' with fields LoanId, MagazineId, and LoanMagazineComment.

So far we have met the requirements to register loans. But let's look at the form generated for the design we have made.

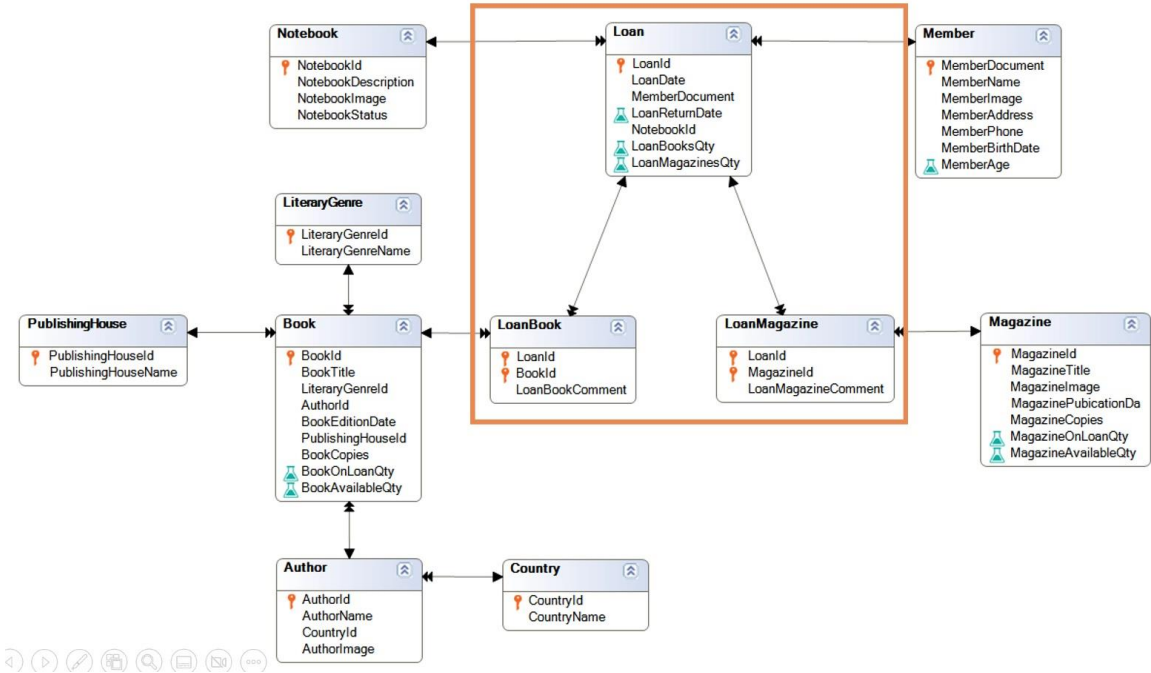
What associated tables are created by GeneXus for this Loan transaction?

It creates 3 tables:

- LOAN, associated with the first level, with LoanId as PK.
- LOANBOOK, associated with the book level, with LoanId and BookId as compound PK.
- LOANMAGAZINE, associated with the magazine level, with LoanId and MagazineId as compound PK.

By defining two parallel levels, we see parallel grids in the form. This may be cumbersome for the end user, who may request to simplify the design, and manage the information in different, simpler screens.

# Table Diagram

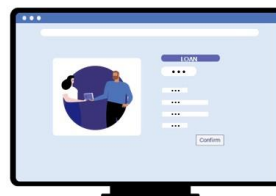


For the current design, GeneXus has created the following tables. In particular, let's look at the LOAN, LOANBOOK and LOANMAGAZINE tables, corresponding to the tables associated with the three levels of the Loan transaction defined.

Can we propose another design for the Loan transaction? Yes.

## Loan entity – Option 2: Three transactions

- A screen for recording general loan information



- A screen for recording the books borrowed

- A screen for recording the magazines borrowed



So, **let's** see a second option to model the Loan.

Let's suppose that the end user has asked us to “**streamline**” or “**lighten**” the loading of this screen, and display the information on three screens:

- a screen for registering the general information of the loan, in which the loan is registered with its date, the member's data, and whether a notebook is included or not.
- a screen for registering the books loaned, in which the books are registered with the necessary controls already defined.
- and a screen for registering the magazines loaned, in which the magazines are registered with the necessary controls already defined.

How can we do so?

## Loan entity – Option 2: Three transactions

| Name           | Type             | Formula              | Nullable |
|----------------|------------------|----------------------|----------|
| Loan           | Loan             |                      |          |
| LoanId         | Id               |                      | No       |
| LoanDate       | Date             |                      | No       |
| MemberDocument | Numeric(8.0)     |                      | No       |
| MemberName     | Name             |                      |          |
| MemberAddress  | Address, GeneXus |                      |          |
| LoanReturnDate | Date             | LoanDate.adddays(15) |          |
| NotebookId     | Id               |                      | Yes      |

```
Default(LoanDate, today());
```

```
noaccept(LoanDate);
```

Associated Tables:



Loan

<ErrorViewer: ErrorViewer>

---

<Toolbar>

Id

Date

Member Document

Member Name

Member Address

Return Date

Notebook Description

---

<FormButtons>

The first screen corresponds to the general loan data, so we delete the Loan transaction levels, and leave the general information, with the following rules.

The following form is generated:

## Loan entity – Option 2: Three transactions

| Name             | Type             | Formula                    | Nullable |
|------------------|------------------|----------------------------|----------|
| <b>BookLoan</b>  | <b>BookLoan</b>  |                            |          |
| LoanId           | Id               |                            | No       |
| LoanDate         | Date             |                            | No       |
| MemberDocument   | Numeric(8.0)     |                            | No       |
| MemberName       | Name             |                            |          |
| MemberAddress    | Address, GeneXus |                            |          |
| LoanReturnDate   | Date             | LoanDate.adddays(15)       |          |
| NotebookId       | Id               |                            | No       |
| LoanBooksQty     | Numeric(4.0)     | count(LoanBookComment)     |          |
| <b>Book</b>      | <b>Book</b>      |                            |          |
| BookId           | Id               |                            | No       |
| BookTitle        | Character(20)    |                            |          |
| LoanBookComment  | Character(40)    |                            | No       |
| BookAvailableQty | Numeric(4.0)     | BookCopies - BookOnLoanQty |          |

Associated Tables:

- Loan
- BookLoanBook

```

Default(LoanDate, today());
noaccept(LoanDate);

Error("Only 3 books can be checked out")
if LoanBooksQty > 3;

Error("There are no available copies of this book")
if BookAvailableQty < 0;
    
```

To provide another screen with the book register, we need a new transaction. We call it BookLoan.

We want to enter the loan identifier, view its general information, and enter the books. Therefore, the primary key will also be LoanId, and thus a new table will not be created because there is already one where LoanId is the primary key. This way we will be referencing the same Loan entity and only the table associated with the second level will be created.

The BookLoan transaction then has the following structure, with the following rules.

The following form is generated:

## Loan entity – Option 2: Three transactions

| Name                 | Type             | Formula                            | Nullable |
|----------------------|------------------|------------------------------------|----------|
| MagazineLoan         | MagazineLoan     |                                    |          |
| LoanId               | Id               |                                    | No       |
| LoanDate             | Date             |                                    | No       |
| MemberDocument       | Numeric(8,0)     |                                    | No       |
| MemberName           | Name             |                                    | No       |
| MemberAddress        | Address, GeneXus |                                    | No       |
| LoanReturnDate       | Date             | LoanDate.adddays(15)               | No       |
| NotebookId           | Id               |                                    | Yes      |
| LoanMagazinesQty     | Numeric(4,0)     | count(LoanMagazineComment)         | No       |
| Magazine             | Magazine         |                                    |          |
| MagazineId           | Id               |                                    | No       |
| MagazineTitle        | Character(20)    |                                    | No       |
| LoanMagazineComment  | Character(40)    |                                    | No       |
| MagazineAvailableQty | Numeric(4,0)     | MagazineCopies - MagazineOnLoanQty | No       |

<Toolbar>

Id

Date

Member Document

Member Name

Member Address

Return Date

Notebook Description

Magazines Qty

Magazine

| GRID | Magazine Id | Magazine Title | Magazine Comment    | Magazine Available Qty |
|------|-------------|----------------|---------------------|------------------------|
|      | MagazineId  | MagazineTitle  | LoanMagazineComment | MagazineAvailableQty   |

<FormButtons>

Associated Tables:

- Loan
- MagazineLoanMagazine

```

Default(LoanDate, today());
noaccept(LoanDate);

Error("Only 4 magazines can be checked out")
if LoanMagazinesQty > 4;

Error("There are no available copies of this magazine")
if MagazineAvailableQty < 0;
    
```

We will do something similar to record the loans of magazines. We create the MagazineLoan transaction with the following structure, and the following rules:

## Loan entity – Option 2: Three transactions

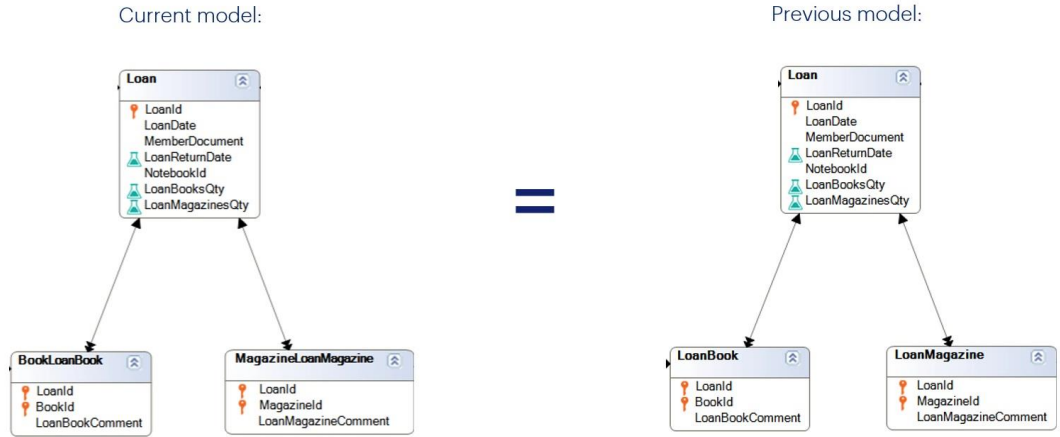


Before analyzing the generated table diagram, let's look at the tables associated with the transactions created in this proposal. GeneXus also creates 3 tables:

- LOAN
- BOOKLOANBOOK
- and MAGAZINELOANMAGAZINE



# Table Diagram

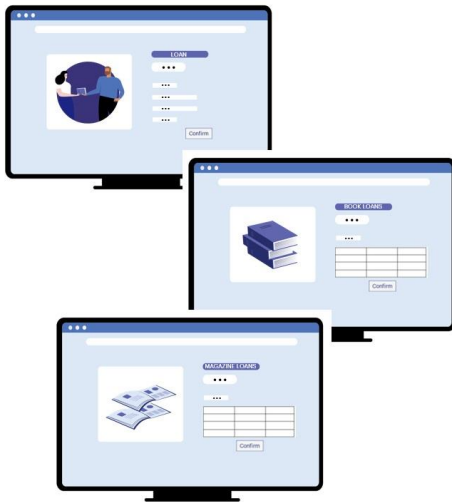


Let's look at the table diagram.

If we look at the LOAN, BOOKLOANBOOK and MAGAZINELOANMAGAZINE tables, we can see that they have exactly the same structure as those generated in the previously proposed design.

## Table Diagram

Current model:



Previous model:



This means that the difference between both designs is not in the structures generated but in the screens offered to the end user for managing the information. It will always be up to the end users to tell us which design is best suited to their business.

From the database structure point of view, it is exactly the same.

## Loan entity – Option 3: Two transactions



But we can also propose a third option for Loan modeling.

Perhaps we could consider that the Library bases the loans on books, and the end user requests a single screen where the general data of the loan and also of the books to be checked out are entered. Then, from another screen, the magazines included in the loan are registered.


# Loan entity – Option 3: Two transactions

| Name             | Type             | Formula                    | Nullable |
|------------------|------------------|----------------------------|----------|
| Loan             | Loan             |                            |          |
| LoanId           | Id               |                            | No       |
| LoanDate         | Date             |                            | No       |
| MemberDocument   | Numeric(8.0)     |                            | No       |
| MemberName       | Name             |                            |          |
| MemberAddress    | Address, GeneXus |                            |          |
| LoanReturnDate   | Date             | LoanDate.adddays(15)       |          |
| NotebookId       | Id               |                            | Yes      |
| LoanBooksQty     | Numeric(4.0)     | count(LoanBookComment)     |          |
| Book             | Book             |                            |          |
| BookId           | Id               |                            | No       |
| BookTitle        | Character(20)    |                            |          |
| LoanBookComment  | Character(40)    |                            | No       |
| BookAvailableQty | Numeric(4.0)     | BookCopies - BookOnLoanQty |          |

Associated Tables:

- Loan
- LoanBook

```
Default(LoanDate, today());  
noaccept(LoanDate);  
Error("Only 3 books can be checked out")  
  if LoanBooksQty > 3;  
Error("There are no available copies of this book")  
  if BookAvailableQty < 0;
```



For this purpose, we consider the Loan transaction with a second level to record the books. Its structure would be as follows, with the following rules:

## Loan entity – Option 3: Two transactions

| Name             | Type             | Formula                    | Nullable |
|------------------|------------------|----------------------------|----------|
| Loan             | Loan             |                            |          |
| LoanId           | Id               |                            | No       |
| LoanDate         | Date             |                            | No       |
| MemberDocument   | Numeric(8,0)     |                            | No       |
| MemberName       | Name             |                            |          |
| MemberAddress    | Address, GeneXus |                            |          |
| LoanReturnDate   | Date             | LoanDate.adddays(15)       |          |
| NotebookId       | Id               |                            | Yes      |
| LoanBooksQty     | Numeric(4,0)     | count(LoanBookComment)     |          |
| Book             | Book             |                            |          |
| BookId           | Id               |                            | No       |
| BookTitle        | Character(20)    |                            |          |
| LoanBookComment  | Character(40)    |                            | No       |
| BookAvailableQty | Numeric(4,0)     | BookCopies - BookOnLoanQty |          |

Associated Tables:

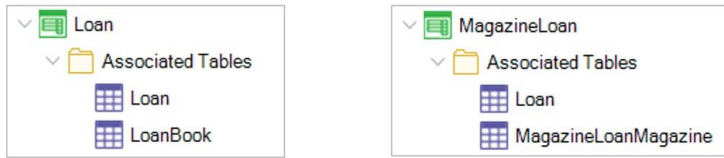
-  Loan
-  LoanBook

```

Default(LoanDate, today());
noaccept(LoanDate);
Error("Only 3 books can be checked out")
  if LoanBooksQty > 3;
Error("There are no available copies of this book")
  if BookAvailableQty < 0;
    
```

For registering the magazines included in the loan, we have the MagazineLoan transaction with the following structure, and the following rules:

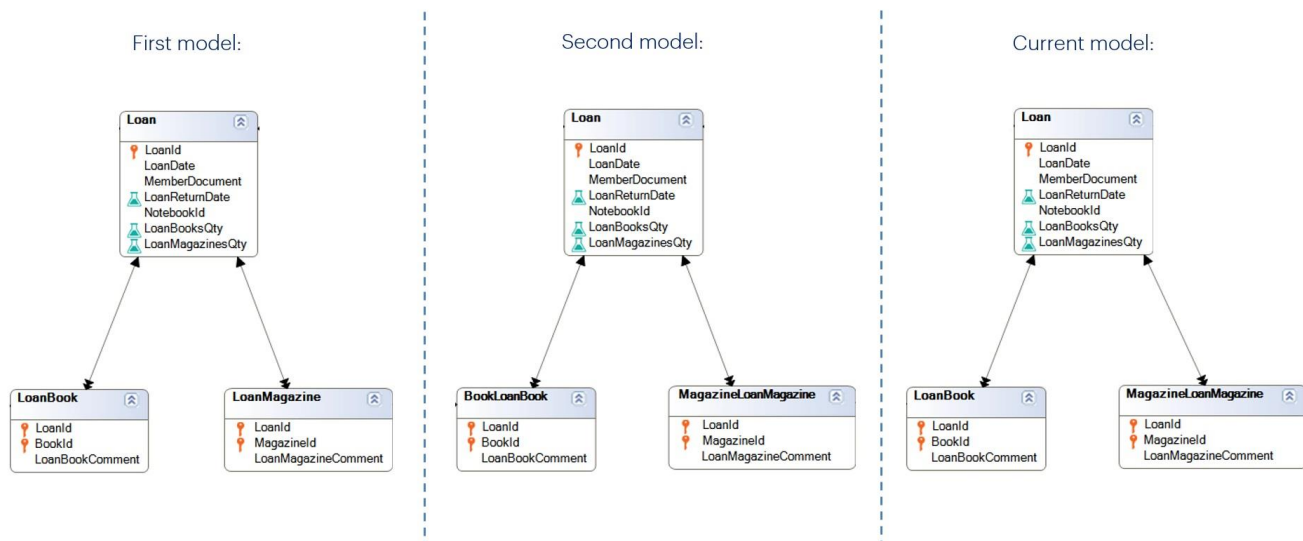
## Loan entity – Option 3: Two transactions



What tables does GeneXus create in relation to these transactions? 3 tables: LOAN, LOANBOOK and MAGAZINELOANMAGAZINE.

- LOAN and LOANBOOK associated with the Loan transaction
- LOAN and MAGAZINELOANMAGAZINE associated with the MagazineLoan transaction.

## Table Diagram



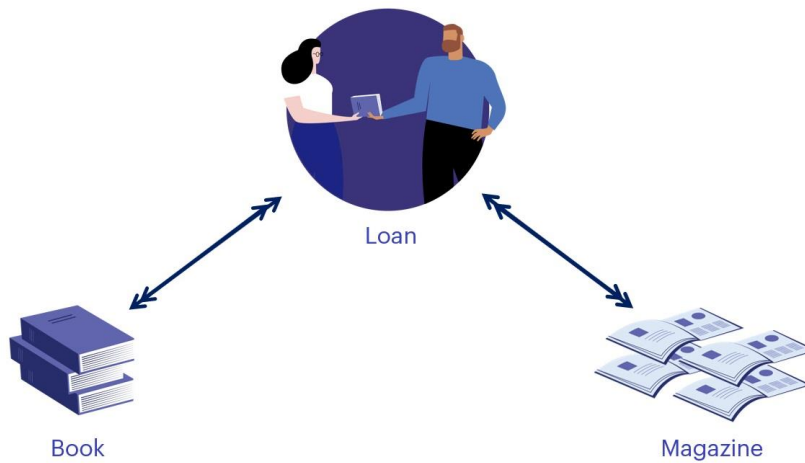
Let's look once again at the generated table diagram.

If we look again at the structure of the LOAN, LOANBOOK and MAGAZINELOANMAGAZINE tables, we see that they are the same as the tables generated in the previous proposals.

These proposals, then, change the screens offered to the end user but not the structure of the database.

If, instead of prioritizing book loans, priority had been given to magazines, the structure of the tables generated would remain the same.

## Reality



Now, is there any other totally flat option, which does not include second levels? If the end user requests simple screens without grids, what can we offer?

Remember that there is an N-N relationship between the entities Loan and Book, and also between Loan and Magazine.

We will analyze it for Loan and Book, and it will be analogous for Loan and Magazine.

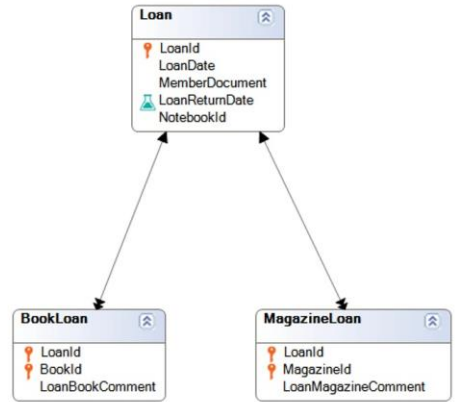
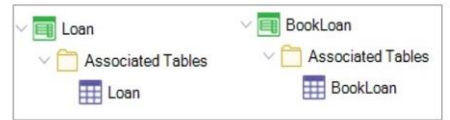


### Loan entity – Option 4: Three simple transactions (one level transactions)

| Name           | Type             |
|----------------|------------------|
| Loan           | Loan             |
| LoanId         | Id               |
| LoanDate       | Date             |
| MemberDocument | Numeric(8.0)     |
| MemberName     | Name             |
| MemberAddress  | Address, GeneXus |
| LoanReturnDate | Date             |
| NotebookId     | Id               |

| Name                | Type          |
|---------------------|---------------|
| Book                | Book          |
| BookId              | Id            |
| BookTitle           | Character(20) |
| BookEditionDate     | Date          |
| AuthorId            | Id            |
| AuthorName          | Name          |
| BookCopies          | Numeric(4.0)  |
| LiteraryGenreId     | Id            |
| LiteraryGenreName   | Name          |
| PublishingHouseId   | Id            |
| PublishingHouseName | Name          |
| BookOnLoanQty       | Numeric(4.0)  |
| BookAvailableQty    | Numeric(4.0)  |

| Name             | Type             |
|------------------|------------------|
| BookLoan         | BookLoan         |
| LoanId           | Id               |
| LoanDate         | Date             |
| MemberDocument   | Numeric(8.0)     |
| MemberName       | Name             |
| MemberAddress    | Address, GeneXus |
| LoanReturnDate   | Date             |
| NotebookId       | Id               |
| BookId           | Id               |
| BookTitle        | Character(20)    |
| BookAvailableQty | Numeric(4.0)     |
| LoanBookComment  | Character(40)    |



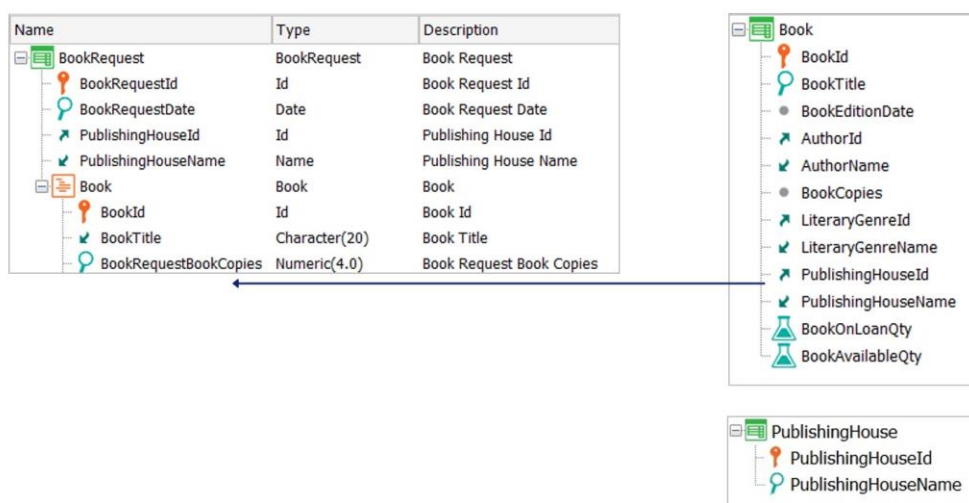
Let's consider the following structures.

For an N-N relationship to exist between them we need a relationship table with the compound key LoanId, BookId.

We then define the BookLoan transaction with the primary key made up of LoanId, BookId, and the following structure.

Which tables are generated? The same ones we have seen before.

## BookRequest transaction: Structure



At this point, we still have to solve the modeling for the request of new copies of books to a certain Publisher.

For this we are going to define the BookRequest transaction, which will represent a request to a Publisher, with the following attributes:

- BookRequestId, based on the ID domain
- BookRequestDate, with the current date by default
- PublishingHouseId, since the request is made to a Publishing House
- PublishingHouseName, which will be an attribute inferred from the foreign key PublishingHouseId.

Between this new entity and the Book there is an N-N relationship, since a request includes several books, and in turn a book can be included in several requests. We have already discussed that there are several design options. Let's add Book as a second level in the Request. Remember that it is an important requirement to check that the books indicated are published by the Publishing House in charge.

If we recall the structure of the Book transaction, we see that it has PublishingHouseId as FK, since a book is published by a Publishing House.

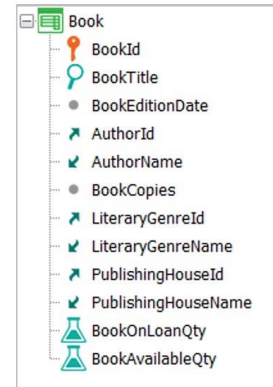
Can we also add that attribute in this new transaction so that it is inferred by BookId and we can check if it is indeed a book published by the Publishing House to which the request is made?

## BookRequest transaction: Structure



| Name                  | Type          | Description              |
|-----------------------|---------------|--------------------------|
| BookRequest           | BookRequest   | Book Request             |
| BookRequestId         | Id            | Book Request Id          |
| BookRequestDate       | Date          | Book Request Date        |
| PublishingHouseId     | Id            | Publishing House Id      |
| PublishingHouseName   | Name          | Publishing House Name    |
| Book                  | Book          | Book                     |
| BookId                | Id            | Book Id                  |
| BookTitle             | Character(20) | Book Title               |
| BookRequestBookCopies | Numeric(4,0)  | Book Request Book Copies |
| PublishingHouseId     |               |                          |

❌ Duplicate Attribute Name: 'PublishingHouseId'



Multiple reference conflict:  
Where to define the subtype group?



What happens when we try to add it? GeneXus returns an error, because the PublishingHouseId attribute is already declared in the transaction structure and we cannot add it again.

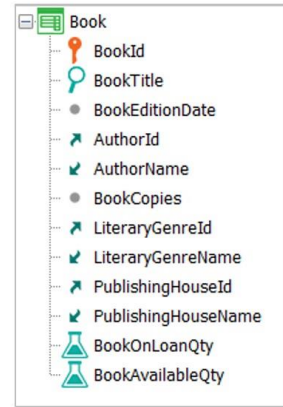
We have a multiple reference problem. We need to make reference to the Publishing House concept more than once. So, how can we solve a conflict of this type? By using subtypes. This means to define new attributes, with other names, that refer to the same concept of the Publishing House.

But... Where do we define the subtype? In the Publishing House concept referred to in the header, or referred to in the 2nd level? Is it the same?

# BookRequest transaction: Structure



| Name                         | Type          | Description              |
|------------------------------|---------------|--------------------------|
| BookRequest                  | BookRequest   | Book Request             |
| BookRequestId                | Id            | Book Request Id          |
| BookRequestDate              | Date          | Book Request Date        |
| PublishingHouseId            | Id            | Publishing House Id      |
| PublishingHouseName          | Name          | Publishing House Name    |
| Book                         | Book          | Book                     |
| BookId                       | Id            | Book Id                  |
| BookTitle                    | Character(20) | Book Title               |
| BookRequestBookCopies        | Numeric(4,0)  | Book Request Book Copies |
| BookRequestPublishingHouseId |               |                          |



Multiple reference conflict:  
Where to define the subtype group?

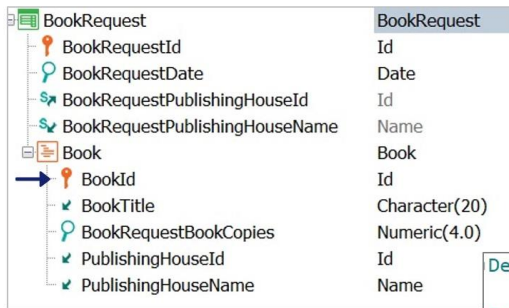


Let's look at the transaction design: the PublishingHouseId attribute is present in the Book transaction structure as FK, therefore, GeneXus infers that value from BookId.

If then both BookId and PublishingHouseId are present in another transaction, as would happen in this case in the 2nd level of BookRequest, then GeneXus will apply that relationship and PublishingHouseId will be an inferred FK.

So, would it be correct to rename the attributes here to define the required subtype group? No, because we would be removing this relationship, and we would allow the user to add a different PublishingHouseId value than the one previously defined in Book.

## BookRequest transaction: Subtype group in the first level



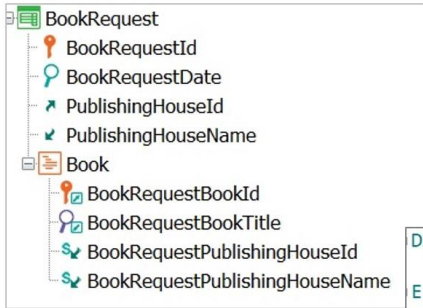
```
Default(BookRequestDate, Today());
Error("The book doesn't belong to the Publishing House")
if PublishingHouseId <> BookRequestPublishingHouseId;
```

| Group Structure                |                                    |                     |
|--------------------------------|------------------------------------|---------------------|
| Subtype                        | Description                        | Supertype           |
| BookRequestPublishingHouse     |                                    |                     |
| BookRequestPublishingHouseId   | Book Request Publishing House Id   | PublishingHouseId   |
| BookRequestPublishingHouseName | Book Request Publishing House Name | PublishingHouseName |

Let's look at the BookRequest header. Here the PublishingHouseId attribute is a common FK, which is not inferred from anywhere; therefore, we can perfectly remove it, define new attributes and declare them as subtypes of PublishingHouseId and PublishingHouseName. Then we will be able to compare them in an Error rule to validate that the requested books are from the indicated Publishing House.

To complete the requirement we declare the following rules. Since this Error rule involves attributes from both levels, it is a rule associated by GeneXus with the 2nd level. Therefore, it will be triggered for each line. This means that when entering a new BookId, the condition will be evaluated, and if true, the error will be triggered.

## BookRequest transaction: Subtype group throughout the second level



```
Default(BookRequestDate, Today());
Error("The book doesn't belong to the Publishing House")
    if PublishingHouseId <> BookRequestPublishingHouseId;
```

| Group Structure                |                                    |                     |
|--------------------------------|------------------------------------|---------------------|
| Subtype                        | Description                        | Supertype           |
| BookPublishingHouse            |                                    |                     |
| BookRequestBookId              | Book Request Book Id               | BookId              |
| BookRequestBookTitle           | Book Request Book Title            | BookTitle           |
| BookRequestPublishingHouseId   | Book Request Publishing House Id   | PublishingHouseId   |
| BookRequestPublishingHouseName | Book Request Publishing House Name | PublishingHouseName |

We have met the requirement. What if we had defined the entire second level as a group of subtypes?

We would also solve it, with the difference that (as we analyzed in detail in the subtypes video) although this solution may be less obvious and lead to having to create more subtypes to be able to infer them where needed, its main advantage is that the ambiguity is resolved in the same table that causes

## BookRequest transaction: Subtype group in the Book transaction



| Name                    | Type          |
|-------------------------|---------------|
| Book                    | Book          |
| BookId                  | Id            |
| BookTitle               | Character(20) |
| BookEditionDate         | Date          |
| AuthorId                | Id            |
| AuthorName              | Name          |
| BookCopies              | Numeric(4.0)  |
| LiteraryGenreId         | Id            |
| LiteraryGenreName       | Name          |
| BookPublishingHouseId   | Id            |
| BookPublishingHouseName | Name          |
| BookOnLoanQty           | Numeric(4.0)  |
| BookAvailableQty        | Numeric(4.0)  |

| Name                    | Type          |
|-------------------------|---------------|
| BookRequest             | BookRequest   |
| BookRequestId           | Id            |
| BookRequestDate         | Date          |
| PublishingHouseId       | Id            |
| PublishingHouseName     | Name          |
| Book                    | Book          |
| BookId                  | Id            |
| BookTitle               | Character(20) |
| BookRequestBookCopies   | Numeric(4.0)  |
| BookPublishingHouseId   | Id            |
| BookPublishingHouseName | Name          |

```
Default(BookRequestDate, Today());
```

```
Error("The book doesn't belong to the Publishing House")  
if PublishingHouseId <> BookPublishingHouseId;
```

| Group Structure         |                            |                     |
|-------------------------|----------------------------|---------------------|
| Subtype                 | Description                | Supertype           |
| BookPublishingHouse     |                            |                     |
| BookPublishingHouseId   | Book Publishing House Id   | PublishingHouseId   |
| BookPublishingHouseName | Book Publishing House Name | PublishingHouseName |

Could we propose some other option to avoid the definition of subtypes in the two-level transaction?

Yes, but we must make it clear that as a working criterion we always suggest addressing the reference conflict in the transaction that presents it. That is why we have solved it in the BookRequest transaction itself.

OK. We could look at the Book transaction, and define the subtype group there.

What is gained with this? That there is no reference conflict in the BookRequest transaction.

But we must keep in mind that if there are queries already defined on Book, for example, a book catalog, they will have to be updated because the names of some attributes have been changed.

Finally, could the use of subtypes be avoided? Yes, for example, by calling on each line, and before recording, a process that returns whether the book is published by the indicated Publishing House or not.

The definition of the subtype group is avoided, but a process is invoked on each line to evaluate a condition that can be resolved in the definition of the transaction itself.

## Summary

Analyze, evaluate the reality to be modeled and implement the option we consider correct, always working together with the end user.



---

In conclusion, we should remember that we must always analyze, evaluate the reality to be modeled and implement the option we consider correct, and for this it is essential to always work together with the end user, who will guide us in selecting the design.

Let's not forget that the application must be a support tool for end users to better manage and develop their business.



*GeneXus*<sup>TM</sup>

[training.genexus.com](http://training.genexus.com)  
[wiki.genexus.com](http://wiki.genexus.com)