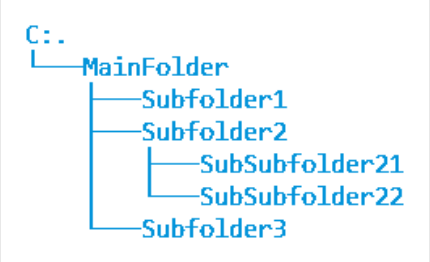
对自引用进行查询

1 简介

有时，用户希望创建一个更通用的域模型，以便更灵活地设置数据的类型和结构。在这种情况下，会经常使用继承或自引用，以便能够设计出简单而高效的模型。这样会使构建微流和应用程序逻辑变得更容易，但查询正确对象可能会变得更具挑战性：尤其是在使用自引用时。

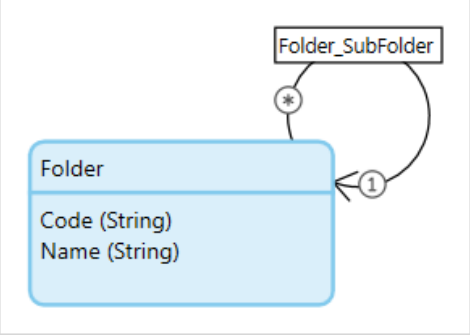
2 示例

该示例关于计算机上的文件夹实现，其中一个文件夹可包含多个子文件夹。



为此，会使用对**文件夹**的自引用。自引用是一个名为 **Folder_SubFolder** 的关联。这可用于构建拥有无限文件夹数量和级别的文件夹结构。

{{% alert type=“info” %}} 在这种情况下，关联为一对多关联，但操作方法同样适用于多对多或一对一关联。{{% /alert %}}

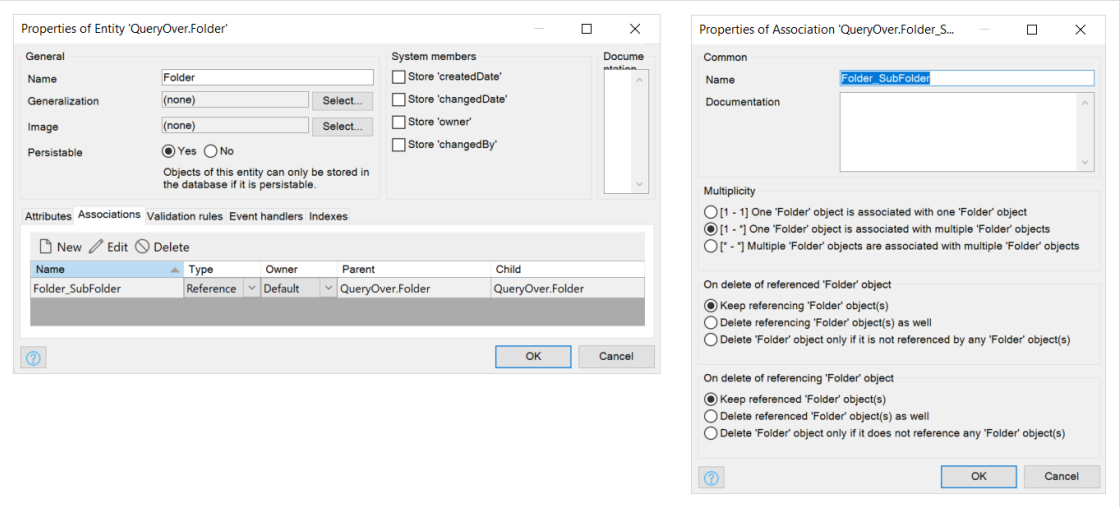


如果在一个名为 **QueryOver** 的模块中创建文件夹功能，则域模型中以两种方式描述关联 **Folder_SubFolder**：

名称	类型	所有者	父	子
Folder_Subfolder	引用	默认	QueryOver.Folder	QueryOver.Folder

- 多重性：一个“文件夹”对象与多个“文件夹”对象关联

子为关联的**所有者** - 换言之，关联始终通过子级进行更新。



上述示例中有六个文件夹，数据库为结构化数据库并已填充下示的属性。在 **Folder_SubFolder** 表中，**ChildFolderID** 显示在左侧，因为它是关联的所有者。

Folder		
ID	Code	Name
1	202002141322015	MainFolder
2	202002141339730	SubFolder1
3	202002141355334	SubFolder2
4	202002141371436	SubFolder3
5	202002141386871	SubFolder21
6	202002141401906	SubFolder22

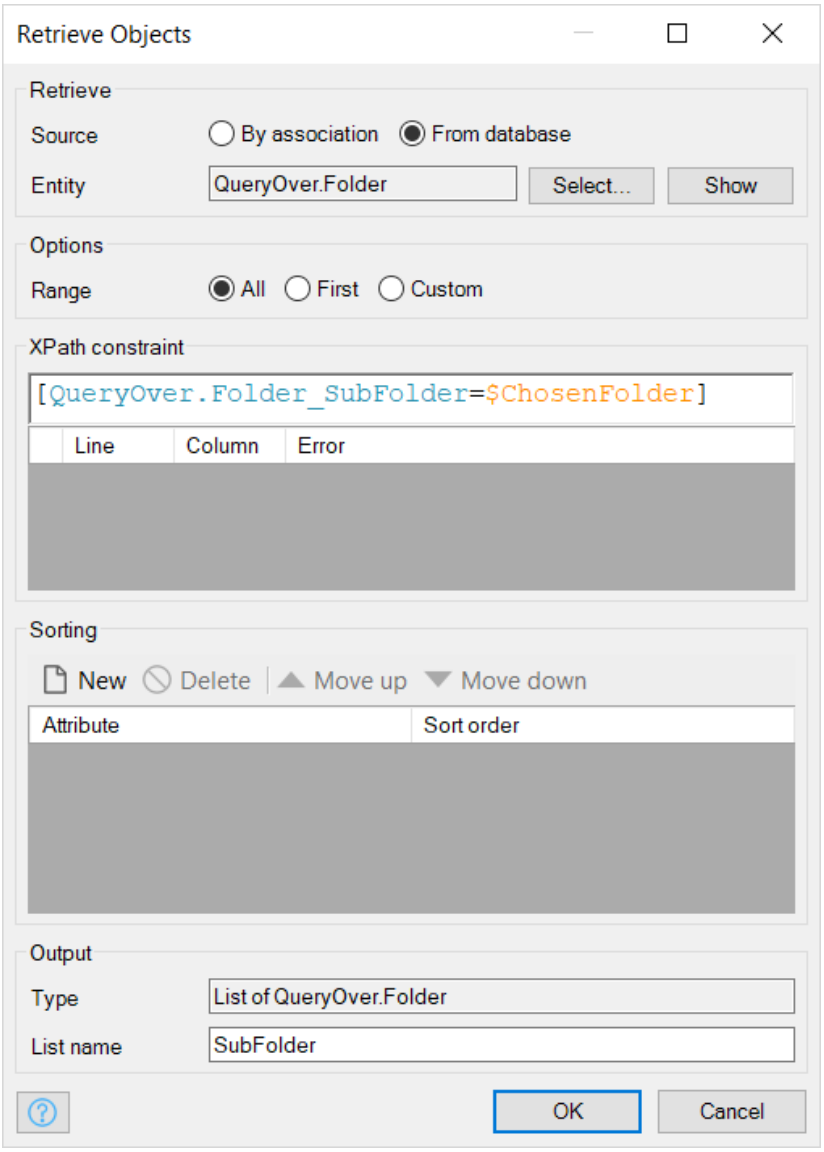
Folder_SubFolder	
ChildFolderID	ParentFolderID
2	1
3	1
4	1
5	3
6	3

关于如何在数据库中实现域模型的更多信息，请参见*域模型*的实现部分。

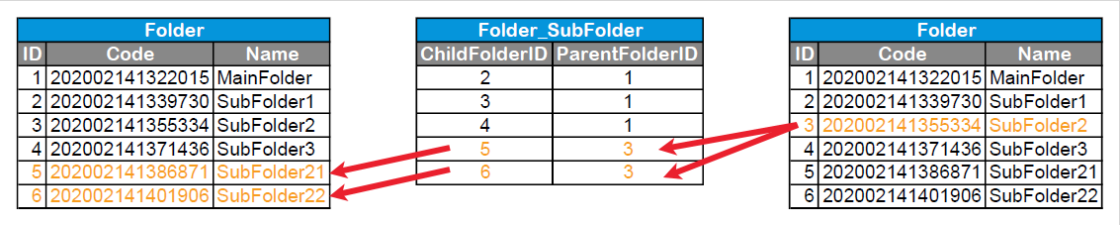
2.1 从文件夹（父）检索子文件夹（子）

如果微流中提供了 $ChosenFolder 对象，则可轻松检索子文件夹。每个关联均有右侧（关联中的父级）和左侧（关联的子级或所有者）。平台读取每个关联并确定父级是否等于 $ChosenFolder。

这通过使用以下 XPath 约束来实现：`[QueryOver.Folder_SubFolder=$ChosenFolder]`。XPath 约束从右向左读取，所产生的文件夹即为结果。这对于如何解释关联的跟随方向十分关键。



如果 $ChosenFolder 对象的**代码**为 `202002141322015`，**名称**为 `SubFolder2`，并已选择 ID 号为 **ID** 3 的文件夹，左侧表中两个以橙色高亮显示的文件夹会予以返回。默认情况下，平台在关联右侧/父侧应用约束，并返回相关的子文件夹。



Folder		
ID	Code	Name
1	202002141322015	MainFolder
2	202002141339730	SubFolder1
3	202002141355334	SubFolder2
4	202002141371436	SubFolder3
5	202002141386871	SubFolder21
6	202002141401906	SubFolder22

Folder_SubFolder	
ChildFolderID	ParentFolderID
2	1
3	1
4	1
5	3
6	3

Folder		
ID	Code	Name
1	202002141322015	MainFolder
2	202002141339730	SubFolder1
3	202002141355334	SubFolder2
4	202002141371436	SubFolder3
5	202002141386871	SubFolder21
6	202002141401906	SubFolder22

2.2 从文件夹检索父文件夹

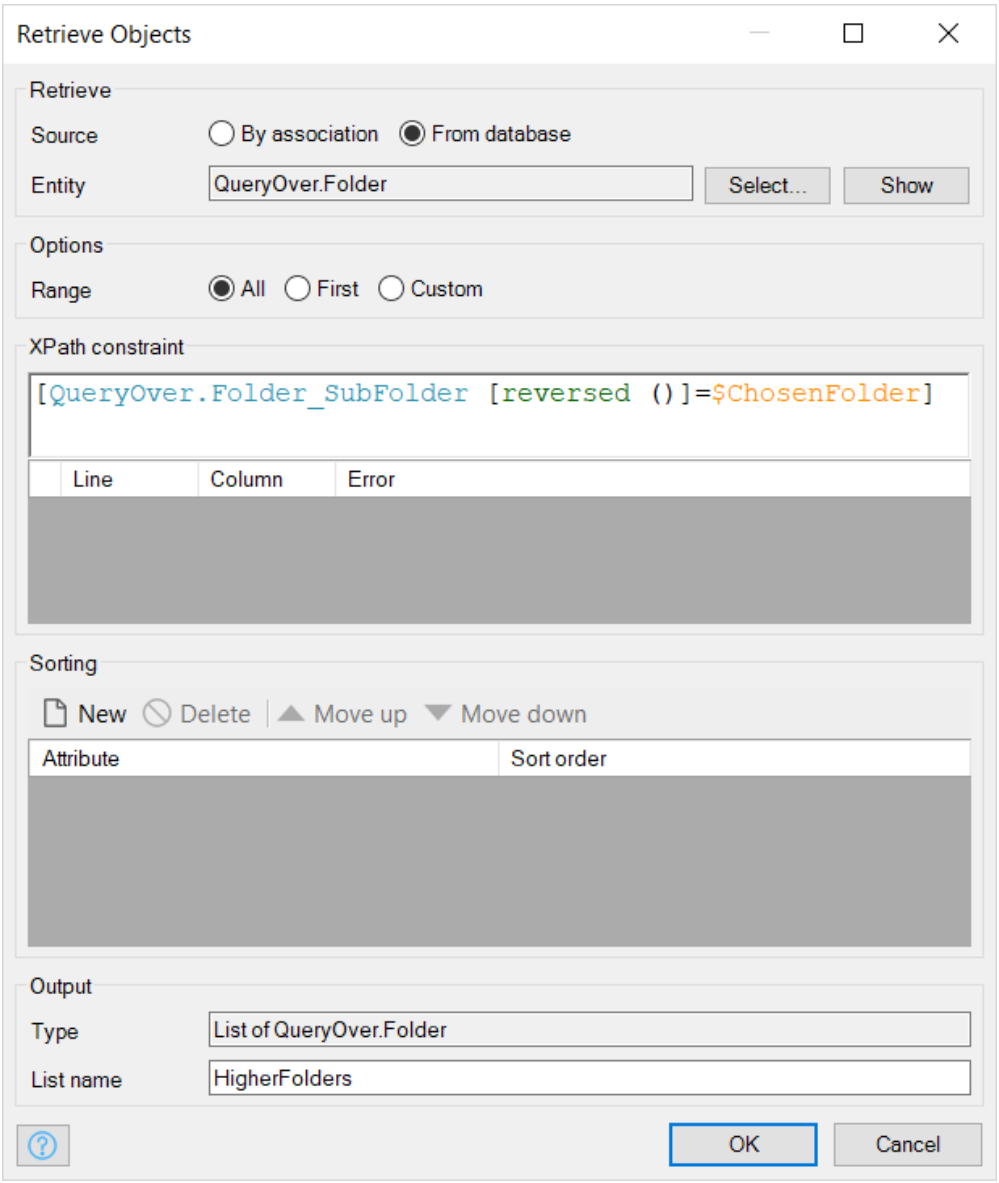
当有 $ChosenFolder 对象可用并且要从数据库检索其父文件夹（层次结构中下一更高级别的文件夹，例如给定 **SubFolder2**，要检索 **MainFolder**）时，情况会变得更加复杂。

使用表达式 `[reversed ()]`，可指示 Mendix 按照通常使用的相反方向读取约束。

`[reversed()]` 仅适用于一个关联。如果有多个关联，则将继续按正常方式予以解释。参见下述“创建更复杂的查询”。

`[reversed()]` 表达式只能应用于自引用。当两个不同对象类型之间存在关联时，平台将能够自动确定连接的方向。

在本示例中，要查找 $ChosenFolder 的父文件夹。现在，查询变为 `[QueryOver.Folder_SubFolder [reversed ()]=$ChosenFolder]`。关联从左到右读取，而不是从右到左读取（父到子）。



如果 $ChosenFolder 对象的**代码**为 `202002141322015`，**名称**为 `SubFolder2`，并已选择 **ID** 3 的文件夹，将返回右侧表中以橙色高亮显示的文件夹。平台在关联左侧/子侧反向应用约束，并返回相关的父文件夹。

Folder

ID	Code	Name
1	202002141322015	MainFolder
2	202002141339730	SubFolder1
3	202002141355334	SubFolder2
4	202002141371436	SubFolder3
5	202002141386871	SubFolder21
6	202002141401906	SubFolder22

Folder_SubFolder

ChildFolderID	ParentFolderID
2	1
3	1
4	1
5	3
6	3

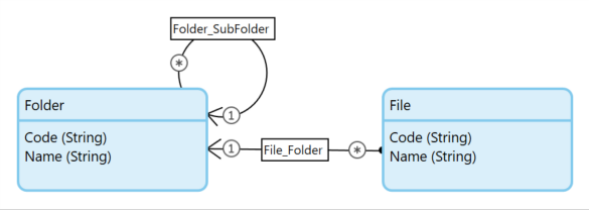
Folder

ID	Code	Name
1	202002141322015	MainFolder
2	202002141339730	SubFolder1
3	202002141355334	SubFolder2
4	202002141371436	SubFolder3
5	202002141386871	SubFolder21
6	202002141401906	SubFolder22

2.3 创建更复杂的查询

上一个示例为简单查询示例。但是，可以在更复杂的查询中使用 `[reversed()]` 表达式。

例如，每个文件夹可包含多个文件，通过关联 **File_Folder** 与文件夹相关联。



用户想要检索文件夹对象 $ChosenFolder 的父文件夹中的所有文件。

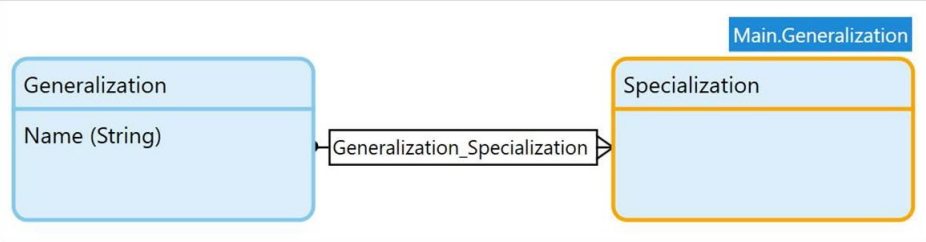
使用约束 →`QueryOver.File_Folder/QueryOver.Folder/QueryOver.Folder_SubFolder →reverse（）→→$ChosenFolder`→，返回与**文件夹**关联（作为父文件夹）的所有**文件**对象，其中作为父文件夹的文件夹关联的**文件夹**与 **$ChosenFolder** 相同。

如果 $ChosenFolder 对象为 `SubFolder2`，将检索通过关联 **File_Folder** 与 `MainFolder` 关联的所有**文件**对象。

3 到专用化的关联

在特殊的自引用情况下，当一对多关联自带专用化时，无法按关联进行检索。

以下是一个继承示例：



在该示例中，如果输入为专用化，当在微流中使用标准的按关联检索时，将无法检索**专用化**列表。

但是，对于这种限制，有一种解决方法：“专用化”列表可通过使用 Java API 的 Java 操作进行检索。该 Java 操作需要两个参数：**专用化**和布尔值**反向**，代码片段如下所示：

```
public class RetrieveAsAssociatedWithB extends CustomJavaAction<java.util.Lis
t<IMendixObject>>
{
    private IMendixObject __B;
    private main.proxies.Specialization B;
    private java.lang.Boolean Reverse;

    public RetrieveAsAssociatedWithB(IContext context, IMendixObject B, java.
lang.Boolean Reverse)
    {
        super(context);
        this.__B = B;
        this.Reverse = Reverse;
    }

    @java.lang.Override
    public java.util.List<IMendixObject> executeAction() throws Exception
    {
        this.B = __B == null ? null : main.proxies.Specialization.initialize
(getContext(), __B);

        // BEGIN USER CODE
        return Core.retrieveByPath(getContext(), __B, "Main.Generalization_Sp
ecialization", Reverse);
        // END USER CODE
    }
}
```

{{% alert type=“info” %}} 确保导入 `com.mendix.core.Core`，以便能够在该代码片段中执行 `Core.retrieveByPath(..)`。{{% /alert %}}

当反向布尔值设为 true 并使用专用化对象作为输入时，返回的列表将包含关联到专用化的所有泛化。