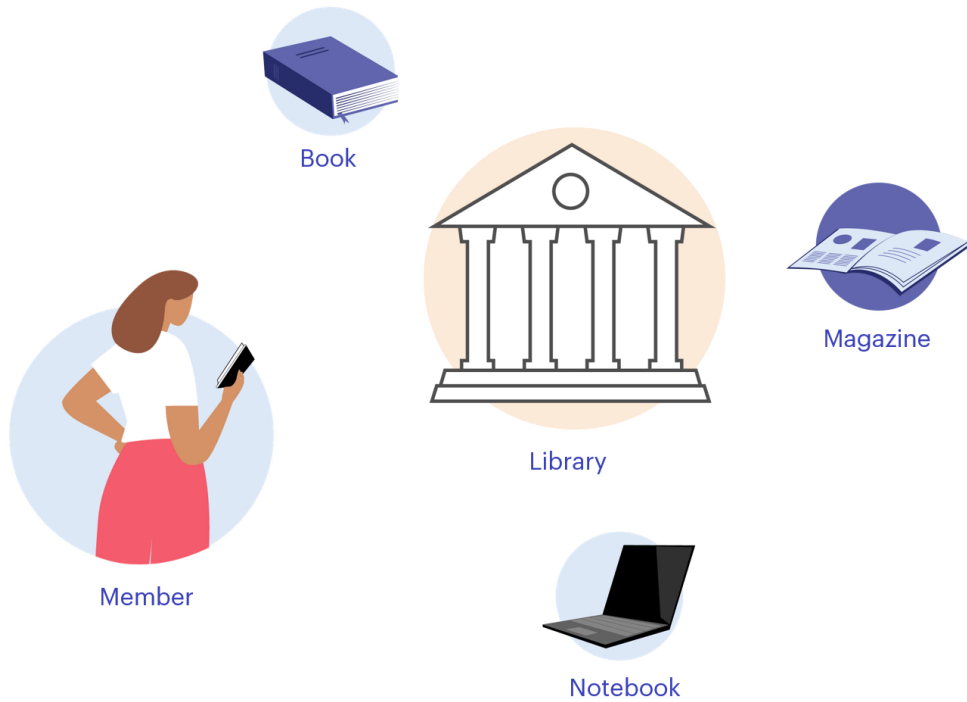


Análisis del modelo de diseño de transacciones

Biblioteca

GeneXus



A lo largo del curso anterior, hemos estudiado todo lo necesario para poder modelar correctamente una determinada realidad en GeneXus. En este video, a través del estudio de una realidad acotada, analizaremos diferentes opciones para resolver el diseño de transacciones, utilizando un conjunto de recursos esenciales y dando solución a una serie de requerimientos reales.

Supongamos que se necesita diseñar una aplicación GeneXus para la gestión de tareas de una Biblioteca en relación al préstamo de libros, revistas y notebooks.

La misma trabaja con Socios que pueden acceder a diferente material de lectura, ya sea libros o revistas, y a equipos informáticos que la Biblioteca ofrece en calidad de préstamo.



Country

- Id
- Name



Member

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



Author

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

La información que se necesita registrar es la siguiente:

País (Country)

Todo País se registra con un identificador único y su nombre.

Socio (Member)

Todo Socio de la Biblioteca se registra con su Documento de identidad, nombre, dirección, foto y un teléfono de contacto.

Los socios de la Biblioteca deben ser mayores de 20 años.

Autor (Author)

Todo Autor se registra con un identificador único, su nombre, foto y país de origen. Un Autor puede tener varios libros publicados.



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

Libro (Book)

Todo Libro se registra con un identificador único, título, fecha de edición y cantidad de ejemplares que dispone la Biblioteca. Puede tratarse de una Novela, un Ensayo, un libro de Poesía, etc, por lo tanto, un libro corresponde a un género literario.

Además, un Libro tiene un Autor y una Editorial a cargo de su publicación.

Revista (Magazine)

Toda Revista se registra con un identificador único, título, fecha de publicación, la imagen de la portada y la cantidad de ejemplares disponible.

Notebook (Notebook)

La Biblioteca ofrece notebooks a sus Socios, en calidad de préstamo, ya que muchos son escritores e investigadores. Cada notebook se registra con un identificador único, su imagen y una breve descripción.

Además, se registra su estado (disponible o en préstamo).



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

Préstamo (Loan)

La Biblioteca ofrece libros, revistas y notebooks en calidad de préstamo a sus Socios.

Todo préstamo se registra con un identificador único, la fecha de retiro, el Socio, y la fecha de devolución que se debe determinar automáticamente. Todo préstamo se realiza por un plazo de 15 días, puede incluir un máximo de 3 libros, 4 revistas, y puede o no incluir el préstamo de una notebook.

Solamente se puede retirar un ejemplar de cada publicación, y es posible registrar algún comentario que se considere necesario a nivel del préstamo de cada ejemplar (por ejemplo, que la tapa se encuentre dañada, que le falte alguna hoja, etc).

Además, la fecha en que se registra un nuevo préstamo siempre es la actual y no debe ser posible modificarla.

Solicitud de ejemplares (BookRequest)

Muchas veces ocurre que ciertos libros tienen mucha demanda, y la Biblioteca decide solicitar más cantidad de ejemplares.

Para eso, realiza una Solicitud a la Editorial correspondiente. El sistema debe controlar que se soliciten ejemplares de libros a cargo de la Editorial indicada.



Country

- Id
- Name



Member

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



Author

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

The screenshot shows the 'Properties' window for the 'Id' domain. The 'Domain: Id' section is expanded, showing a table of properties:

| Name | Id |
|-------------------|---------------|
| Description | Id |
| Nulls in Forms | Empty as Null |
| Class | Attribute |
| Module | Root Module |
| Qualified Name | Id |
| Object Visibility | Public |

The 'Type Definition' section is also expanded, showing a table of properties:

| Property | Value |
|-------------------|-------------|
| Based on | (none) |
| Data Type | Numeric |
| Length | 4 |
| Decimals | 0 |
| Signed | False |
| Enum Values | |
| Collection | False |
| Autonumber | True |
| Autonumber start | 1 |
| Autonumber step | 1 |

The screenshot shows the 'Properties' window for the 'Name' domain. The 'Domain: Name' section is expanded, showing a table of properties:

| Name | Name |
|-------------------|---------------|
| Description | Name |
| Nulls in Forms | Empty as Null |
| Class | Attribute |
| Module | Root Module |
| Qualified Name | Name |
| Object Visibility | Public |

The 'Type Definition' section is also expanded, showing a table of properties:

| Property | Value |
|--------------------------|------------------|
| Based on | (none) |
| Data Type | Character |
| Length | 20 |
| Enum Values | |
| Collection | False |
| Dimensions | Scalar |
| Initial value | |
| Enable national language | Yes |

Comencemos a analizar esta realidad.

En principio, claramente se distinguen ciertas entidades simples que podemos comenzar a definir, como lo son, por ejemplo, País (Country), Socio (Member) y Autor (Author).

Pero antes, es bueno tomar en cuenta que la mayoría de las entidades se registran con un identificador que perfectamente puede ser autoenumerado, a excepción del Nro de Socio, que como mencionamos, se registra con su documento de identidad.

Por lo tanto, definimos el Dominio Id, con la propiedad Autonumber en True.

Definimos también el dominio Name, como carácter de 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 |

Vamos a empezar definiendo la transacción Country, con **CountryId** como llave primaria, y **CountryName** como atributo secundario. CountryId queda basado en el dominio Id, y para controlar que el nombre no se repita, definimos el correspondiente índice unique. Esta misma definición vale en todas las entidades donde es necesario controlar que el nombre no se repita.

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 |

Si fuera necesario registrar las ciudades de cada país, porque es de interés conocer la ciudad de nacimiento del autor, podríamos modelarla como una entidad débil respecto del país, ya que una ciudad no existe fuera de ese contexto.

Así que se agregaría un segundo nivel a la transacción Country, pero al considerar City como una entidad débil no existirá como una transacción en sí misma.

Esto significa que CityId no existirá como llave primaria en ninguna tabla, y por lo tanto, para poder conocer, por ejemplo, la ciudad de nacimiento de un autor, se necesitará el par compuesto por los atributos CountryId, CityId, que será la llave primaria de la tabla COUNTRYCITY asociada al segundo nivel de la transacción Country.

De todas formas, como la descripción de la realidad que estamos estudiando no incluye el concepto de la ciudad, vamos a modelar el País (Country) como una transacción simple.

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;
  
```

Pasemos ahora al Socio. Para eso definimos la transacción Member, con los siguientes atributos:

- **MemberDocument**, que corresponde al documento de identidad y por lo tanto no es autonumerado, así que lo definimos como numérico de 8 dígitos
- **MemberName**, de tipo Name
- **MemberImage**, de tipo Image
- **MemberAddress**, basado en el dominio semántico Address
- y **MemberPhone**, de tipo Phone

Pero, además, la realidad nos dice que los socios deben ser mayores de 20 años, por lo tanto necesitamos su fecha de nacimiento y la edad, que deberá ser calculada en forma automática.

Así que agregamos a la estructura de la transacción los atributos:

- **MemberBirthDate**, de tipo Date
- y **MemberAge**, numérico de 3 dígitos

¿Cómo calculamos la edad del socio? Utilizando la función Age, que calcula la edad a partir de la fecha de nacimiento y la fecha actual.

Así que definimos MemberAge como un atributo calculado que obtiene su valor a partir de la siguiente expresión:

Age(MemberBirthDate, Today())

¿Por qué utilizamos la función Today() y no la variable &today?

Porque no es posible utilizar variables en la declaración de fórmulas.

Para controlar que todo socio sea mayor de 20 años alcanza con declarar la siguiente regla
Error:

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

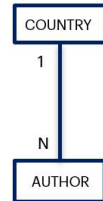
Es importante destacar que no estamos considerando en este análisis la declaración de reglas para el control básico de ingreso de datos, como por ejemplo, controlar que no se ingrese un socio sin nombre, etc.

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 |



Consideremos ahora a los Autores. Debemos definir la transacción Author, con los atributos:

- AuthorId
- AuthorName
- AuthorImage

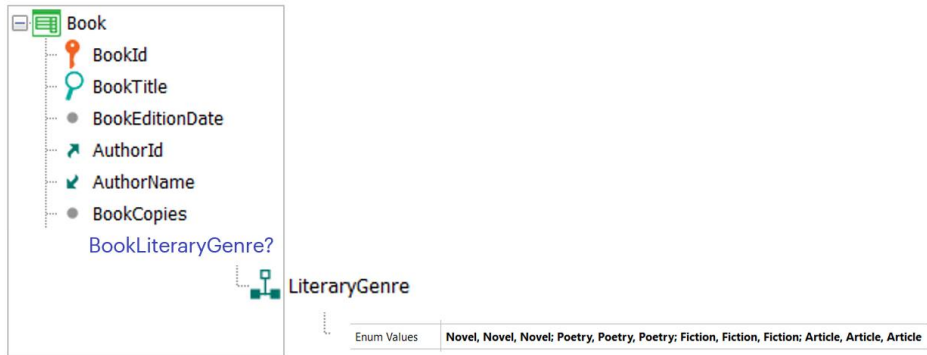
Y sabemos que se debe registrar el país del mismo, ya que todo Autor tiene un país de nacimiento, por lo tanto, agregamos CountryId y CountryName, donde CountryId es una clave foránea y CountryName un atributo que se infiere a partir de dicha clave foránea.

De esta forma representamos que entre Country y Author existe una relación 1-N.

Transaction definition: Book



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



Pensemos ahora en el concepto del Libro. Se trata de un concepto fuerte que se identifica también con un valor autonumerado, tiene un título, una fecha de edición, un autor y también la cantidad de ejemplares adquiridos por la Biblioteca.

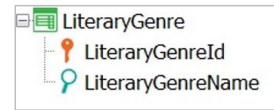
Pero, además, la realidad nos dice que un libro corresponde a un género literario, ya que puede tratarse de una Novela, un Ensayo, etc.

Entonces, ¿cómo modelamos el concepto del Género literario? Si pensamos que estos géneros son finitos, una primera opción que podemos considerar es la creación de un dominio enumerado.

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 |

Pero la categorización de los géneros literarios ha ido cambiando a lo largo del tiempo y es posible que siga haciéndolo, entonces no parece que el dominio enumerado sea la mejor opción, ya que los cambios deberían realizarse en forma manual y no sería escalable.

Podemos considerar una transacción GeneroLiterario (LiteraryGenre), con los atributos:

- **LiteraryGenreId**
- y **LiteraryGenreName**

Es una buena decisión definir desde ya un índice único sobre el nombre del género para evitar el registro de géneros literarios con el mismo nombre.

Pero el Libro requiere registrar también el Autor y la Editorial responsable de su publicación. El Autor ya lo tenemos definido como entidad, pero la Editorial no. Y si bien no está declarada explícitamente, en el análisis surge la necesidad de modelar la entidad Editorial.

Así que definimos la transacción PublishingHouse con los atributos:

- **PublishingHouseId**
- y **PublishingHouseName**

También podemos desde ya crear el correspondiente índice único sobre el nombre de la editorial.

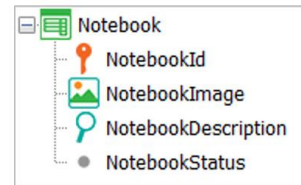
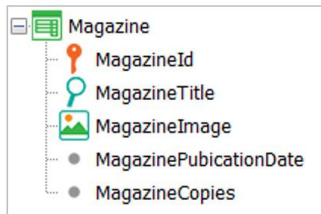
Transactions definition: Magazine and Notebook



- Id
- Title
- Publication Date
- Image
- Copies



- Id
- Image
- Description
- Status



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

Enum Values **OnLoan, On loan, OnLoan; Available, Available, Available**

La Biblioteca también ofrece Revistas y Notebooks a sus socios, así que definimos la transacción Magazine, con los atributos:

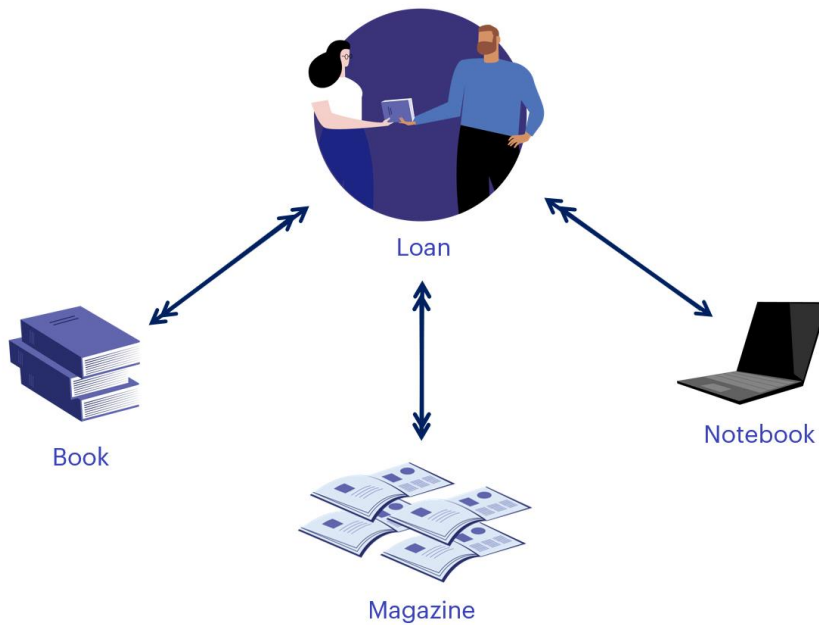
- **MagazineId**
- **MagazineTitle**
- **MagazineImage**
- **MagazinePublicationDate**
- y **MagazineCopies** para registrar la cantidad de ejemplares adquiridos.

Y también la transacción Notebook, con los atributos:

- **NotebookId**
- **NotebookImage**
- **NotebookDescription** de tipo LongVarChar
- y **NotebookStatus**

Una notebook está "Disponible" o "En préstamo", así que definimos el dominio enumerado Status con esos valores.

Transaction definition: Loan



Analizamos ahora el concepto del Préstamo, que es una entidad fundamental en esta realidad.

La Biblioteca permite que un socio retire, en calidad de préstamo y por 15 días, una notebook, 3 libros y 4 revistas.

Tenemos entonces una relación 1-N entre las entidades Préstamo y Notebook, y una relación N-N entre Préstamo y Libro, y entre Préstamo y Revista.

¿Cómo podemos modelarla?

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) | | |

Analizamos una **primera opción**:

Creamos la transacción Loan, y vamos definiendo los siguientes atributos:

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

Se solicita que la fecha de devolución se calcule automáticamente. Un préstamo se realiza por 15 días, así que declaramos el siguiente cálculo asociado al atributo LoanReturnDate:

LoanDate.AddDays(15)

Además, un préstamo puede incluir o no una notebook, así que agregamos:

- **NotebookId**
- y **NotebookDescription**

Pero en esta transacción Loan, NotebookId es una clave foránea no obligatoria, por lo tanto, indicamos su propiedad Nullable con el valor 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;
```

Un préstamo incluye varios libros, con un máximo de 3. Así que definimos un segundo nivel para registrarlos, incluyendo un atributo LoanBookComment, para registrar algún comentario que se necesite al momento de realizar el préstamo.

¿Cómo controlamos que no se ingresen más de 3 libros?

Podemos definir un nuevo atributo LoanBooksQty, que cuente dicha cantidad y condicionar luego ese valor en una regla Error:
Error("Only 3 books can be checked out") if LoanBooksQty > 3;

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;

```

Pero el mismo préstamo puede incluir hasta 4 revistas, así que podemos definir otro segundo nivel, paralelo al libro, para registrar las revistas. De igual manera se puede controlar que no se retiren más de 4.

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);

```

La realidad describe claramente que la fecha de registro de un préstamo siempre debe ser la actual, sin posibilidad de ser modificada.

Para eso declaramos las siguientes reglas:

```

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 |

Si bien el objetivo de este video es concentrarnos en analizar el diseño de transacciones, vamos a sugerir una opción de implementación para que, al momento de registrar un préstamo, se ofrezcan solamente las notebooks que se encuentren en estado disponible (Available). Esta implementación podemos realizarla desde la definición de la propia transacción y por eso la incluimos.

Vamos a eliminar el atributo NotebookDescription, porque “disfrazaremos” el identificador de una notebook con su descripción. Seleccionamos NotebookId y lo definimos como un **Dynamic Combo Box**, con NotebookId en la propiedad *Item Values*, y NotebookDescription en la propiedad *Item Descriptions*.

Esto cargará todas las notebooks registradas. Para que solamente se ofrezcan las que tienen estado disponible, declaramos la siguiente condición:

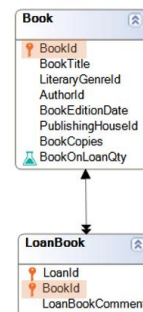
NotebookStatus = Status.Available;

También debemos configurar la propiedad *Empty Item* con el valor True, ya que NotebookId puede no elegirse, pues puede ser nulo.

Si bien esta no es la única manera de controlar que una notebook se encuentre disponible al momento de registrar un préstamo, es una implementación que podemos resolver desde la definición de la propia transacción.

Available quantities of Books and Magazines

| Name | Type | Formula |
|---------------------|---------------|------------------------|
| 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) | count(LoanBookComment) |



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

Otro requerimiento necesario, es conocer la disponibilidad de los libros o revistas de interés.

Sabemos la cantidad de ejemplares adquiridos por la Biblioteca para cada libro y revista, pero no sabemos la cantidad que va quedando disponible.

¿Podemos obtener esos valores de una forma simple y basándonos en el diseño de transacciones?

Si en la transacción Book, definimos un nuevo atributo, BookOnLoanQty, podemos calcular la cantidad de ejemplares prestados, y por lo tanto saber la cantidad que quedan disponibles.

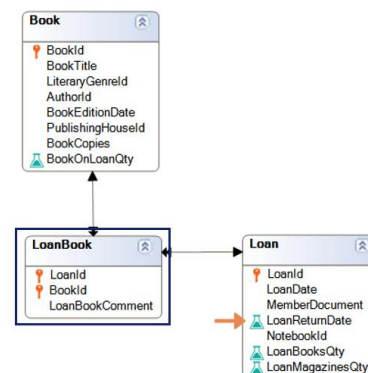
¿Qué sucede si a ese nuevo atributo BookOnLoanQty le asociamos el cálculo Count(LoanBookComment)? ¿Sumará efectivamente la cantidad de ejemplares de ese libro que se encuentran en préstamo?

La respuesta es SÍ, porque la tabla asociada a la transacción donde definimos el cálculo es BOOK. Y la tabla donde se resuelve el cálculo es LOANBOOK, ya que queda determinada por el atributo LoanBookComment.

Entre ambas tablas existe un atributo en común que es BookId, y por lo tanto es un filtro implícito que GeneXus aplicará, o sea que ese cálculo devolverá el total de ejemplares de dicho libro que ha sido prestado.

Available quantities of Books and Magazines

| Name | Type | Formula |
|---------------------|---------------|---|
| 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) | <code>count(LoanBookComment, LoanReturnDate >= servernow())</code> |



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

Pero necesitamos saber la cantidad de ejemplares actualmente en préstamo, así que necesitamos agregar una condición al cálculo que permita contar la cantidad de ejemplares en préstamos actualmente vigentes. Para eso consideramos los préstamos cuya fecha de devolución sea mayor o igual a la actual.

Podemos declarar lo siguiente:

Count(LoanBookComment, LoanReturnDate >= servernow())

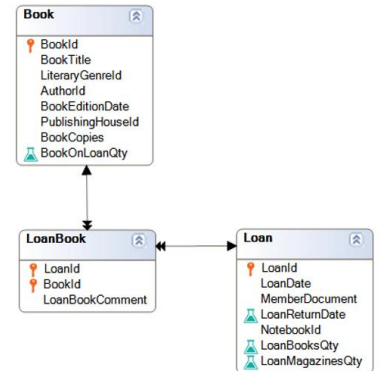
La función `servernow()` permite obtener la fecha y hora actual del server. Se podría utilizar también la función `Today()`.

¿Podemos declarar esa condición de cálculo? Sí, porque el atributo involucrado (`LoanReturnDate`) pertenece a la tabla extendida de la tabla donde se resuelve el cálculo declarado, que es `LOANBOOK`.

Available quantities of Books and Magazines

| Name | Type | Formula |
|---------------------|---------------|---|
| 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) | count(LoanBookComment, LoanReturnDate >= servernow()) |
| BookAvailableQty | Numeric(4,0) | BookCopies - BookOnLoanQty |

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



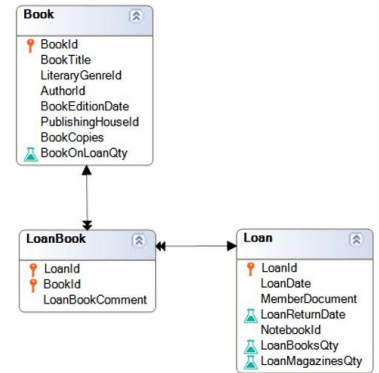
Por lo tanto, si sabemos la cantidad total de ejemplares, y la cantidad de ejemplares en préstamos vigentes, entonces podemos saber la cantidad de ejemplares actualmente disponibles.

Así que la estructura de la transacción Book nos queda así:

Available quantities of Books and Magazines

| Name | Type | Formula |
|---------------------|---------------|---|
| 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) | count(LoanBookComment, LoanReturnDate >= servernow()) |
| BookAvailableQty | Numeric(4,0) | BookCopies - BookOnLoanQty |

| Name | Type | Formula |
|-------------------------|-----------------|---|
| Magazine | Magazine | |
| 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 |



De igual forma podemos saber la cantidad de ejemplares disponibles de una determinada revista:

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;

```

Ahora tenemos disponible toda la información que necesitamos para validar un préstamo, así que agreguemos los atributos necesarios para validar que exista disponibilidad de los ejemplares que se desean retirar en el préstamo:

- **BookAvailableQty**, en el nivel correspondiente a los libros, y
- **MagazineAvailableQty**, en el nivel correspondiente a las revistas

Y controlamos con las siguientes reglas Error:

Loan Transaction: Form and Table structure

Hemos resuelto hasta aquí los requerimientos para registrar Préstamos. Pero observemos el form generado para el diseño que hemos hecho.

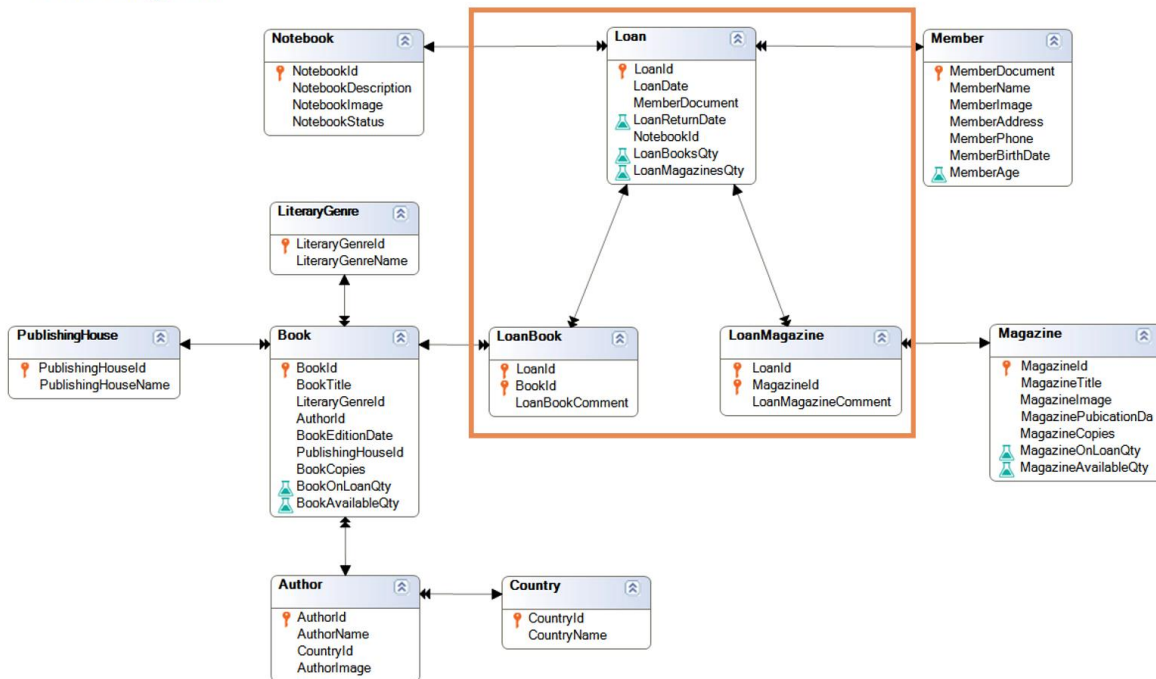
¿Qué tablas asociadas crea GeneXus para esta transacción Loan?

Crea 3 tablas:

- **LOAN**, asociada al primer nivel, con LoanId como PK.
- **LOANBOOK**, asociada al nivel de los libros, con LoanId y BookId como PK compuesta
- **LOANMAGAZINE**, asociada al nivel de las revistas, con LoanId y MagazineId como PK compuesta.

Al definir dos niveles paralelos, vemos en el form grillas paralelas. Esto tal vez puede ser engorroso para el usuario final, que puede solicitar simplificar dicho diseño, y manejar la información en diferentes pantallas más simples.

Table Diagram

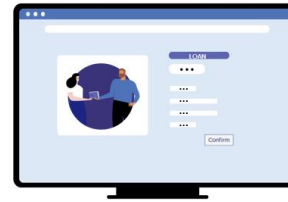


Para el diseño actual, GeneXus ha creado las siguientes tablas. Observemos en particular, las tablas LOAN, LOANBOOK y LOANMAGAZINE, que corresponden a las tablas asociadas a los tres niveles de la transacción Loan definida.

¿Podemos proponer otro diseño para la transacción Préstamo? Sí.

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



Veamos entonces una **segunda opción para modelar el Préstamo (Loan)**.

Vamos a suponer que el usuario final nos solicita “aplanar” o “alivianar” la carga de esta pantalla, y visualizar la información en **tres pantallas**:

- **una pantalla para el registro general del préstamo**, en la cual se registra el préstamo, con su fecha, los datos del socio y si se incluye notebook o no.
- **una pantalla para el registro de los libros que se prestan**, en la cual se registran los libros con los controles necesarios ya definidos.
- **y una pantalla para el registro de las revistas que se prestan**, en la cual se registran las revistas con los controles necesarios ya definidos.

¿Cómo podemos lograrlo?

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

Loan

<ErrorViewer: ErrorViewer>

<Toolbar>

Id

Date

Member Document

Member Name

Member Address

Return Date

Notebook Description

<FormButtons>

La primera pantalla corresponde a los datos generales del préstamo, así que a la transacción Loan le borramos los niveles, y dejamos la información general, con las siguientes reglas.

Se genera entonces el siguiente form:

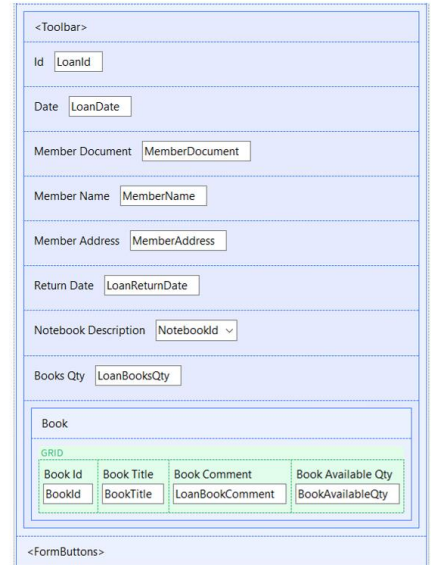
Loan entity – Option 2: Three transactions

| Name | Type | Formula | Nullable |
|------------------|------------------|----------------------------|----------|
| BookLoan | BookLoan | | |
| 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) | |
| Book | Book | | |
| 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;
```



Para ofrecer otra pantalla con el registro de libros, necesitamos una nueva transacción. La llamaremos BookLoan.

Se desea ingresar el identificador del préstamo, ver su información general, e ingresar los libros. Por lo tanto, la llave primaria será también LoanId, y de esa forma no se creará una nueva tabla porque ya existe una donde LoanId es clave primaria. De esa forma estaremos referenciando a la misma entidad Loan y solamente se creará la tabla asociada al segundo nivel.

La transacción BookLoan tiene entonces la siguiente estructura, con las siguientes reglas.

Se genera entonces el siguiente form:

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 | | |
| MemberAddress | Address, GeneXus | | |
| LoanReturnDate | Date | LoanDate.adddays(15) | |
| NotebookId | Id | | Yes |
| LoanMagazinesQty | Numeric(4.0) | count(LoanMagazineComment) | |
| Magazine | Magazine | | |
| MagazineId | Id | | No |
| MagazineTitle | Character(20) | | |
| LoanMagazineComment | Character(40) | | No |
| MagazineAvailableQty | Numeric(4.0) | MagazineCopies - MagazineOnLoanQty | |

```



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;

```

Associated Tables:

-  Loan
-  MagazineLoanMagazine

<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>

Algo similar haremos para registrar el préstamo de las revistas. Creamos la transacción MagazineLoan con la siguiente estructura, y las siguientes reglas:

Loan entity – Option 2: Three transactions

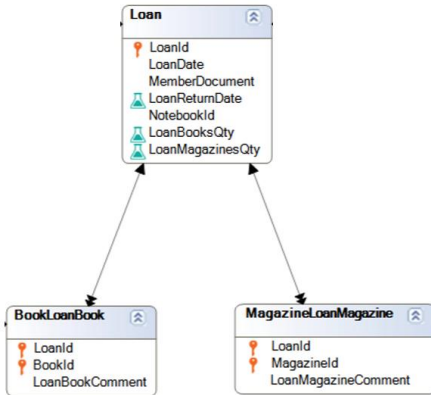


Antes de analizar el diagrama de tablas generado, observemos las tablas asociadas a las transacciones creadas en esta propuesta. GeneXus crea también 3 tablas:

- LOAN
- BOOKLOANBOOK
- y MAGAZINELOANMAGAZINE

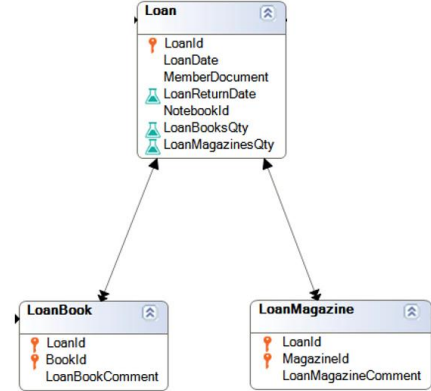
Table Diagram

Current model:



=

Previous model:

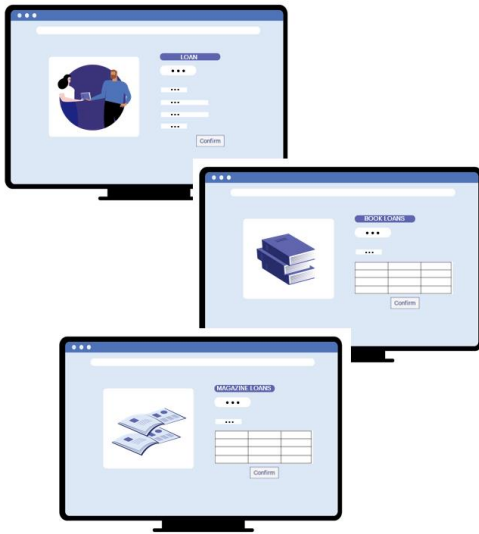


Observemos el diagrama de tablas.

Si nos detenemos en las tablas LOAN, BOOKLOANBOOK y MAGAZINELOANMAGAZINE, podemos ver que tienen exactamente la misma estructura que las generadas en el diseño propuesto anteriormente.

Table Diagram

Current model:



Previous model:



Esto significa que la diferencia entre ambos diseños no está en las estructuras generadas si no en las pantallas ofrecidas al usuario final para el manejo de la información. Será siempre el usuario final quien nos diga qué diseño se adapta mejor a su negocio.

Desde el punto de vista de la estructura de la base de datos, es exactamente igual.

Loan entity – Option 3: Two transactions





Pero también podemos proponer una **tercera opción** para el modelado del Préstamo.

Tal vez podríamos considerar que la Biblioteca basa los préstamos en los libros, y el usuario final solicita una sola pantalla donde se ingresan los datos generales del préstamo y también de los libros que se retiran. Luego, desde otra pantalla, se registran las revistas que se incluyen en el préstamo.

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;
```

<Toolbar>

Id

Date

Member Document

Member Name

Member Address

Return Date

Notebook Description

Books Qty

Book

GRID

| Book Id | Book Title | Book Comment | Book Available Qty |
|---------|------------|-----------------|--------------------|
| BookId | BookTitle | LoanBookComment | BookAvailableQty |



<FormButtons>

Para eso, consideramos la transacción Loan, con un segundo nivel para registrar los libros. Su estructura sería la siguiente, con las siguientes reglas:

Loan entity – Option 3: Two transactions

| Name | Type | Formula | Nullable |
|----------------------|------------------|------------------------------------|----------|
| MagazineLoan | MagazineLoan | | |
| LoanId | Id | | No |
| LoanDate | Date | | No |
| MemberDocument | Numeric(8.0) | | No |
| MemberName | Name | | |
| MemberAddress | Address, GeneXus | | |
| LoanReturnDate | Date | LoanDate.adddays(15) | |
| NotebookId | Id | | Yes |
| LoanMagazinesQty | Numeric(4.0) | count(LoanMagazineComment) | |
| Magazine | Magazine | | |
| MagazineId | Id | | No |
| MagazineTitle | Character(20) | | |
| LoanMagazineComment | Character(40) | | No |
| MagazineAvailableQty | Numeric(4.0) | MagazineCopies - MagazineOnLoanQty | |

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;
```

<Toolbar>

Id

Date

Member Document

Member Name

Member Address

Return Date

Notebook Description

Magazines Qty

Magazine

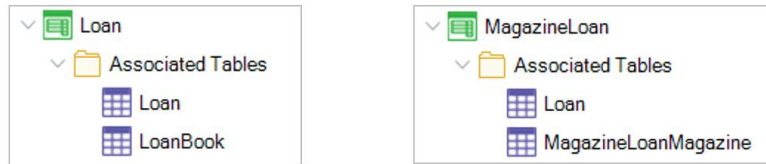
GRID

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

<FormButtons>

Para el registro de las revistas que se incluyen en el préstamo, tenemos la transacción MagazineLoan con la siguiente estructura, y las siguientes reglas:

Loan entity – Option 3: Two transactions

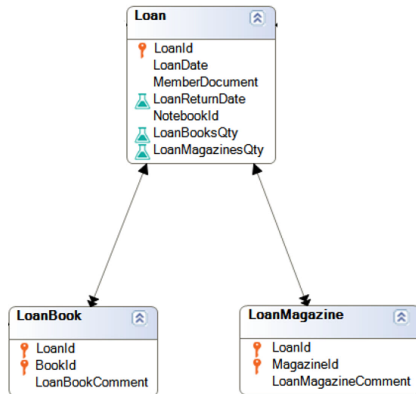


¿Qué tablas crea GeneXus asociadas a estas transacciones? 3 tablas: LOAN, LOANBOOK y MAGAZINELOANMAGAZINE.

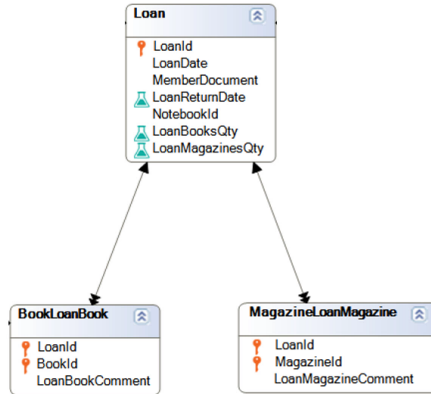
- LOAN y LOANBOOK asociadas a la transacción Loan
- LOAN y MAGAZINELOANMAGAZINE asociadas a la transacción MagazineLoan

Table Diagram

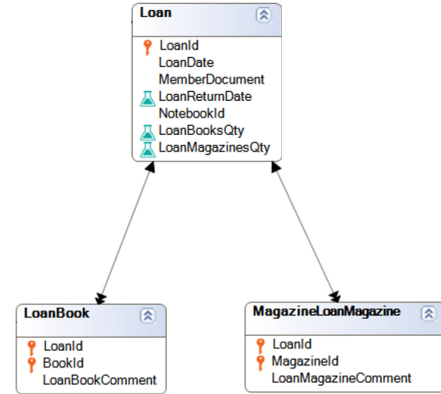
First model:



Second model:



Current model:



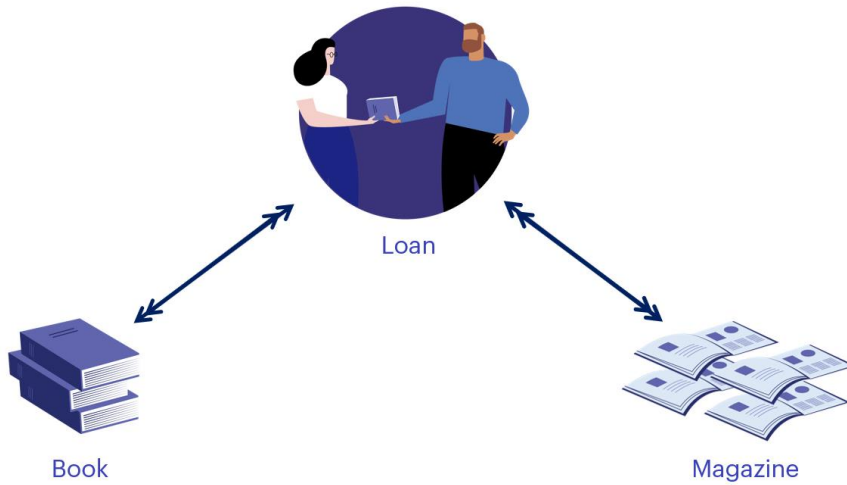
Observemos una vez más el diagrama de tablas generado.

Si nuevamente observamos la estructura de las tablas LOAN, LOANBOOK y MAGAZINELOANMAGAZINE, vemos que son iguales a las tablas generadas en las propuestas anteriores.

Estas propuestas, entonces, cambian las pantallas ofrecidas al usuario final pero no la estructura de la base de datos.

Si en lugar de priorizar el préstamo de los libros, se hubiera hecho con las revistas, la estructura de las tablas generadas seguirá siendo la misma.

Reality



Ahora bien, ¿existe alguna otra opción totalmente plana, que no incluya segundos niveles? Si el usuario final solicita pantallas simples sin grillas, ¿qué podemos ofrecerle?

Recordemos que entre las entidades Préstamo y Libro existe una relación N-N, y entre Préstamo y Revista también.

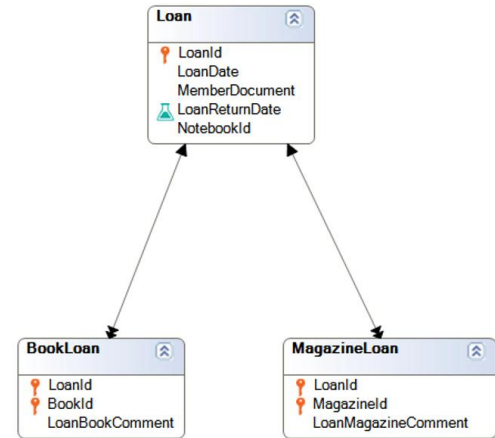
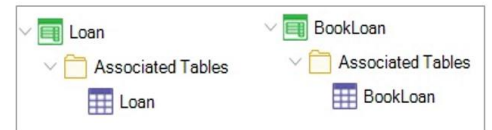
Vamos a analizarlo para Préstamo y Libro, y será análogo para Préstamo y Revista.

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) |



Consideremos las siguientes estructuras.

Para que entre ellas exista una relación N-N necesitamos una tabla relación con la clave compuesta LoanId, BookId.

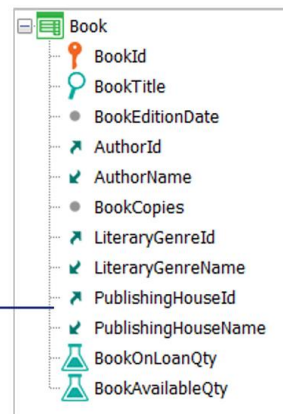
Definimos entonces la transacción BookLoan con la llave primaria compuesta por LoanId, BookId, y la siguiente estructura.

¿Qué tablas se generan? Las mismas que hemos visto anteriormente.

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 |



Llegados a este punto, nos resta resolver el modelado para la solicitud de nuevos ejemplares de libros a una determinada Editorial.

Para eso vamos a definir la transacción BookRequest, que representará una solicitud a una Editorial, con los siguientes atributos:

- **BookRequestId**, basado en el dominio Id
- **BookRequestDate**, con la fecha actual por defecto
- **PublishingHouseId**, ya que la solicitud es a una Editorial
- **PublishingHouseName**, que será un atributo inferido a partir de la clave foránea PublishingHouseId.

Entre esta nueva entidad y el Libro (Book) existe una relación N-N, ya que una solicitud incluye varios libros, y a su vez un libro puede estar incluido en varias solicitudes. Ya hemos analizado que existen varias opciones de diseño.

Vamos a agregar Book como segundo nivel en la Solicitud. Recordemos que es un requerimiento importante controlar que los libros indicados sean publicados por la Editorial a cargo.

Si recordamos la estructura de la transacción Book, vemos que tiene PublishingHouseId como FK, ya que un libro es publicado por una Editorial.

¿Podemos entonces agregar ese atributo también en esta nueva transacción para que sea inferido por BookId y poder controlar si efectivamente se trata de un libro publicado por la Editorial a la cual se le hace la solicitud?

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?



¿Qué ocurre cuando intentamos agregarlo? GeneXus nos da un error, porque el atributo PublishingHouseId ya se encuentra declarado en la estructura de la transacción y no podemos agregarlo nuevamente.

Tenemos un problema de referencia múltiple. Necesitamos referenciar más de una vez al concepto de la Editorial. Entonces, ¿cómo podemos resolver un conflicto de este tipo? Con el uso de subtipos. Esto significa definiendo nuevos atributos, con otros nombres, que hagan referencia al mismo concepto de la Editorial.

Pero... ¿dónde definimos el subtipo? ¿En el concepto de la Editorial que se referencia en el cabezal, o que se referencia en el 2do nivel? ¿Es lo mismo?

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?

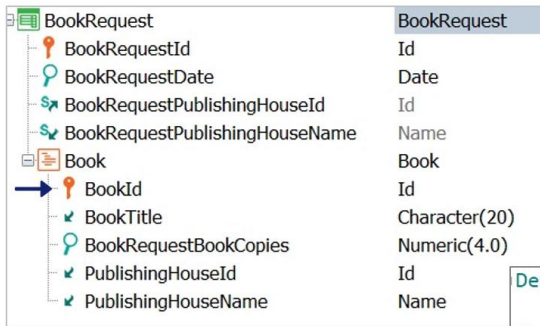


Observemos el diseño de transacciones: el atributo PublishingHouseId está presente en la estructura de la transacción Book como FK, por lo tanto, de BookId GeneXus infiere dicho valor.

Si luego tanto BookId como PublishingHouseId se encuentran presentes en otra transacción, como sucedería en este caso en el 2do nivel de BookRequest, entonces GeneXus aplicará esa relación y PublishingHouseId será una FK inferida.

Entonces, ¿sería correcto cambiar aquí el nombre de los atributos para definir el grupo de subtipos necesario? No, porque estaríamos cortando esta relación, y permitiríamos al usuario agregar un valor de PublishingHouseId diferente al definido previamente en 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 |

Miremos entonces el cabezal de BookRequest. Aquí el atributo PublishingHouseId es una FK común, que no se infiere de ningún lado, por lo tanto, podemos quitarla perfectamente, definir nuevos atributos y declararlos como subtipos de PublishingHouseId y PublishingHouseName. Luego sí podremos compararlos en una regla Error para validar que los libros solicitados sean de la Editorial indicada.

Para finalizar el requerimiento declaramos las siguientes reglas. Como esta regla Error involucra atributos de ambos niveles, entonces es una regla que GeneXus asocia al 2do nivel. Por lo tanto, será disparada para cada línea. Esto significa que al ingresar un nuevo BookId, se evaluará la condición, y en caso de ser verdadera, se disparará el error.

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 |

Hemos resuelto el requerimiento. ¿Y si hubiéramos definido todo el segundo nivel como un grupo de subtipos?

También lo resolveríamos, con la diferencia que (tal como analizamos detalladamente en el video de subtipos) si bien esta solución puede resultar menos obvia y conducir a tener que crear más subtipos para poder inferirlos donde se precisen, tiene como principal ventaja que la ambigüedad se resuelve en la propia tabla que la provoca.

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 |
|-------------------------|
| BookRequest |
| BookRequestId |
| BookRequestDate |
| PublishingHouseId |
| PublishingHouseName |
| Book |
| BookId |
| BookTitle |
| BookRequestBookCopies |
| BookPublishingHouseId |
| BookPublishingHouseName |

```
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 |

¿Podríamos proponer alguna otra opción para evitar la definición de los subtipos en la transacción de dos niveles?

Sí, pero debemos dejar claro que como criterio de trabajo sugerimos siempre abordar el conflicto de referencia en la transacción que lo presenta. Por eso lo hemos resuelto en la propia transacción BookRequest.

Ahora bien. Podríamos observar la transacción Book, y definir allí el grupo de subtipos.

¿Qué ganamos? Que en la transacción BookRequest no exista conflicto de referencia.

Pero debemos tener presente que en caso de tener consultas ya definidas sobre Book, por ejemplo, un catálogo de libros, deberán ser actualizadas porque se cambiaron los nombres de algunos atributos.

Por último, ¿se podría evitar el uso de subtipos? Sí, por ejemplo, llamando en cada línea, y antes de su grabación, a un proceso que devuelva si el libro es publicado por la Editorial indicada o no.

Se evita la definición del grupo de subtipos, pero se invoca en cada línea a un proceso para que evalúe una condición que puede ser resuelta en la definición de la propia transacción.

Summary

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



Por todo lo visto, y para finalizar, recordemos que siempre debemos analizar, evaluar la realidad a modelar e implementar la opción que consideremos correcta, y para eso es fundamental trabajar siempre junto al usuario final, que será quien nos guíe en la selección del diseño.

No olvidemos que la aplicación debe ser una herramienta de apoyo al usuario final para un mejor manejo y desarrollo de su negocio.

GeneXus™

training.genexus.com
wiki.genexus.com
training.genexus.com/certifications