Practice exercises

GeneXus™ 16

March 2019

CONTENTS

THE PROBLEM

A multinational company in charge of managing amusement parks hires you to develop a system for storing and managing the information with which the company works. Imagine that the system consists of two modules:

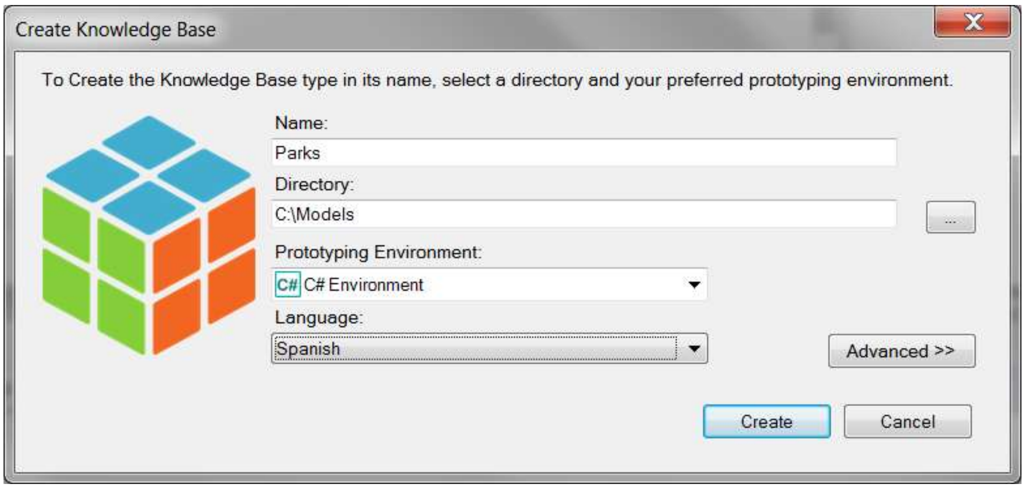
- Backend: part of the app to be run on a web server so that the company's employees may handle the information from any location connected to the Internet.
- Simple application for mobile devices: part of the app intended to be downloaded by the company's customers to allow them queries on the countries available and on the main amusement parks available in each city, including the games offered.

NEW PROJECT, NEW KNOWLEDGE BASE

GO to GeneXus and create a knowledge base called *Parks* to start developing the application.

We suggest:

- Select C# as the development environment. Ensure the installation of everything necessary (including SQL Server). If you use GeneXus Trial, the generation environment with C# and SQL Server is predefined already, prototyping in the Amazon cloud.
- That you should not create the knowledge base in the "My Documents" folder or any other folder under "Documents and Settings" because these folders have special permits from Windows.


Create Knowledge Base

To Create the Knowledge Base type in its name, select a directory and your preferred prototyping environment.

Name: Parks

Directory: C:\Models

Prototyping Environment: C# Environment

Language: Spanish

Take a few minutes to **become familiarized** with the **IDE (GeneXus integrated development environment)**. Try to **move windows, visualize specific windows desired** (View and View/Other Tool Windows) and note the contents of the **KBExplorer** (Knowledge Base Explorer) window. You will see **domains,** some **objects**, and **images**, among other things, appear initialized already.

Recommendation: keep the properties window open (**F4**), because you will be using it permanently. In the **'Preferences'** window, where the **'Environment'** is configured.

INITIAL TRANSACTIONS

During your meetings with the company, they tell you:

> "We record the data on amusement parks to manage both their employees and their games and activities available to visitors."

To start building the application, we must begin by identifying the actors from reality, to represent them by means of **transactions**. What transactions should we create then in the knowledge base (KB)?

"EMPLOYEE" TRANSACTION

We ask: what data do the company´s employees record? The answer is:

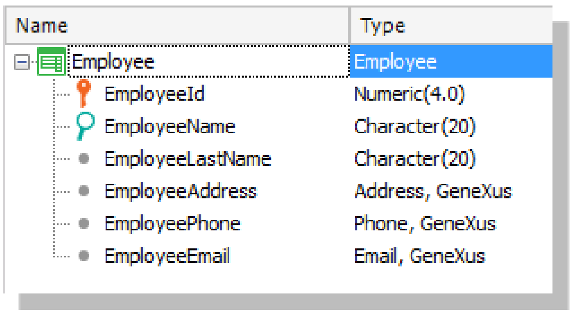
> The **name** (not over 20 characters), **surname** (not over 20 characters either), **address, telephone** and the **e-mail**.

With this data, you may already create the Employee transaction.

Remember that:

- There are several alternatives to create objects:
  - Doing it from the menu: File/New/Object
  - Ctrl+N
  - Icon of the tool bar
- You will need an attribute to identify each employee (EmployeeId).
- If you type dot (".") when you are about to enter a new attribute, it will be initialized with the name of the transaction.

The structure of the transaction should look as shown below:



Name	Type
Employee	Employee
EmployeeId	Numeric(4.0)
EmployeeName	Character(20)
EmployeeLastName	Character(20)
EmployeeAddress	Address, GeneXus
EmployeePhone	Phone, GeneXus
EmployeeEmail	Email, GeneXus

Remember that:

- **Address, Phone** and **Email** are **semantic domains** assigned automatically to the attributes defined and they respectively contain the texts Address, Phone or Email in their names.
- When you define the data type of the identifier attribute, instead of using Numeric(4.0) directly, define the **Id domain** with that data type.
  Set up the **Autonumber** property of that domain as **True**, so that all the attributes based on it are automatically numbered, without the need for the user to be concerned with that.

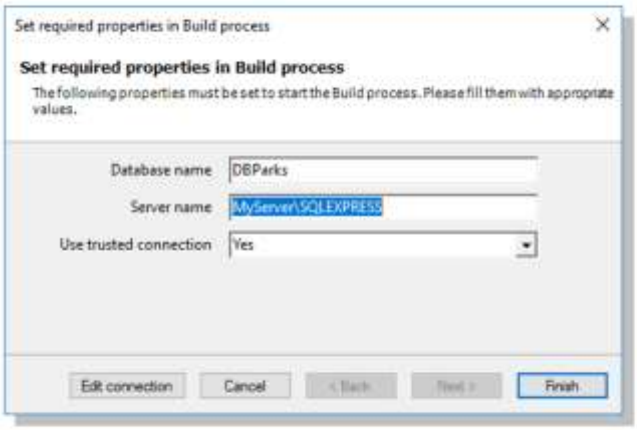
The next step is to test the application in runtime. Make sure that you have the GeneXus *Output* window enables and at sight. (*View/Other Tool Windows /Output*).

Now you may **test** the application in runtime by pressing **F5**.

What will happen?

Solution

If you decide to create the database and programs **locally**, a window like the one shown below will open up for you to enter data on the Database, Server and connection method. Remember that if a database with the name indicated on that server does not exist, thin GeneXus will **create it**.



Set required properties in Build process

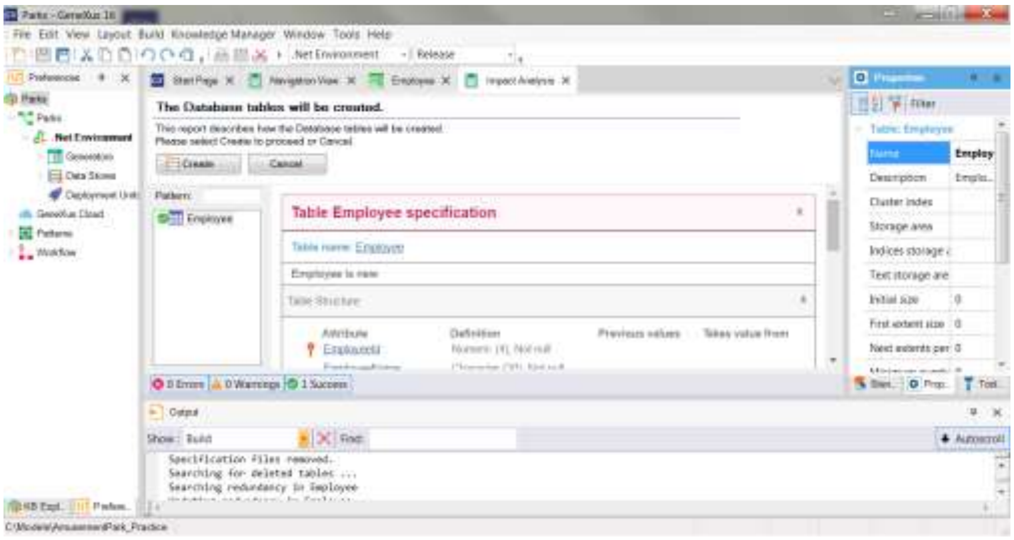
Set required properties in Build process

The following properties must be set to start the Build process. Please fill them with appropriate values.

Database name	DBParks
Server name	MyServer\SQLEXPRESS
Use trusted connection	Yes

If the database and programs are created in the cloud, then the dialog shown above will not appear because GeneXus knows the server data in the cloud and automatically sets up the name of the database and all the information relative to connections to it.

Then comes the deployment of an **Impact Analysis** detailing that the database will be created, with the EMPLOYEE table inside it:



The Database tables will be created.

This report describes how the Database tables will be created.
Please select Create to proceed or Cancel.

Table Employee specification

Table name: Employee

Employee is new

Attribute	Definition	Previous values	Takes value from
EmployeeId	Numeric (4), Not null		

```
Specification files removed.
Searching for deleted tables ...
Searching redundancy in Employee
```

If you press the *Create* button, GeneXus will execute the program that will be doing the creation. Upon the process´ end, the menu with links to execute the objects defined will be opened in the navigator set up as the predetermined one. In this case, there is only one: the Employee transaction.

Enter some employees in the system. Then modify some data on any of the previously entered employees and delete any employee.

You may also use the arrows provided for moving from one employee record to the next, as well as the *SELECT* option that offers a "selection list" to view the list of recorded employees in order to select one.

Now let´s identify and create the following transaction. Remember what their statement was, to which some additions were made:

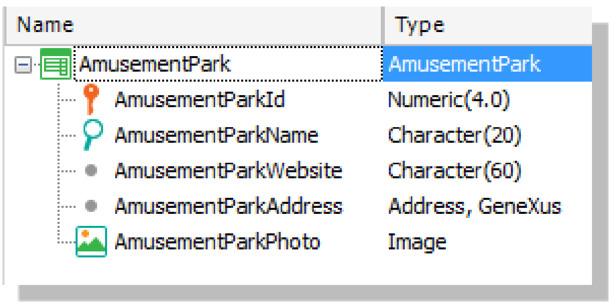
"We record the data of **amusement parks** in different cities in different **countries**, to manage both their employees and their games and activities available to visitors."

"AMUSEMENTPARK" AND "COUNTRY" TRANSACTIONS, RELATED

We asked the company's employees the following question: what are the data of amusement parks that you record for your work? The answer was:

The **name** (with up to 40 characters), **website** (with up to 60 characters), **address** and an **emblematic photo**.

With this data, it is now possible to create the AmusementPARK transaction.



Name	Type
AmusementPark	AmusementPark
AmusementParkId	Numeric(4.0)
AmusementParkName	Character(20)
AmusementParkWebsite	Character(60)
AmusementParkAddress	Address, GeneXus
AmusementParkPhoto	Image

Remember that:

- **Address** is a **semantic domain** automatically assigned to the attributes defined by containing the text Address in its name.
- When you define the data type of the identifier attribute, instead of using Numeric(4.0) directly, define the **Id domain** with that data type.
- Set up the **Autonumber property** in that domain as **True**, so that all attributes based on it will be numbered automatically, without the user being concerned about that.

We will create a transaction to record the <u>countries</u> to which the amusement parks belong.

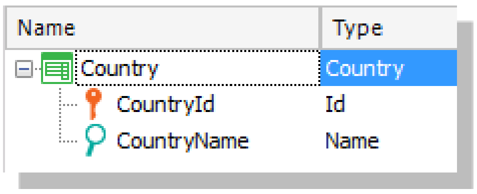
<u>Remember that</u>:

- By pressing dot (".") when you are about to name an attribute in the structure of the transaction, it will appear initialized with the name of the transaction.
- You will need an identifier attribute, CountryId.

In defining the data type of the identifier attribute, instead of using *Numeric(4.0)* directly, define the Id domain with that data type.

Set up the Autonumber property of that domain as ***True***, so that all attributes based on it will be numbered automatically without the user being concerned about that.

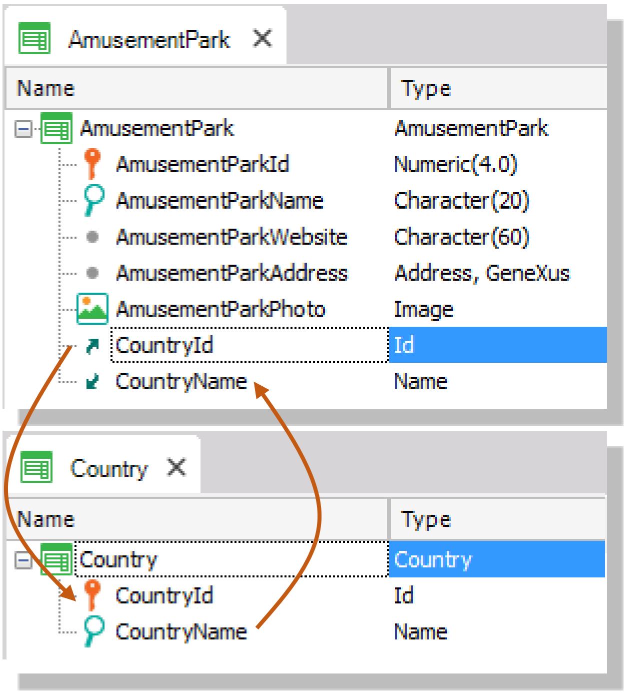
Define the ***CountryName*** attribute creating and using a new domain: ***Name***=*Character(50).*



Name	Type
Country	Country
CountryId	Id
CountryName	Name

Now we will go back to the AmusementPARK transaction to add the *CountryId* and *CountryName* attributes.

Let's take this chance to change the data type of *AmusementId*, assigning to it the Id domain created previously, if you have not configured it yet.



AmusementPark

Name	Type
AmusementPark	AmusementPark
AmusementParkId	Numeric(4.0)
AmusementParkName	Character(20)
AmusementParkWebsite	Character(60)
AmusementParkAddress	Address, GeneXus
AmusementParkPhoto	Image
CountryId	Id
CountryName	Name

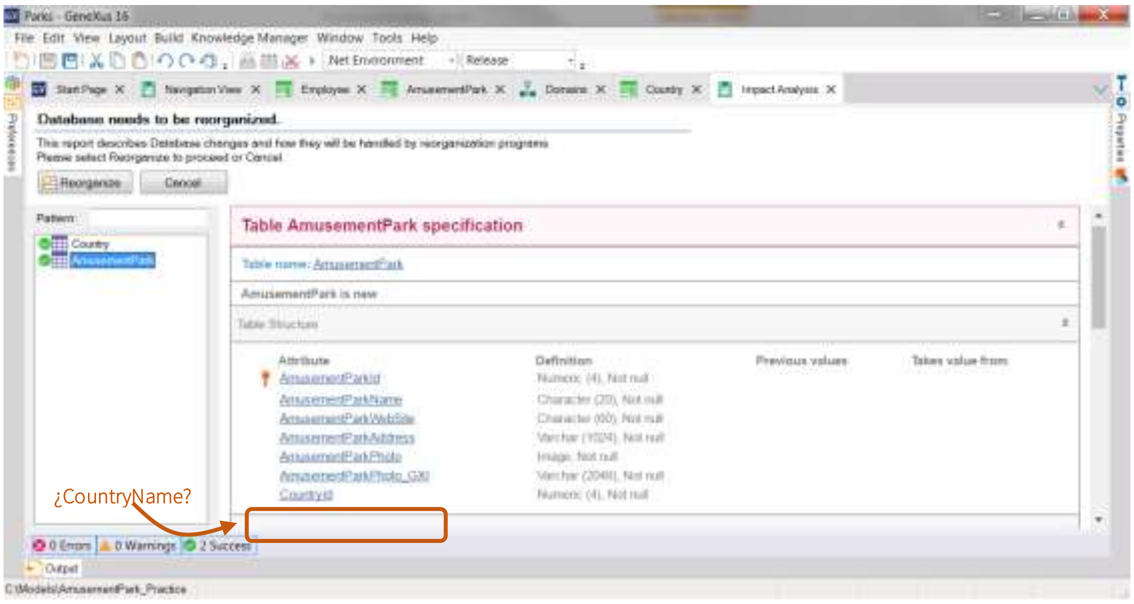
Country

Name	Type
Country	Country
CountryId	Id
CountryName	Name

Why did you include the *CountryName* in *AmusementPark* attribute in addition to *CountryId?*

Answer: It is important that the *CountryName* attribute appear in the screen of the transaction to show us the name of the country, which is the data we best recall about the country, instead of just viewing its identifier. It is also necessary to add the attribute in the structure of the transaction if we want to use it later, for example, as part of a rule.

Execute to test (F5) and you will get the following report:



Database needs to be reorganized.

This report describes Database changes and how they will be handled by reorganization programs
Please select Reorganize to proceed or Cancel

Table AmusementPark specification

Table name: AmusementPark

AmusementPark is new

Table Structure

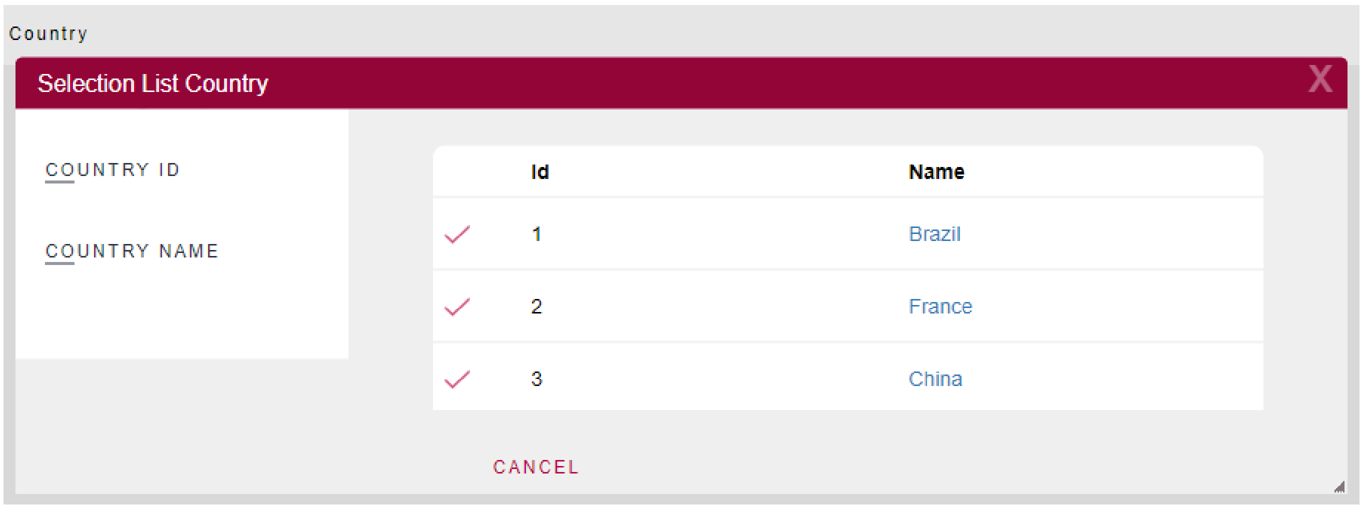
Attribute	Definition	Previous values	Takes value from
AmusementParkId	Numeric (4), Not null		
AmusementParkName	Character (20), Not null		
AmusementParkWebSite	Character (60), Not null		
AmusementParkAddress	Varchar (1024), Not null		
AmusementParkPhoto	Image, Not null		
AmusementParkPhoto_GXI	Varchar (2048), Not null		
CountryId	Numeric (4), Not null		

C:\Models\AmusementPark_Practice

Why doesn't the *CountryName* attribute not appear in the *AmusementPark* table, which GeneXus indicates that must be modified in the database? In other words, why will the physical table not contain it, when it is in the structure of the transaction?

After studying the report, if we agree, we press ***Reorganize*** in order for what is informed to be actually carried out. The navigator will open up with the menu with links to the 3 programs corresponding to each transaction (*AmusementPark*, *Country,* and *Employee*).

Enter the following as **countries**: Brazil, France, and China. Note that, by leaving value 0 as the value in the identifier, upon saving, the number following the last one assigned is automatically assigned (in fact, it is being autonumbered).


Country

Selection List Country

COUNTRY ID

COUNTRY NAME

	Id	Name
✓	1	Brazil
✓	2	France
✓	3	China

CANCEL

Enter the amusement park "Beto Carrero World", located in Brazil. If you don't remember the identifier for Brazil in the system, how can you enter the country? There is an icon with an arrow next to *CountryId* for opening a "Selection List" of countries that is created automatically by GeneXus. This is because *CountryId* has the foreign key role in this transaction (that is, it is "pointing at" another table).

RELATED DATA: HOW TO MAINTAIN INTEGRITY

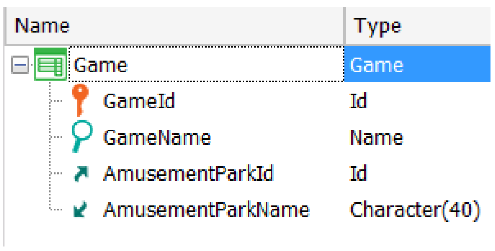
AmusementPark and *Country* are related. When we place *CountryId* in the structure of *AmusementPark*, because the name is the same as the name of the attribute that is the primary key in the Country transaction, GeneXus will understand that, in *AmusementPark,* the *CountryId* attribute is foreign key and will automatically maintain the physical integrity of the information. So, for example:

- Try to enter an amusement park with a country Id that does not exist. Are you allowed to save that park?
- Select a park that has been entered previously (for example, 'Beto Carrero World'), and change the country for a non-existent country. Were you able to save that change?
- Try to delete a country (using the Country transaction) with an associated park (such as Brazil). Are you allowed to do that?

Conclusion: the programs corresponding to the transactions ensure data integrity.

'GAME' TRANSACTION

As requested from the start, the system must offer the possibility of entering the games available in each park, so we must create a transaction containing its name and the amusement park where it belongs.



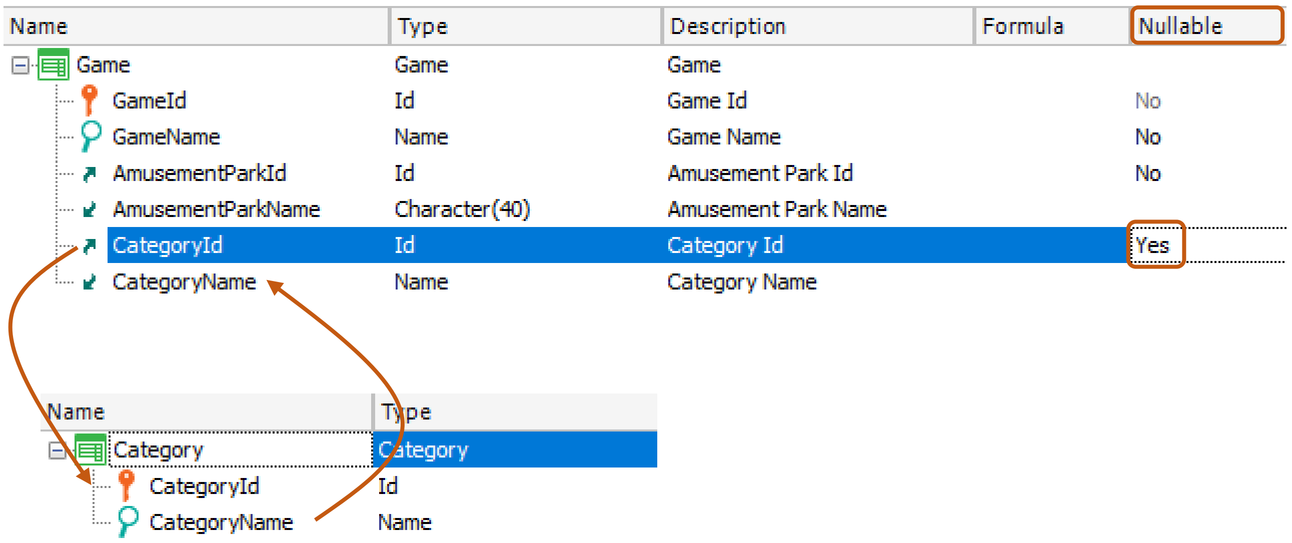
Name	Type
Game	Game
GameId	Id
GameName	Name
AmusementParkId	Id
AmusementParkName	Character(40)

'CATEGORY' TRANSACTION

We still have to complete the information on the Game transaction. The employees explained that they also record the **category** (children, radical, recreational, etc.) to which a game belongs. So, we will have to create a transaction to record that information, and add the category to the Game transaction.

But we were also informed that it is not mandatory to necessarily record the category to which a specific game that is handled belongs. That may be left empty. Knowing that GeneXus controls integrity automatically, how can we do this?

Solution:



Name	Type	Description	Formula	Nullable
Game	Game	Game		
GameId	Id	Game Id		No
GameName	Name	Game Name		No
AmusementParkId	Id	Amusement Park Id		No
AmusementParkName	Character(40)	Amusement Park Name		
CategoryId	Id	Category Id		Yes
CategoryName	Name	Category Name		

Name	Type
Category	Category
CategoryId	Id
CategoryName	Name

To end the definition of the Game transaction, we should add the data missing, that is, the photo.

To do this, create the *GamePhoto* attribute of the *Image* data type.

Get GeneXus to build the app so you can test it in runtime (F5).

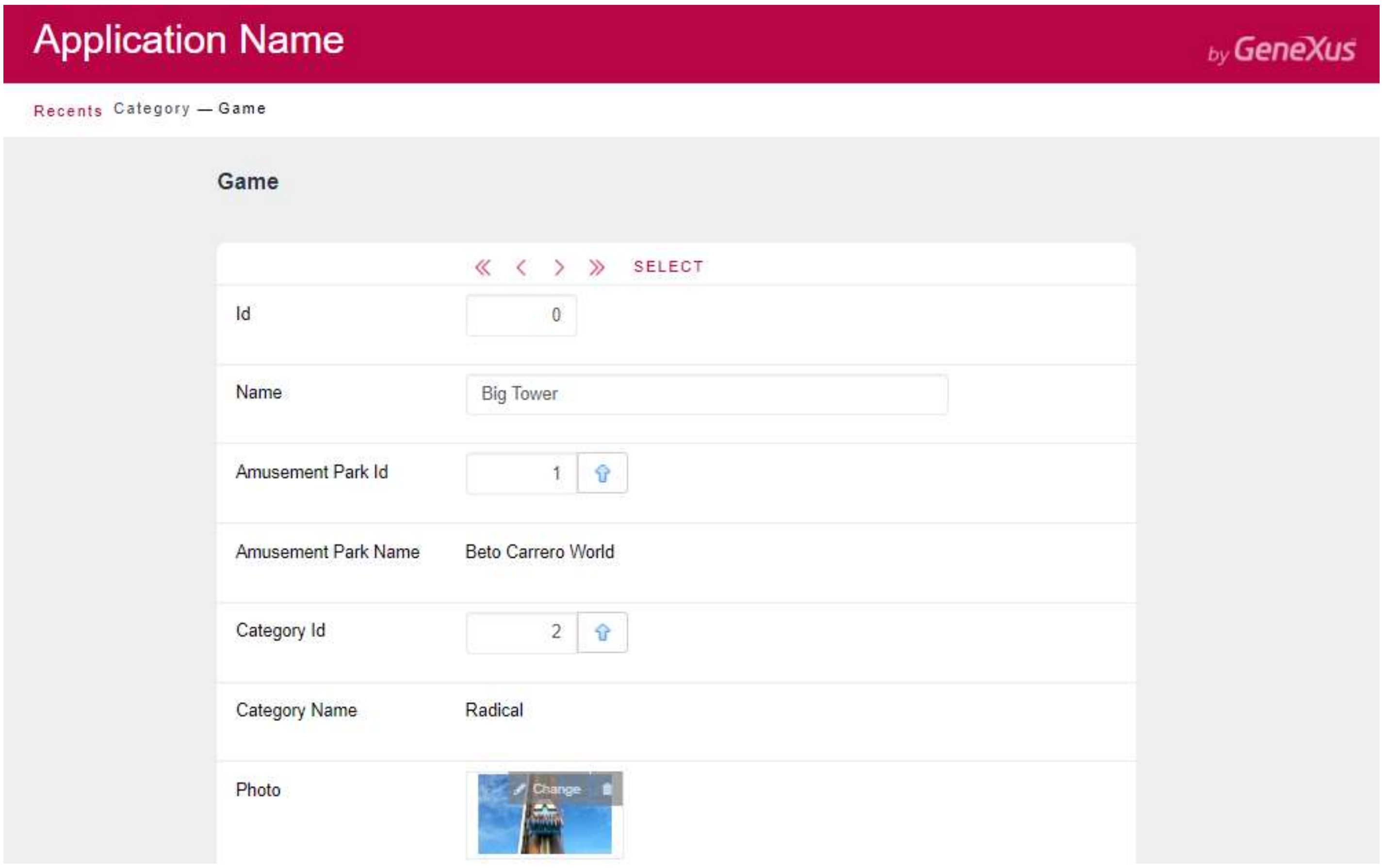
Note what the Impact Analysis report informs. The *Category* and *Game* tables must be created (don't mind figuring out why two values are to be stored per image).

Reorganize and execute.

Enter categories (like children and radical) and games (like rollercoasters and merry-go-rounds).

Note that, in this case, you may leave the category empty (because the Nullable property was configured as Yes in the structure of the transaction).

However, if you try to insert a non-existent value in the value of *CategoryId* for the game, you will be prevented from saving that.


Application Name

by GeneXus

Recents Category — Game

Game

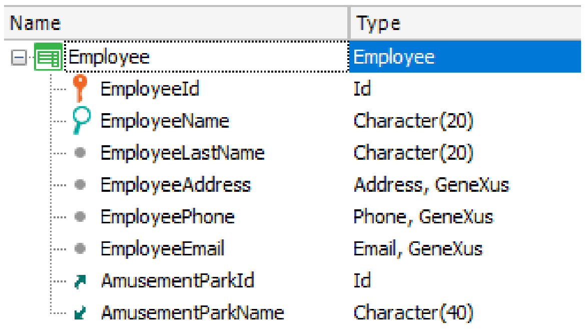
Id	0
Name	Big Tower
Amusement Park Id	1
Amusement Park Name	Beto Carrero World
Category Id	2
Category Name	Radical
Photo	

”EMPLOYEE” AND “AMUSEMENTPARK” TRANSACTIONS, RELATED

As we were told from the start of the app development, the system will manage the amusement parks and their employees. So, we must link the employees recorded with the park where each of them works.

To do that, let’s go to the Employee transaction and add the *AmusementParkId* and *AmusementParkName* attributes.

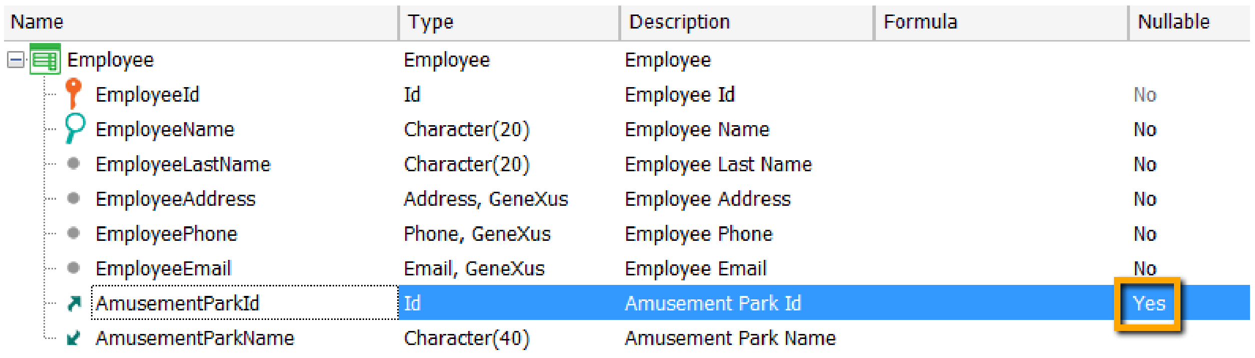
Let’s take this opportunity to change the data type of *EmployeeId* by assigning to it the domain Id previously created, if you still have not configured it.



Name	Type
Employee	Employee
EmployeeId	Id
EmployeeName	Character(20)
EmployeeLastName	Character(20)
EmployeeAddress	Address, GeneXus
EmployeePhone	Phone, GeneXus
EmployeeEmail	Email, GeneXus
AmusementParkId	Id
AmusementParkName	Character(40)

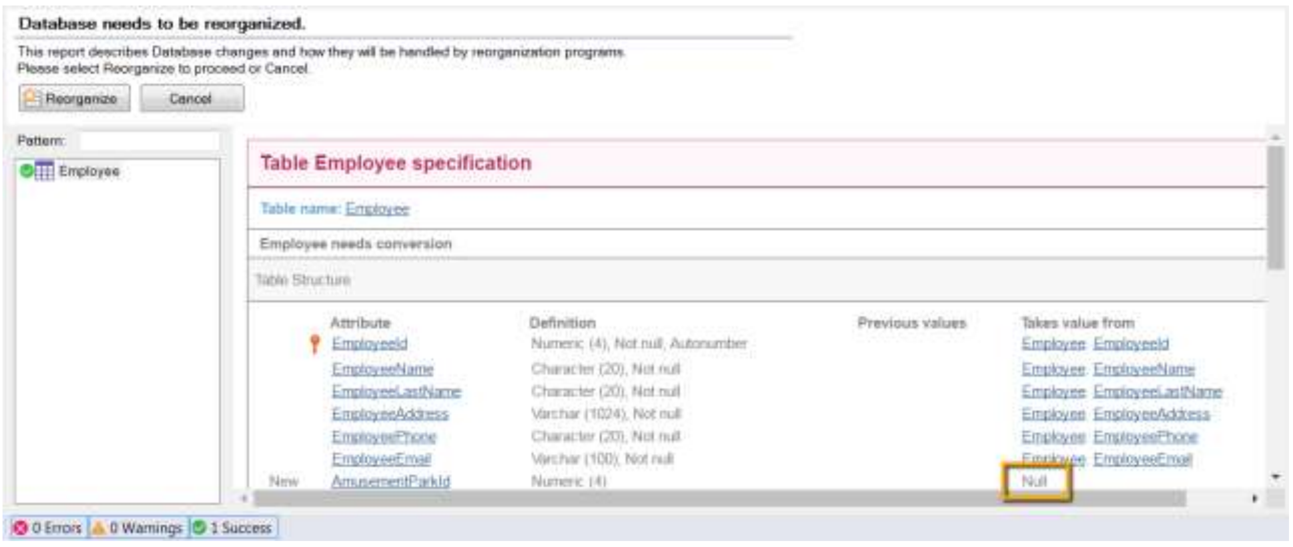
We were informed that it is not mandatory to record, at that moment, the park where the employee works, that is, we can leave it empty. What should we do?

Solution:



Name	Type	Description	Formula	Nullable
Employee	Employee	Employee		
EmployeeId	Id	Employee Id		No
EmployeeName	Character(20)	Employee Name		No
EmployeeLastName	Character(20)	Employee Last Name		No
EmployeeAddress	Address, GeneXus	Employee Address		No
EmployeePhone	Phone, GeneXus	Employee Phone		No
EmployeeEmail	Email, GeneXus	Employee Email		No
AmusementParkId	Id	Amusement Park Id		Yes
AmusementParkName	Character(40)	Amusement Park Name		

Try executing (**F5**) and you will get the following report indicating that the AmusementParkId attribute now enables us to leave a value unspecified:



Database needs to be reorganized.

This report describes Database changes and how they will be handled by reorganization programs.
Please select Reorganize to proceed or Cancel.

Table Employee specification

Table name: Employee

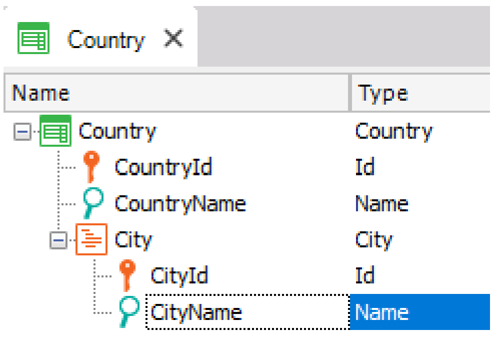
Employee needs conversion

Table Structure

	Attribute	Definition	Previous values	Takes value from
	EmployeeId	Numeric (4), Not null, Autonumber		Employee, EmployeeId
	EmployeeName	Character (20), Not null		Employee, EmployeeName
	EmployeeLastName	Character (20), Not null		Employee, EmployeeLastName
	EmployeeAddress	Varchar (1024), Not null		Employee, EmployeeAddress
	EmployeePhone	Character (20), Not null		Employee, EmployeePhone
	EmployeeEmail	Varchar (100), Not null		Employee, EmployeeEmail
New	AmusementParkId	Numeric (4)		Null

ADDING THE CITIES TO THE 'COUNTRY' TRANSACTION

In addition to the countries, we must also record the information about cities. So, we must add a second level in the Country transaction, with the identifier and the city name.



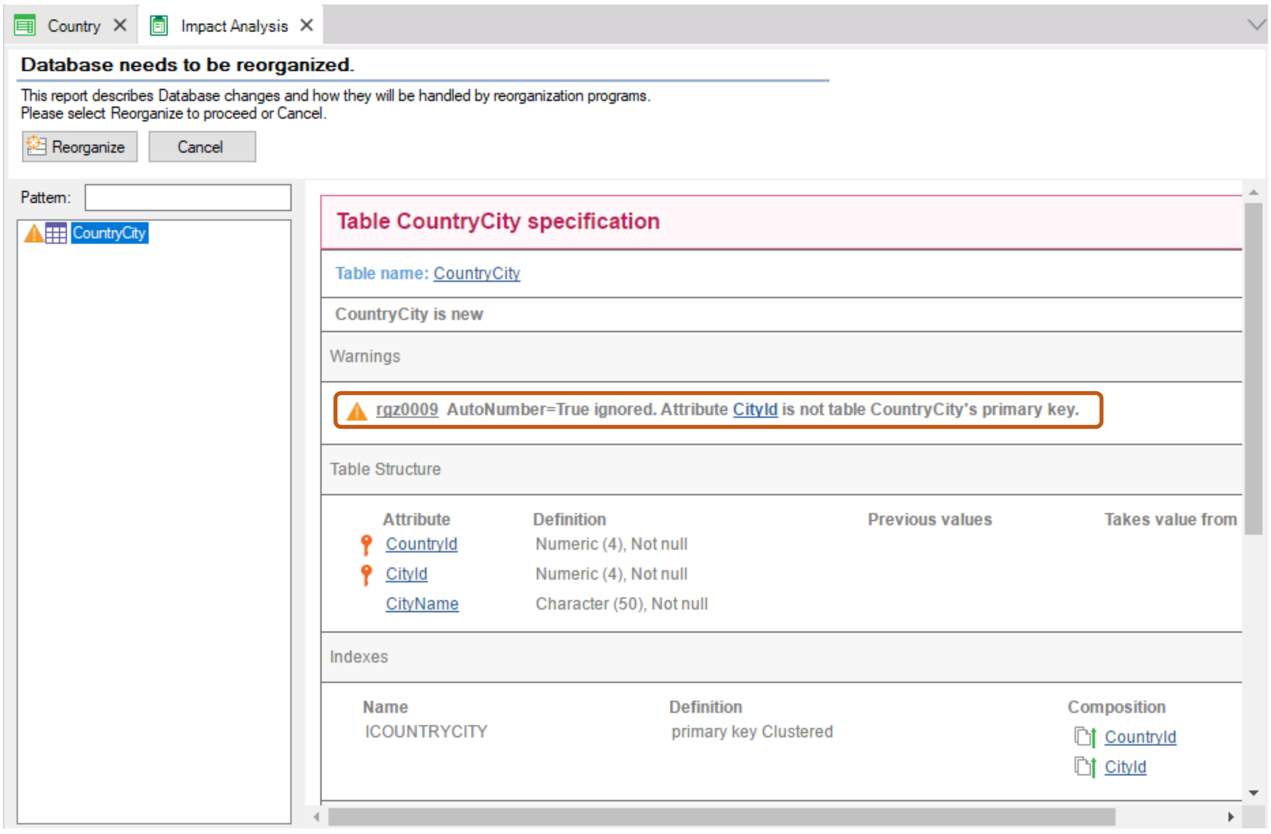
Name	Type
Country	Country
CountryId	Id
CountryName	Name
City	City
CityId	Id
CityName	Name

Remember that:

- Once positioned on the *CountryName* attribute, with a right click and ***Insert Level*** we will be adding the sublevel.
- After you define a name for the new level by typing quotation marks (") instead of dot, the attribute you define will be initialized with the name of the level.
- The cities will be identified by means of their own Id, in combination with the country Id. This means that you will not be able to identify a city without having provided the data on the country in question beforehand. So, it will be possible to have a city 1 Rosario for Uruguay as well as for Argentina:
  Country: 1 (Uruguay) – City: 1 (Rosario)
  Country: 2 (Argentina) – City: 1 (Rosario)

- Or it could also be possible that Rosario in Argentina be identified with a different number:
  Country: 2 (Argentina) – City: 4 (Rosario)

Reorganize and execute (F5).



Database needs to be reorganized.

This report describes Database changes and how they will be handled by reorganization programs.
Please select Reorganize to proceed or Cancel.

Table CountryCity specification

Table name: CountryCity

CountryCity is new

Warnings

rgz0009 AutoNumber=True ignored. Attribute CityId is not table CountryCity's primary key.

Table Structure

Attribute	Definition	Previous values	Takes value from
CountryId	Numeric (4), Not null		
CityId	Numeric (4), Not null		
CityName	Character (50), Not null		

Indexes

Name	Definition	Composition
ICOUNTRYCITY	primary key Clustered	CountryId CityId

<u>Note</u> that the Navigation Listing will inform that:

- The Autonumber property for the case of ***CityId*** will be ignored. This means that, in runtime, the user must manually enter the city identifiers. The explanation is that the Autonumber property only autonumbers simple primary keys, and in this case, *CityId* is the second component of a composite key.
- A new ***CountryCity*** table will be created to store the information corresponding to the cities.

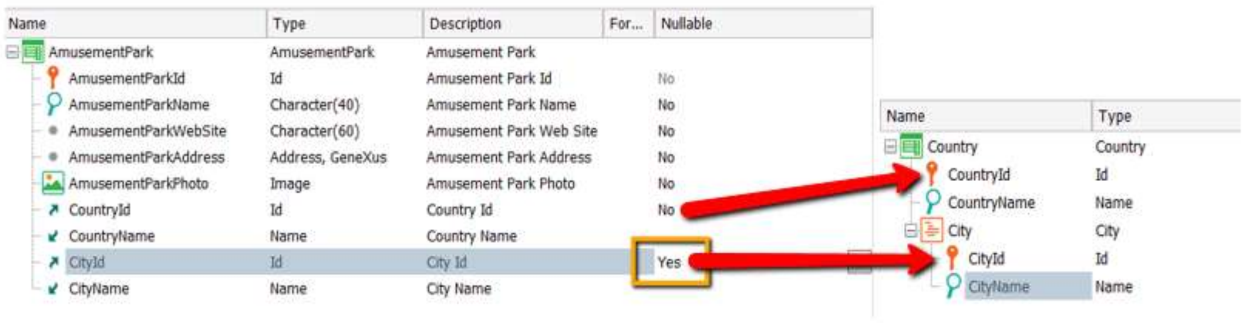
Enter cities for the countries you had already registered.

"AMUSEMENTPARK" TRANSACTION: ADDING THE CITY.

In the AmusementPARK transaction, let's add the city of the country to which the park belongs. What should you do if the company informs you that the value might be unknown or might be irrelevant for a given park at a specific moment?

Build the app and test it (F5 and Reorganize).

Solution:



Name	Type	Description	For...	Nullable
AmusementPark	AmusementPark	Amusement Park		
AmusementParkId	Id	Amusement Park Id		No
AmusementParkName	Character(40)	Amusement Park Name		No
AmusementParkWebSite	Character(60)	Amusement Park Web Site		No
AmusementParkAddress	Address, GeneXus	Amusement Park Address		No
AmusementParkPhoto	Image	Amusement Park Photo		No
CountryId	Id	Country Id		No
CountryName	Name	Country Name		
CityId	Id	City Id		Yes
CityName	Name	City Name		

Name	Type
Country	Country
CountryId	Id
CountryName	Name
City	City
CityId	Id
CityName	Name

ADDING BEHAVIOR TO TRANSACTIONS (RULES)

After they try with us the application that we continue to develop, the company tells us that there is a specific behavior that employees must comply with at the time of handling information through the program (Employee transaction).

Which behavior?

They tell us that:

- "The system should not allow access to employees with no name or surname."
- "Users should be warned when the phone is not assigned, in case it was a mistake."
- "The employee entry date in the system should be recorded (**EmployeeAddedDate**) proposing today's date as the predetermined value for that attribute."

Specify that behavior and test it (F5 and Reorganize).

Remember that:

- Rules end with a semi-colon ";".
- The **IsEmpty()** method applied to an attribute returns True when the attribute is empty, or False otherwise.
- The **&Today** variable is part of the system and has the value of today's date already loaded.

In order to write a variable inside the **Rules** screen, when you type "&" you will be deploying all the variables defined to that time, for you to select the one you need. The other possibility is to use **Insert** / **Variable.**

Try to enter a new employee and leave the name empty. Does the system allow you to save or move to the next field?

Try the same with the surname. Does the same happen in the case of the telephone?

If you are later informed that the date of entry in the system should not be handled by the user, but rather just viewed, how would you establish that behavior?

Specify it and test it in runtime.

PATTERNS: IMPROVING THE INTERFACE TO WORK WITH INFORMATION

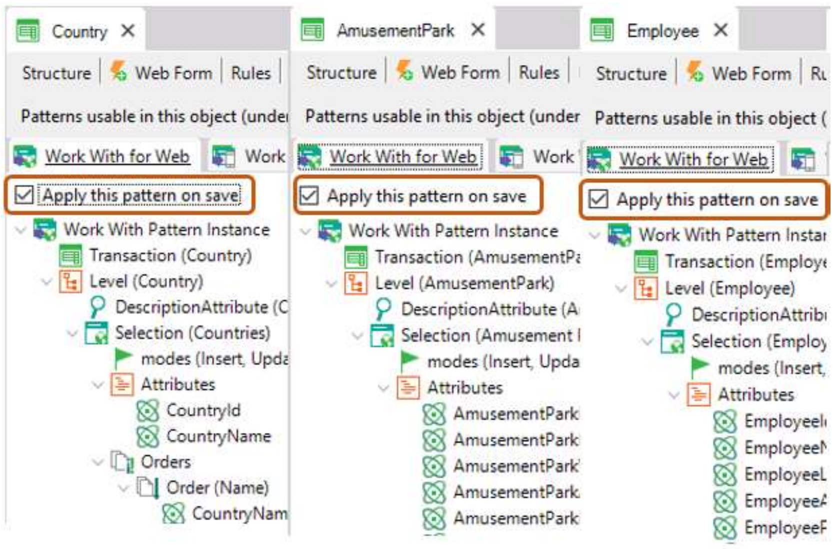
After seeing what has been done so far, the customer requests to handle information on countries, amusement parks, employees, categories and games in a more powerful and appealing manner (with the possibility of queries, filters and the insertion, modification and deletion of data, and so on).

To achieve this, the pattern *Work With for Web* should be applied to the three transactions. Try it and see it in runtime.

Note that:

- There is a Work With for Smart Devices as well. But the one you must apply is the one corresponding to the web application that you are building.
- GeneXus will automatically create several objects per transaction to implement the "Work with" that entity.

Solution:



Country ✕
Structure | Web Form | Rules |
Patterns usable in this object (unde
Work With for Web | Work
☑ Apply this pattern on save
Work With Pattern Instance
Transaction (Country)
Level (Country)
DescriptionAttribute (C
Selection (Countries)
modes (Insert, Upda
Attributes
CountryId
CountryName
Orders
Order (Name)
CountryNam

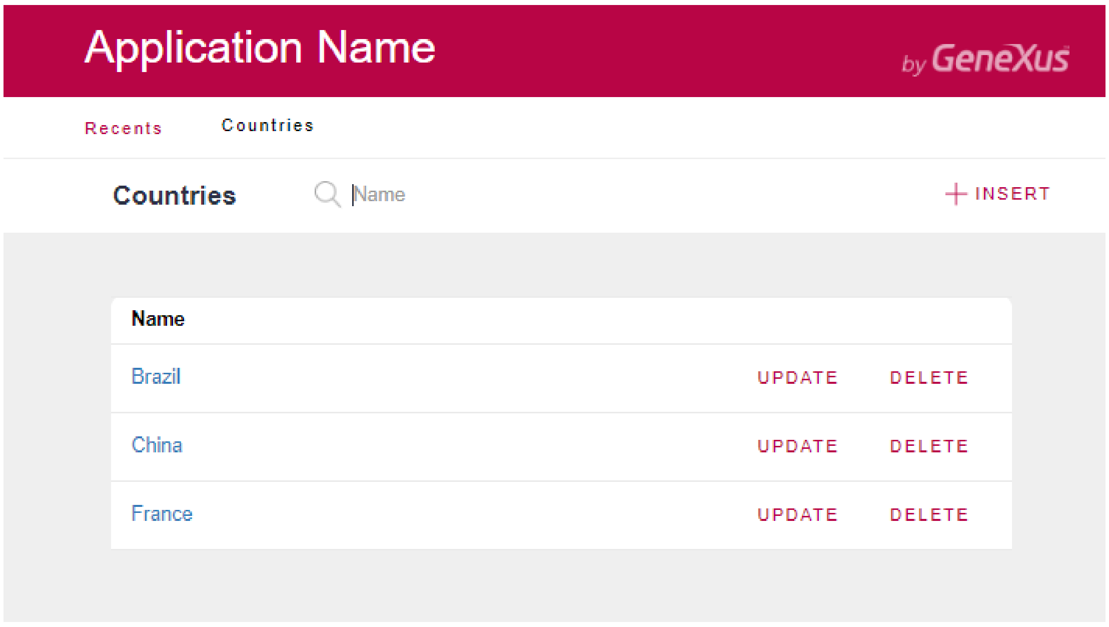
AmusementPark ✕
Structure | Web Form | Rules |
Patterns usable in this object (under
Work With for Web | Work
☑ Apply this pattern on save
Work With Pattern Instance
Transaction (AmusementPa
Level (AmusementPark)
DescriptionAttribute (A
Selection (Amusement I
modes (Insert, Upda
Attributes
AmusementPark
AmusementPark
AmusementPark
AmusementPark
AmusementPark

Employee ✕
Structure | Web Form | Ru
Patterns usable in this object (
Work With for Web |
☑ Apply this pattern on save
Work With Pattern Instar
Transaction (Employe
Level (Employee)
DescriptionAttribi
Selection (Employ
modes (Insert,
Attributes
Employeel
EmployeeN
EmployeeL
EmployeeA
EmployeeF

Why do the *Country, AmusementPark* and *Employee* transactions don't appear anymore in the *Developer Menu?*

Try the following:

1. Enter a new country.
2. Modify an existing country (for example, by adding a city to it).
3. Delete an existing country.
4. View the information relative to a country.
5. Do a search by country name.


Application Name by GeneXus

Recents   Countries

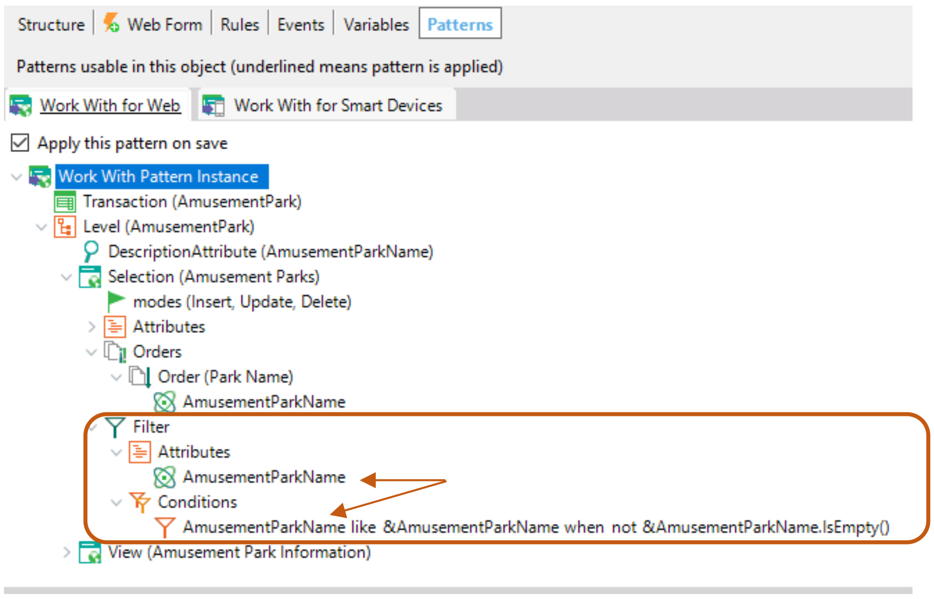
Countries   Name   + INSERT

Name		
Brazil	UPDATE	DELETE
China	UPDATE	DELETE
France	UPDATE	DELETE

6. Enter a couple of amusement parks (Ex: Shangai Disney Resort, of China/Shanghai, Parc Du Bocasse of France/Normandie, Happy Valley, of China/Beijing).
7. Filter the amusement parks whose name starts by P. What if you now want to view all parks in China? This possibility is not included, so we must customize the *Work With* pattern of that transaction in order to add it. Do it in GeneXus and test in runtime.

Solution:

To do this, first note how the existing filter has been specified, by name of the amusement park:



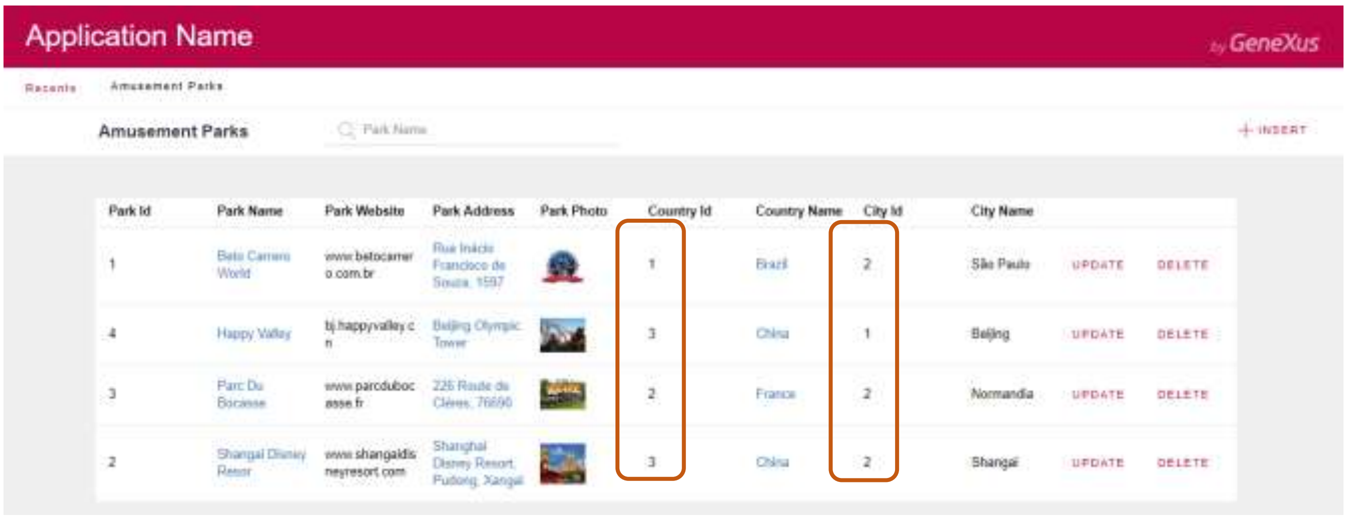
Patterns usable in this object (underlined means pattern is applied)

Work With for Web | Work With for Smart Devices

☑ Apply this pattern on save

```
Work With Pattern Instance
  Transaction (AmusementPark)
  Level (AmusementPark)
    DescriptionAttribute (AmusementParkName)
    Selection (Amusement Parks)
      modes (Insert, Update, Delete)
      Attributes
      Orders
        Order (Park Name)
          AmusementParkName
      Filter
        Attributes
          AmusementParkName
        Conditions
          AmusementParkName like &AmusementParkName when not &AmusementParkName.IsEmpty()
    View (Amusement Park Information)
```

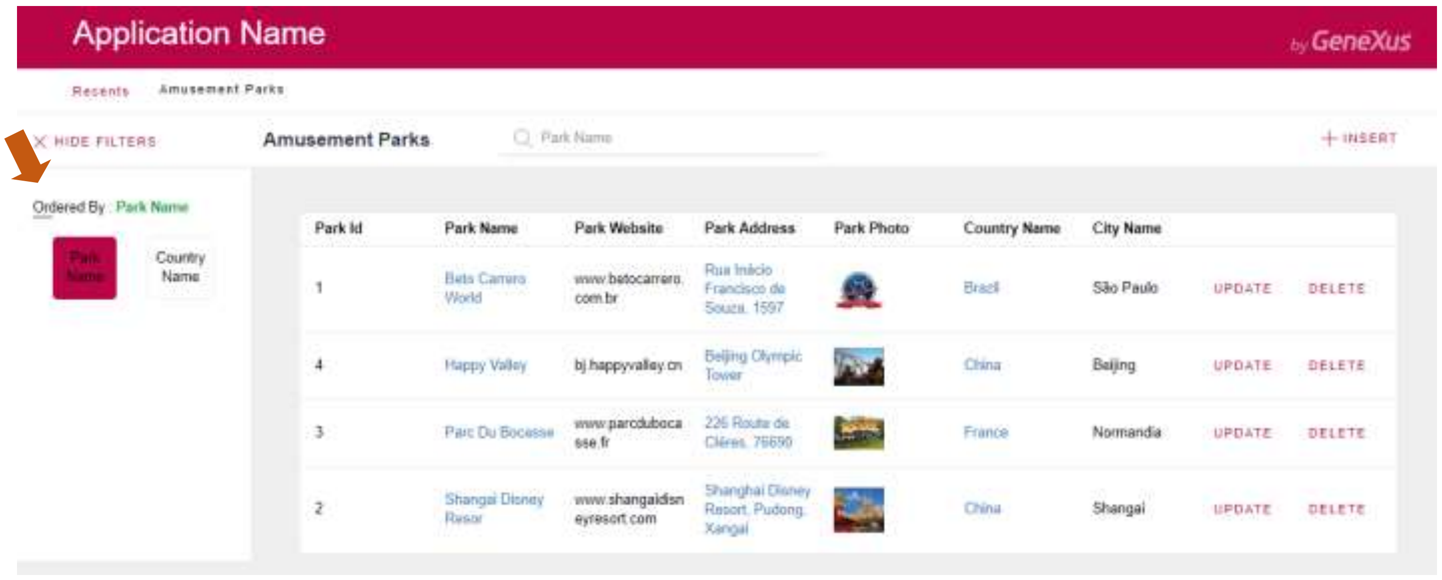
8. Now remove the country and city identifiers from the screen of the *Work With* and test it in runtime.



Amusement Parks

Park Id	Park Name	Park Website	Park Address	Park Photo	Country Id	Country Name	City Id	City Name		
1	Beto Carrero World	www.betocarrero.com.br	Rua Inácio Francisco de Souza, 1597		1	Brazil	2	São Paulo	UPDATE	DELETE
4	Happy Valley	bj.happyvalley.cn	Beijing Olympic Tower		3	China	1	Beijing	UPDATE	DELETE
3	Parc Du Bocasse	www.parcdubocasse.fr	226 Route de Clères, 76690		2	France	2	Normandia	UPDATE	DELETE
2	Shangai Disney Resor	www.shangaidisneyresort.com	Shanghai Disney Resort, Pudong, Xangai		3	China	2	Shangai	UPDATE	DELETE

9. If you now want to offer the possibility for the user to decide whether he/she wants to view the amusement parks sorted by park name or by country name, **implement it** and **test it**.



Amusement Parks

Ordered By : Park Name

Park Id	Park Name	Park Website	Park Address	Park Photo	Country Name	City Name		
1	Beto Carrero World	www.betocarrero.com.br	Rua Inácio Francisco de Souza, 1597		Brazil	São Paulo	UPDATE	DELETE
4	Happy Valley	bj.happyvalley.cn	Beijing Olympic Tower		China	Beijing	UPDATE	DELETE
3	Parc Du Bocasse	www.parcdubocasse.fr	226 Route de Clères, 76690		France	Normandia	UPDATE	DELETE
2	Shangai Disney Resor	www.shangaidisneyresort.com	Shanghai Disney Resort, Pudong, Xangai		China	Shangai	UPDATE	DELETE

“REPAIR” AND “TECHNICIAN” TRANSACTIONS, AND THE NEED TO DEFINE SUBTYPES

Now there is the need to record the games that go into repairs. Each repair has an identifier, a date as of which the game is out of order, the estimated number of days for the repair, the game identifier, its name, the main technician and the substitute technician. Every repair also has a cost. For the cost, create a domain called *Cost* of the type Numeric(8.2).

Create a transaction to record the technicians that will work on the repairs. Each technician has an identifier, a name and a surname, a telephone, a country and a city.

How do we define that each repair has a main technician and a substitute technician?

Remember

1) That, in the structure of the transaction:

- an icon with an upward arrow ↗ informs that the attribute is foreign key, that is, it points at another table.
- an icon with a downward arrow ↙ informs that the attribute is inferred from another table.
- an icon with an S indicates that the attribute is a subtype.

2) About the groups of subtypes:

- They are defined in the same way as any type of object.
- Each group of subtypes must necessarily contain a subtype of a primary attribute (that is primary key of a table) or a set of attributes that comprise a primary key.
- In each group of subtypes we must include all the attribute subtypes to be known and belonging to the base table and/or extended table of the group’s primary key.

Execute and **verify** that, upon trying to enter a repair, an error will be triggered when the main technician that is to be assigned to the repair does not exist. And the same goes for the substitute technician.

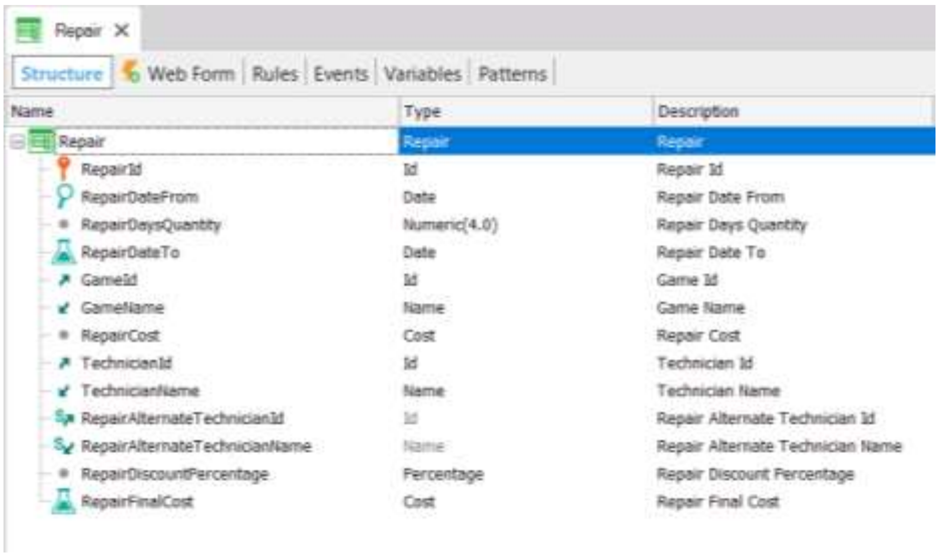
The entry of a repair whose main technician is the same as the substitute technician should not be allowed. Implement that behavior and test it in runtime.

FORMULAS

It should be possible to record the **current discount of each repair.** Define a new attribute in the *Repair* transaction **to save that data.** Name the new attribute as: *RepairDiscountPercentage* and make its data type a *Percentage* domain that is numeric and of length 3.

They require to view the final price of the repair, with the discount applied. To solve this, define another attribute called *RepairFinalCost* that is **global formula** and calculates the repair's final price automatically.

Add a new field called *RepairDateTo*, which will be an addition of the start date of the repair and the number of days that the repair will imply.



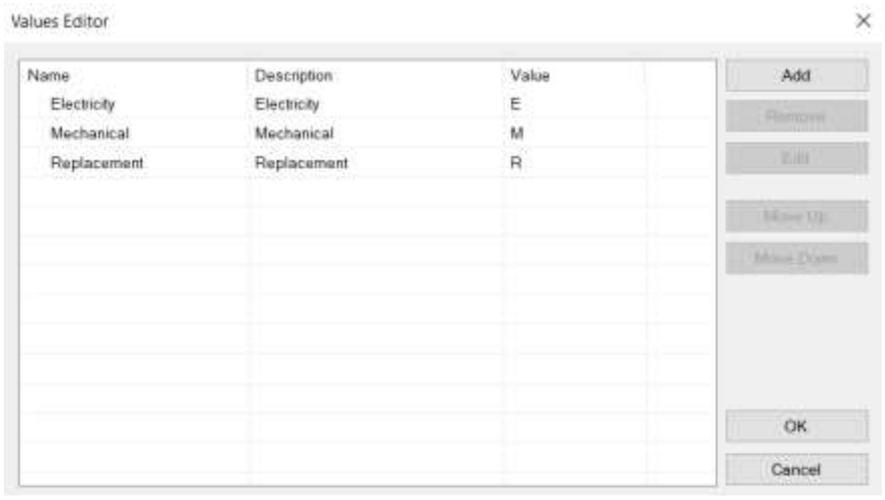
Name	Type	Description
Repair	Repair	Repair
RepairId	Id	Repair Id
RepairDateFrom	Date	Repair Date From
RepairDaysQuantity	Numeric(4.0)	Repair Days Quantity
RepairDateTo	Date	Repair Date To
GameId	Id	Game Id
GameName	Name	Game Name
RepairCost	Cost	Repair Cost
TechnicianId	Id	Technician Id
TechnicianName	Name	Technician Name
RepairAlternateTechnicianId	Id	Repair Alternate Technician Id
RepairAlternateTechnicianName	Name	Repair Alternate Technician Name
RepairDiscountPercentage	Percentage	Repair Discount Percentage
RepairFinalCost	Cost	Repair Final Cost

Press F5, **note, in the Impact Analysis, which attribute is created physically and which attribute is not,** and reorganize and test the application's functioning.

CREATION OF SECOND LEVEL

A second level is to be created in the *Repair* transaction to store details on the type of problems encountered that must be repaired.

To that end, the first thing to do is create a domain called *KindName*, Character(1). Limit the possible values for the domain: so that values "E","M" and "R" are valid (editing the **Enum Values** property as shown below).



Name	Description	Value
Electricity	Electricity	E
Mechanical	Mechanical	M
Replacement	Replacement	R

Create a second level in the *Repair* transaction called *Kind* for recording the type of repair. This level will have the following three attributes:

- *RepairKindId* – Numeric(4) (it will be key in this second level)
- *RepairKindName* – Based on the *KindName* domain (Genexus will suggest it automatically).
- *RepairKindRemarks* – Character(120), containing remarks on the problem, and a brief detail on the problem encountered, or the piece to be replaced.

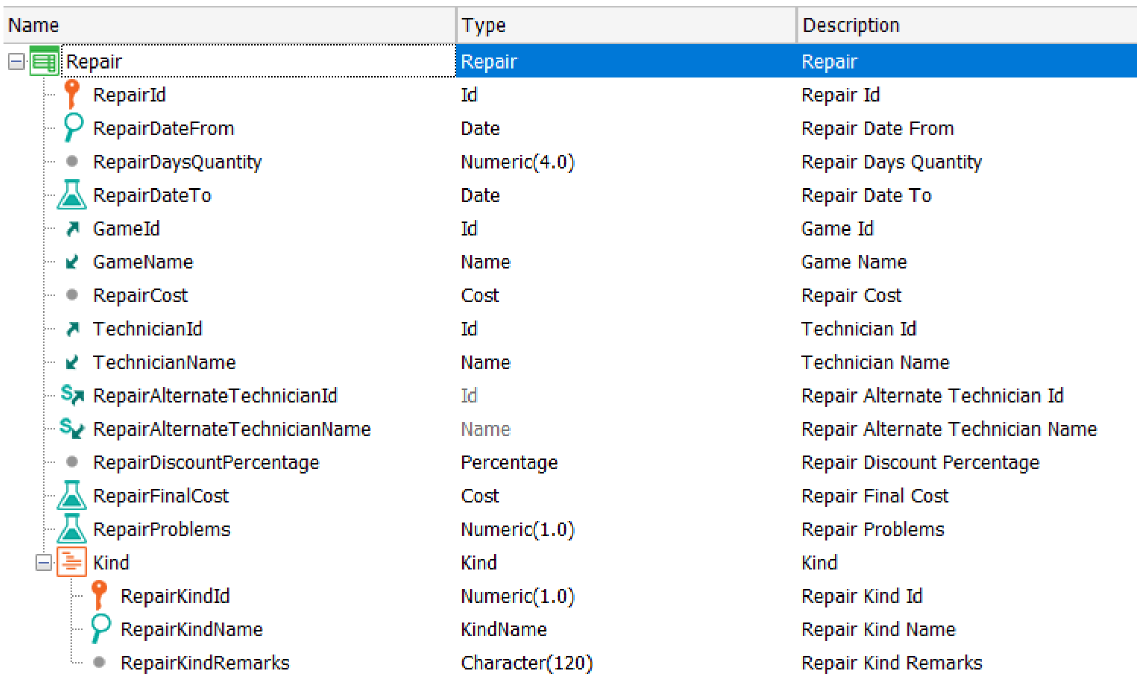
Remember that in order to define that a specific attribute is part of the primary key, you must right click on the attribute, and the contextual menu will show you the **Toggle Key** option. In this case, it will not be necessary because only the first attribute is part of the primary key and it already appears with the key icon.

In order to know the number of problem types implied in a repair, create a new attribute in the first level of the *Repair* transaction called *RepairProblems*, Numeric(1) and define it as global formula, which will have to count the types of problems encountered.

A repair could imply electricity problems, mechanical problems, or the need to replace a spare part. There could also be more than one problem of the same type, such as two electrical records, for instance. In the case of many, further detail may be provided using the remarks attribute that enables the possibility of writing a small text.

It is necessary to view the number of problems on the web form and to control the entry of 1 to 3 lines of problem types.

This control must be done **when data entry in the second level is finished** and after pressing the *Confirm* button.



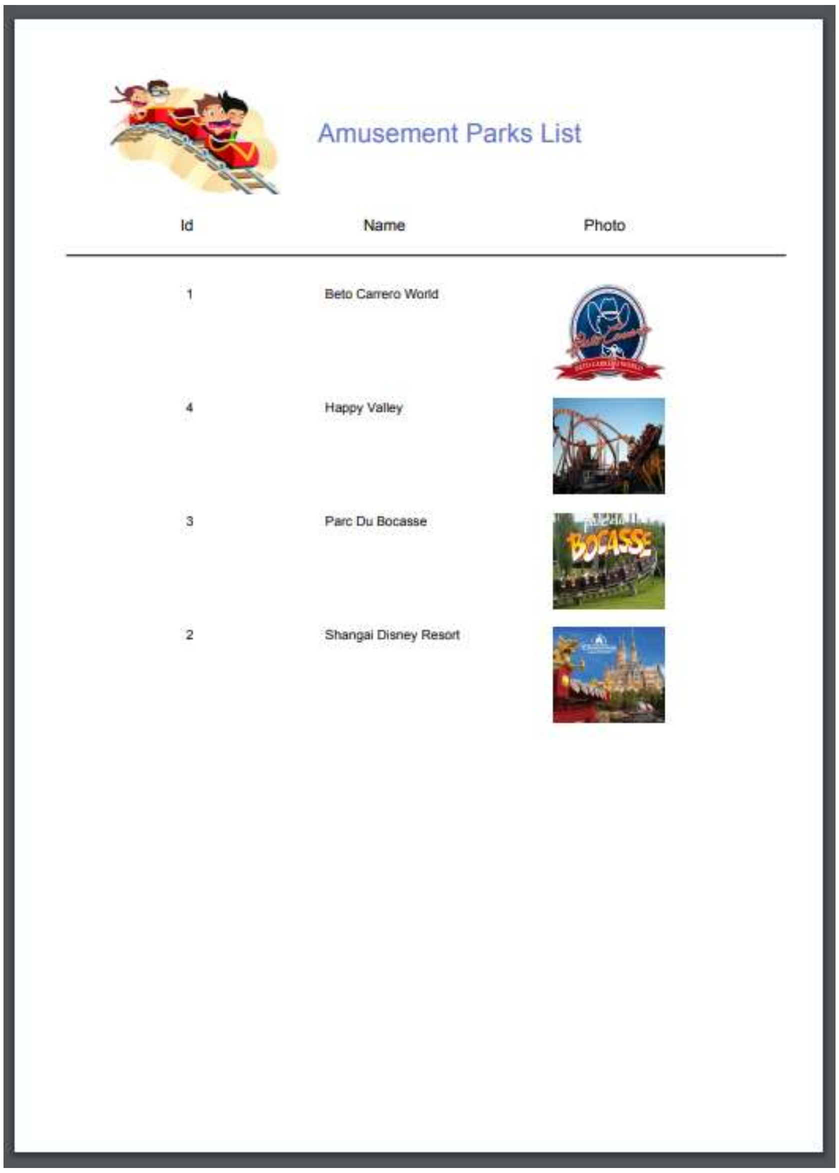
Name	Type	Description
Repair	Repair	Repair
RepairId	Id	Repair Id
RepairDateFrom	Date	Repair Date From
RepairDaysQuantity	Numeric(4.0)	Repair Days Quantity
RepairDateTo	Date	Repair Date To
GameId	Id	Game Id
GameName	Name	Game Name
RepairCost	Cost	Repair Cost
TechnicianId	Id	Technician Id
TechnicianName	Name	Technician Name
RepairAlternateTechnicianId	Id	Repair Alternate Technician Id
RepairAlternateTechnicianName	Name	Repair Alternate Technician Name
RepairDiscountPercentage	Percentage	Repair Discount Percentage
RepairFinalCost	Cost	Repair Final Cost
RepairProblems	Numeric(1.0)	Repair Problems
Kind	Kind	Kind
RepairKindId	Numeric(1.0)	Repair Kind Id
RepairKindName	KindName	Repair Kind Name
RepairKindRemarks	Character(120)	Repair Kind Remarks

The value of *RepairKindId* is entered manually. The normal thing would be to assign values to it starting with 1. Remember that **it is not possible to autonumber** using the *Autonumber* property.

PDF LISTINGS

Now let's suppose that, as part of the application, we must implement the possibility for deploying PDF Listings with the information required upon the user's request. For example, suppose that a listing is required to show amusement parks stored in the database, in alphabetical order.

We know that it should look approximately as shown below:

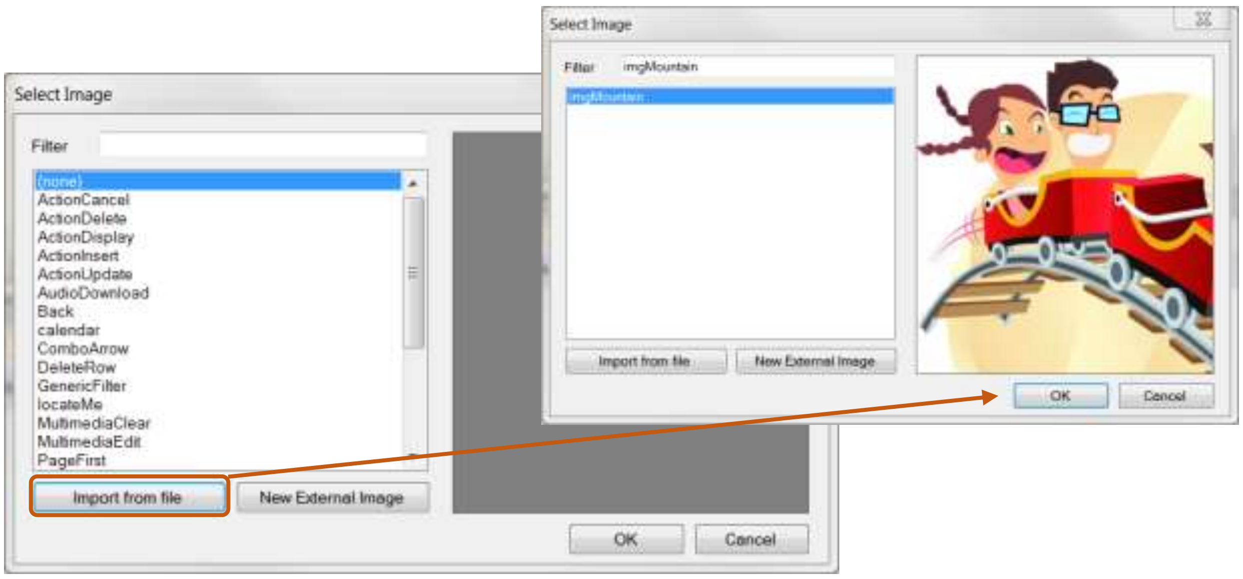


Amusement Parks List

Id	Name	Photo
1	Beto Carrero World	
4	Happy Valley	
3	Parc Du Bocasse	
2	Shangai Disney Resort	

A suggestion to place an image in the title is to use the *Image* control from the *Toolbox* for deploying an image next to the listing's title.

That image must be integrated with the knowledge base. To do that, select the *Import from File* option and search for the image to assign a name to it.



Implement it in GeneXus.

Remember that, to view a listing directly from the browser, it must be generated as PDF. To this end, you must configure the following properties of the procedure object:

- Main program = 'True'
- Call Protocol = HTTP
- Report output = Only to File

And the following rule:

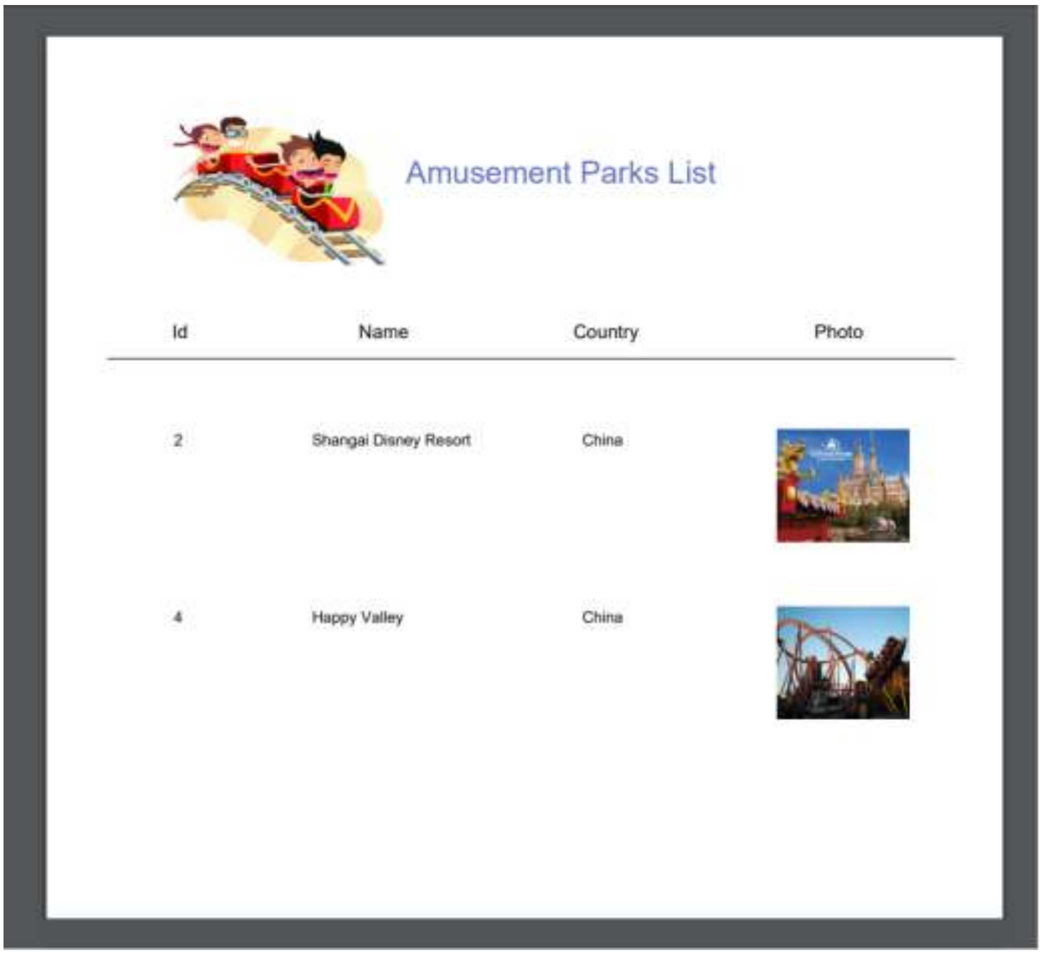
- Output_file('name-archivo.pdf' , 'PDF')

To execute the listing, right click on the object's tab / **Run with this only**

Did you notice what the procedure's navigation listing is informing you?

What if now the listing is needed to be ordered by country name? Implement it and note what is informed on the navigation listing and test it.

And if the requirement now is to only list the amusement parks in China? Test it (and note the navigation listing).

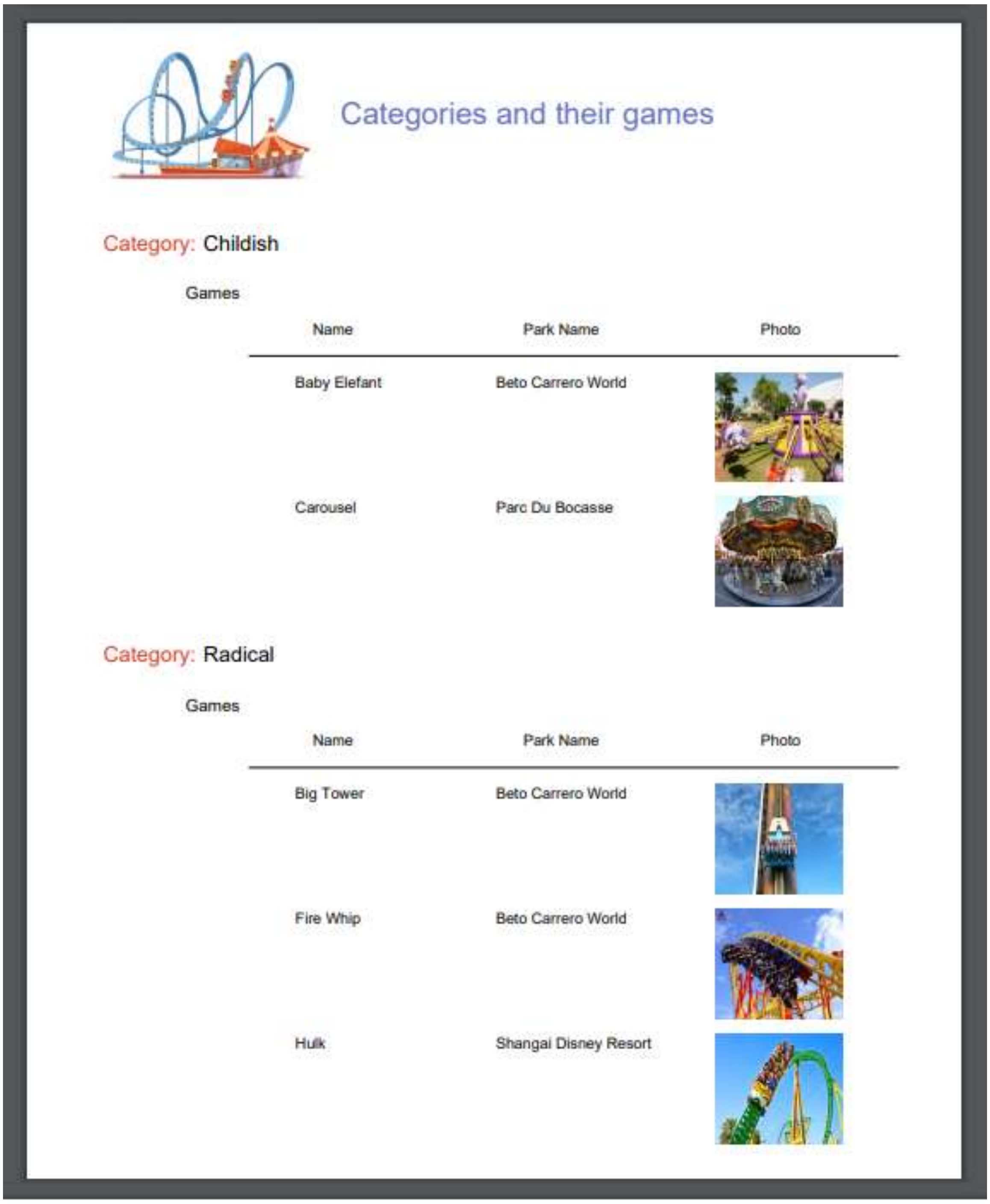


Amusement Parks List

Id	Name	Country	Photo
2	Shangai Disney Resort	China	
4	Happy Valley	China	

In each case, find out which table of the database is being run over to do the query of the *for each*.

A listing like the one below is also required (to show each category and for each of them, the games included). Implement it and test what is done.



Categories and their games

Category: Childish

Games

Name	Park Name	Photo
Baby Elefant	Beto Carrero World	
Carousel	Parc Du Bocasse	

Category: Radical

Games

Name	Park Name	Photo
Big Tower	Beto Carrero World	
Fire Whip	Beto Carrero World	
Hulk	Shangai Disney Resort	

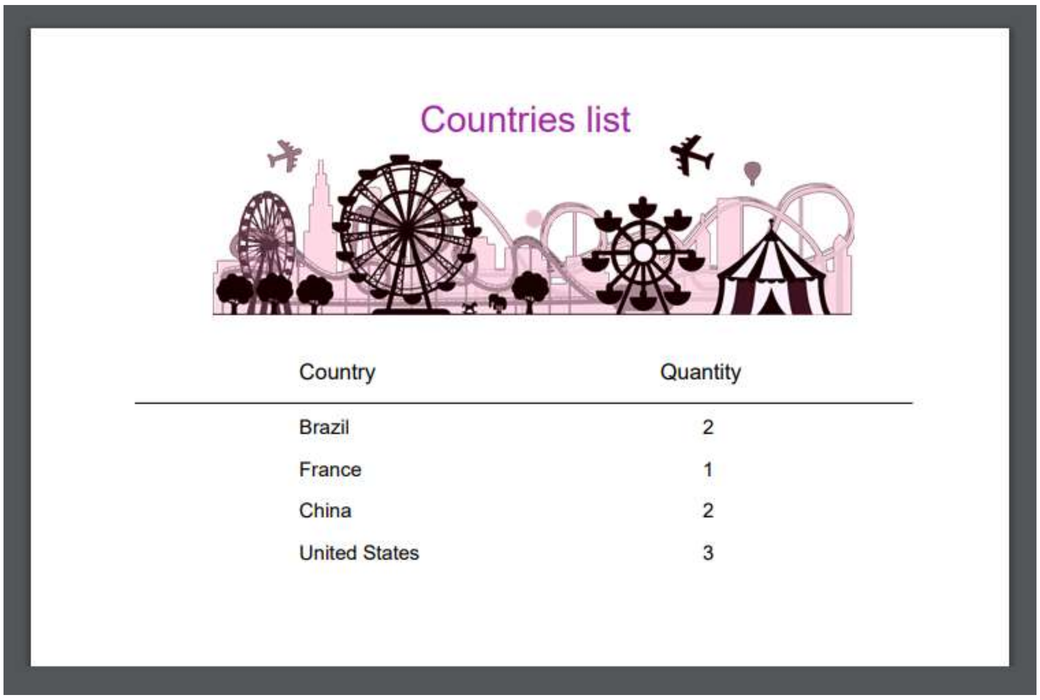
Add a new category to the system, such as *Aquatic*. Execute the previous listing again. Was the category listed now?

Modify the previous listing so that categories without related games are not listed.

What changes did you find in the navigation listing?

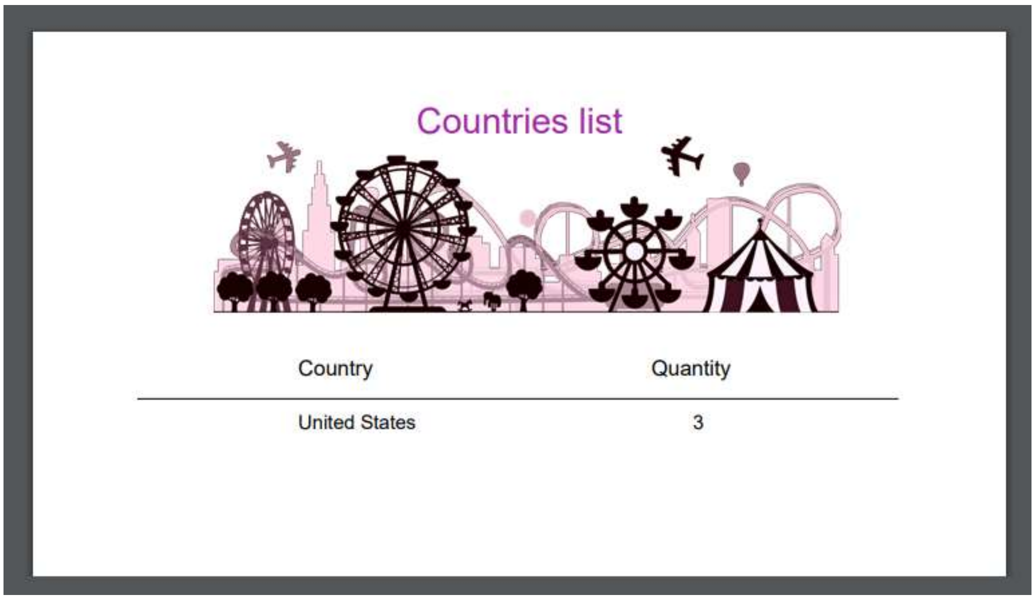
Answer: the word ***break***, appearing to indicate that a control break is taking place, showing also that the base table of the external *for each* is the same as the table of the nested *for each*.

Another requirement from the company is a listing to show all country names, and for each country the number of amusement parks offered:


Countries list

Country	Quantity
Brazil	2
France	1
China	2
United States	3

They also requested another listing to show all countries with **more than 2 parks available to visitors**:



Country	Quantity
United States	3

PASSING PARAMETERS

We often have the need for an object to receive parameters so that, based on them, it will execute its logic. For example, one thing is to create a PDF listing of all amusement parks in the database, and a different thing is to create a listing of those parks whose name is part of a given range.

LISTING OF PARKS IN A GIVEN RANGE

Save, with a different name, the procedure created previously to list amusement parks and modify it so that now it only lists the parks whose name is within a range received by parameter (the initial value and the end value of the range will be the parameters received).

Implement a screen to request, from the user, the values of that range, invoking this object and passing those values by parameter. Test in runtime.

Remember that:

- If you define a variable **based on** an attribute, it will be linked to the attribute's data type, that is, it this is modified the variable's will also be modified accordingly.
- The variables used for an invocation in the object called and those used in declaring the parameters received in the object called do not necessarily have matching names, but they do have to have compatible data types.

Solution:

To the procedure (let's call it *AmusementParksFromTo*), we will add the *parm* rule (and we define both variables in the procedure):

```
parm(in: &fromName, in: &toName);
```

And in the Source:

```
print Title
for each AmusementPark
  where AmusementParkName >= &fromName
  where AmusementParkName <= &toName
        print Data
endfor
```

We then create a web panel, where we define two variables, whose names may be any, such as for example &A and &B, but that require a data type compatible with that of the &fromName and &toName variables. Why? Because it will be in those variables that the user will enter values which in the event we program will be sent by parameter to the procedure. Therefore:

AmusementParksFromTo(&A, &B)

BUSINESS COMPONENTS

We will perform some operations on the database by means of *Business Components.*

PRICE INCREASE ON REPAIRS

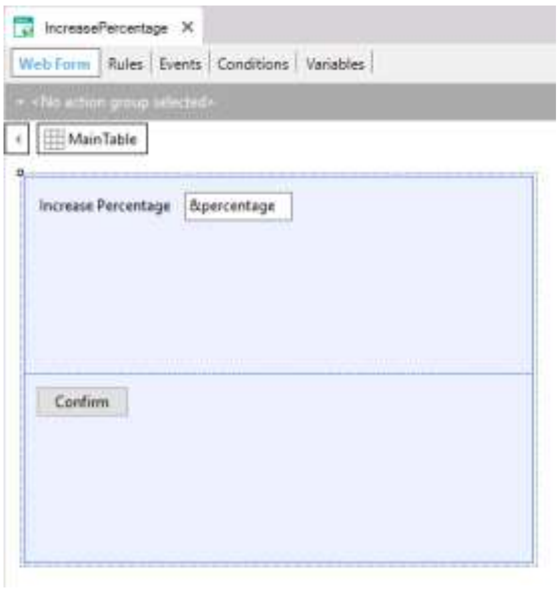
The company reserves its right to increase the prices of repairs by a given percentage as from a specific time. In order to perform a massive record, we will test this functionality. To do so we must implement a **screen** that will allow the user to specify that **increase percentage**, and impart the **order** for it to be applied to all repairs in the database. **Implement** it and **test** it.

Remember that:

- The web panel object allows the implementation of flexible screens for data input and output.
- To enter information (meaning that the user is allowed to enter values into the screen) from the *Toolbar* you may enter an Attribute/*Variable* control in the form and assign a variable to it.
- To edit menu bars, go up in GeneXus, on the bar, and right click to insert, for example, the *Formatting* bar.
- For the web panel to take a specific action, you may insert buttons and program the associated "event".
- *Business Components* are data types created when we set up the property called *Business Component* of the transaction object with value *Yes*. When we do that, to insert, modify or delete records from the corresponding tables, in addition to the transaction we may also use a variable of the *Business Component* data type in any other object (for example: a web panel) and do the same operations by means of the *Load(), Save()* and *Delete()* methods.
- For the operations done with the *Business Component* to be established as permanent, you must execute the *Commit* command immediately after.
- When we wish to increment a value (X) by 20%, all we need to do is X = X*(1+20/100) = X*1,20

Solution:

One possible solution (try to implement the requirement but before you read what follows here below) is to create a web panel, with a &percentage variable of the Numeric(3.0) data type.



When we double click on the button we go to the Events section, editing the event associated with the moment of defining the button. In our example, it is the Confirm event.

There, we will program the logic that we want executed when the user presses that button.

We must go over all the repairs and overwrite the price they had previously, increasing them by the percentage indicated by the user in the variable on screen.

In order to go over all the repairs in the database we use the For each command.

And for each repair, how do we edit the value to modify RepairCost?

We will need a variable for that, with the whole structure of the Repair transaction, and it must also allow us to save in the database. What data type will that variable have then? The Business Component Repair (note that the Business Component data type has the same name as the transaction). But GeneXus does not create that data type automatically for each transaction. We have to request that. How? With the corresponding property of the transaction.

So, once we have done that:

```
Event 'Confirm'
    For each Repair
        &Repair.Load(RepairId)
        &Repair.RepairCost = &Repair.RepairCost * (1 + &percentage/100)
        &Repair.Save()
    Endfor
    commit
Endevent
```

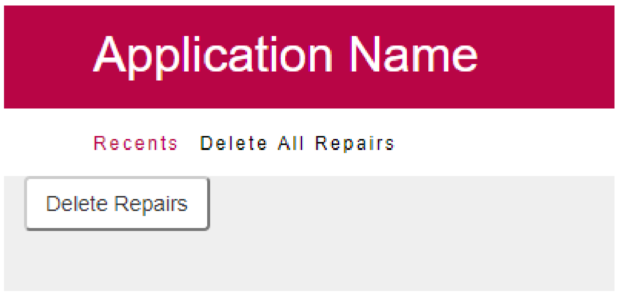
Since the recording or deleting operations for the database (by means of the Save() and Delete() methods) may cause errors, it is important to know exactly what happened. To know that, we have the Success() method that will return True when there have been no errors, and False otherwise.

```
Event 'Confirm'
    For each Repair
        &Repair.Load(RepairId)
        &Repair.RepairCost = &Repair.RepairCost * (1 + &percentage/100)
        &Repair.Save()
        If &Repair.Success()
            commit
        else
            rollback
        endif
    Endfor
Endevent
```

When we insert, modify or delete records from a table in the database, as long as it is not specified that: “everything done remains as permanent”, such modifications will be temporary. What does that mean? That, for example, if there is a blackout, those changes will be lost. The way to establish that “everything done remains as permanent” is to execute the Commit command. If, otherwise, we want everything we did to be undone, then we execute the Rollback command.

SCREEN FOR DELETING ALL REPAIRS

Save the previous web panel with another name (all you need to do this is go to the tab and right click, then *Save as*) and modify the Form so that it only contains a button with the text *Delete Repairs*, and then in runtime, this web panel will look as follows:



As the user presses the button, all repairs should be deleted from the database.

What must modify the *Confirm* event you had programmed?

Note: By going to the button to see the properties, in the property called *Caption* you may modify the button’s text.

Solution:

```
Event 'Confirm'
    For each Repair
        &Repair.Load(RepairId)
        &Repair.Delete()
        If &Repair.Success()
            commit
            msg("Successful deletion")
        else
            msg("Deletion failed")
            rollback
        endif
    Endfor
Endevent
```

With *Msg* you will be deploying that message on the web panel screen.

PROCEDURES TO UPDATE RECORDS

PRICE INCREASE ON REPAIRS

Suppose that there are thousands of repairs whose prices must be increased by a given percentage. Considering that the price increase is a simple procedure that will not cause integrity to fail in any way, practice solving this with a procedure but without using *Business Components*.

Remember that:

- In a procedure, you may update the extended table's attributes with the *For each* command by means of simple assignments.
- The "direct" update by means of procedures will not control data integrity.
- Every object must declare the parameters received and the parameters returned. Otherwise, it will neither receive nor return any values.
- Parameters are declared with the ***parm*** rule.
- Variables are local in relation to the object where they are used. This means that when we want to receive a value as parameter in &X variable, we must declare it in the object.

DELETING ALL REPAIRS

And if we now need to delete all repairs as done before in the practice, but this time using a procedure?

Solution:

Create a *RepairsDeletion* procedure that will not receive parameters, with the following code:

```
for each Repair
   delete
endfor
```

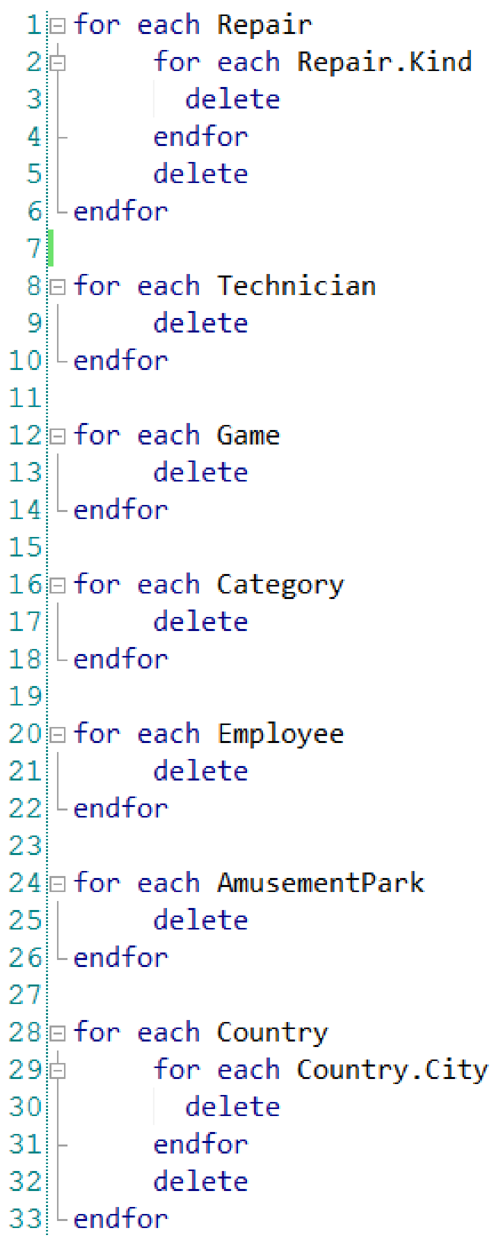
Do a *Save as* of the web panel you had implemented to do the deletion by means of *Business Component*), and change the *Enter* event:

```
Event Enter
    RepairsDeletion()
EndEvent
```

And if you now want to delete all data from the database?

Solution

Create a DeleteAll procedure with the following Source:



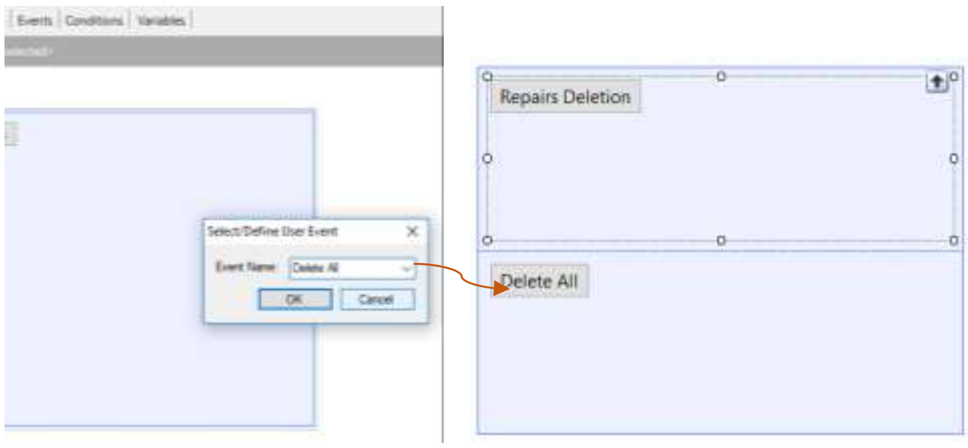
```
for each Repair
    for each Repair.Kind
        delete
    endfor
    delete
endfor

for each Technician
    delete
endfor

for each Game
    delete
endfor

for each Category
    delete
endfor

for each Employee
    delete
endfor

for each AmusementPark
    delete
endfor

for each Country
    for each Country.City
        delete
    endfor
    delete
endfor
```

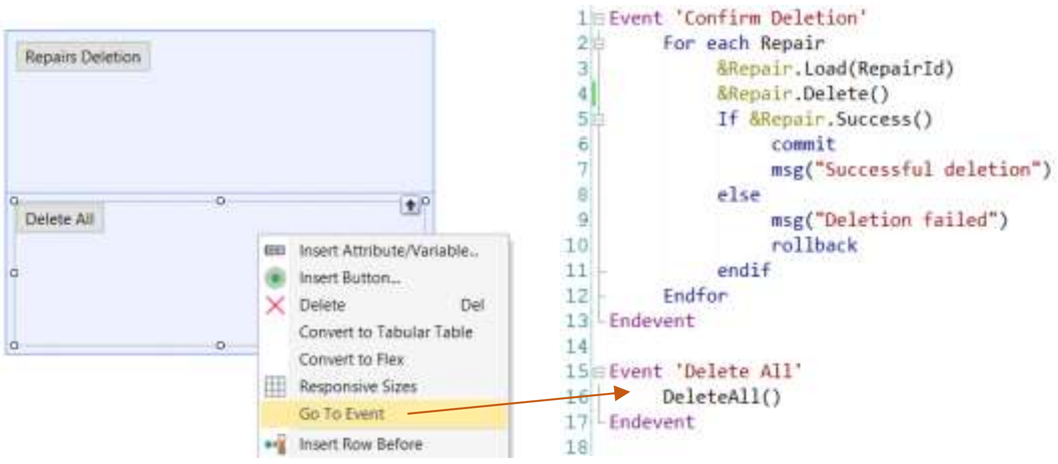
Remember that procedures do not validate data consistency and the database does. This means that the database validates the consistency of inter-related data, so the order in which you try to delete data is important.

For example, if you try to delete countries before you delete parks, the database will prevent you from doing that and the program will cancel, and it will no longer be user-friendly.

On the same web panel used to delete all the Repair records using Business Components, add a new button to which we will associated a "user event" and we will do the call to the procedure.

Drag the control button over the form and define the new event.





```
Event 'Confirm Deletion'
        For each Repair
                &Repair.Load(RepairId)
                &Repair.Delete()
                If &Repair.Success()
                        commit
                        msg("Successful deletion")
                else
                        msg("Deletion failed")
                        rollback
                endif
        Endfor
Endevent

Event 'Delete All'
        DeleteAll()
Endevent
```

Afterwards, right click on the button and select Go To Event. Once on the event associated with the button, define the procedure inside the invocation.

INITIALIZATION OF DATABASE INFORMATION [OPTIONAL]

When the app you are developing goes into production (that is, when it starts to be used by the company) all data relative to countries, parks, categories, games, technicians and repairs must be loaded. Use the facilities provided by the *Transaction* object for this purpose and initialize those tables with information we received from the company, and test it.

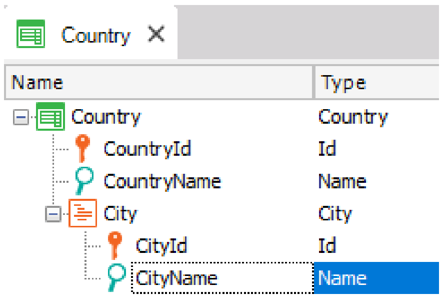
Bear in mind that:

- Transactions have a ***Data Provider*** property that enables us to have a data provider associated with the transaction, so when the table is created in the database, it will be initialized with the values we define in that data provider. To that end, we must also set up the ***Used to*** and ***Update Policy*** properties.
- Images must be inserted in the KB in order to be used. A way of doing this is: in the *KB Explorer* go to the *Customization* node and select *Images* where you will find a list of all images currently stored in the *KB*, and insert a new one (from a file) and give it a name.

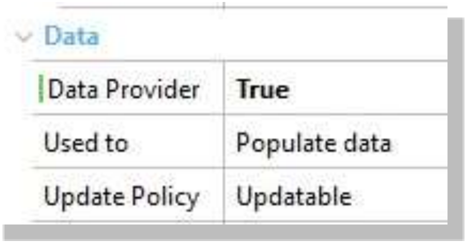
Solution

So as to try different options, we will test the countries and categories using the Data Provider property in the transactions.

- Open the Country transaction and set up the value True for the Data Provider property:



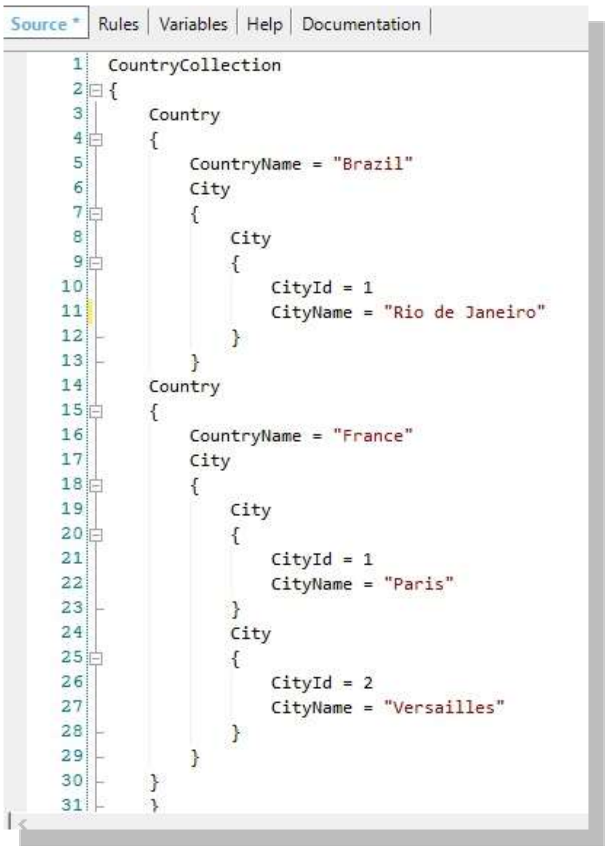
Name	Type
Country	Country
CountryId	Id
CountryName	Name
City	City
CityId	Id
CityName	Name



Data

Data Provider	True
Used to	Populate data
Update Policy	Updatable

- Load all the data desired in Country_DataProvider (you may find it under the Country transaction, in the KBExplorer window). When it is created, the table will be loaded with the data indicated.



```
CountryCollection
{
    Country
    {
        CountryName = "Brazil"
        City
        {
            City
            {
                CityId = 1
                CityName = "Rio de Janeiro"
            }
        }
    Country
    {
        CountryName = "France"
        City
        {
            City
            {
                CityId = 1
                CityName = "Paris"
            }
            City
            {
                CityId = 2
                CityName = "Versailles"
            }
        }
    }
}
```

Remember:
The *DP* associated with the transaction will be triggered again

- every time that the corresponding table is reorganized.
- every time that the content of the *DP* is edited.

Therefore, it is necessary to control that information is not duplicated. This is possible:

- when the primary key is autonumbered. A *unique* index may be defined on the descriptor attribute.
- by leaving the primary key without autonumbering, so that its value will be assigned in a fixed manner when the *DP* is defined, and it will not be duplicated.

- Load the categories in the same manner.

WEB PANELS

A page is required to show all countries, and for each of them the number of amusement parks available to visitors.

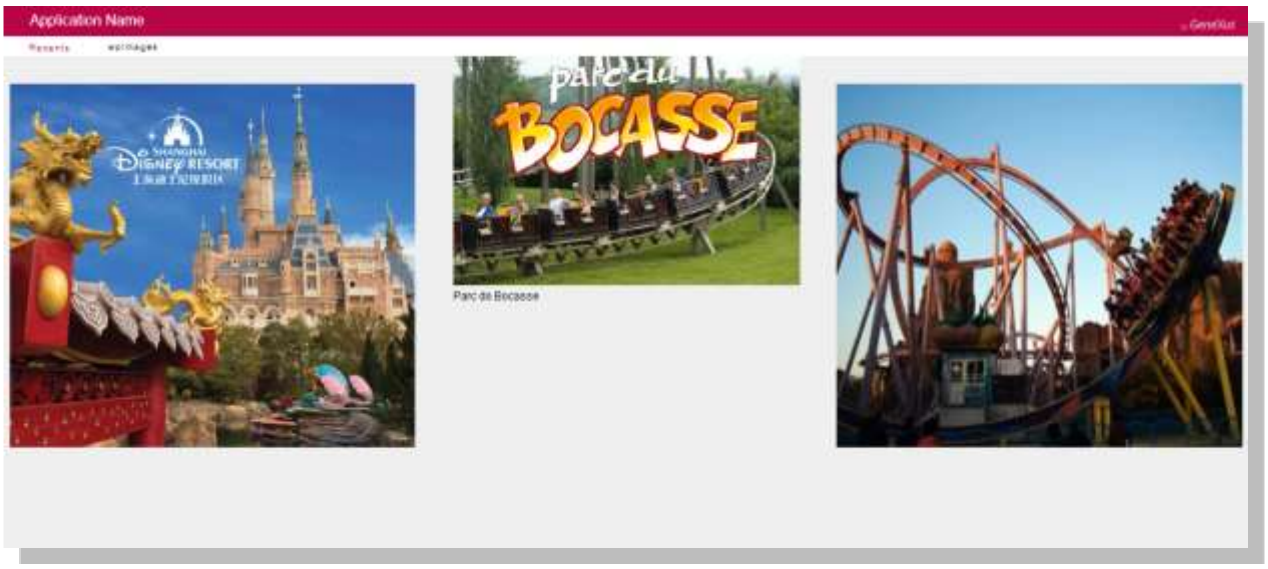
Remember that the ***Load*** **event in a web panel with base table with a grid** is executed just before each line is loaded on the grid. This event is adequate for assigning to the variable the calculation returned by the cities count of each country navigated and about to be loaded on a grid line.

Now add two variables (&*CountryNameFrom* and &*CountryNameTo*) to the panel defined and define the conditions necessary to filter the countries included in that range.

EXTENDED CONTROLS

Using the *ImageGallery* extended control, design a web panel to show the photo gallery of all amusement parks.

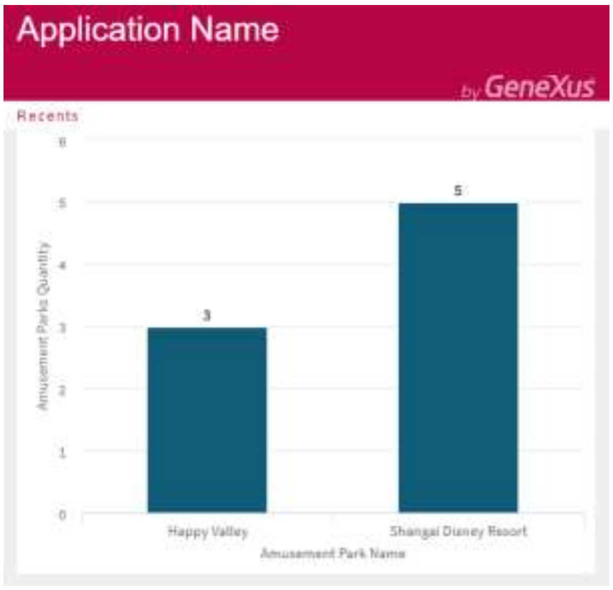
Customize the properties of the *extended control*, with the following set up: Width=1000, Height= 500, and Type=Slider:



QUERY OBJECT

Define a query object that will only return China's amusement parks in alphabetical order, each with its respective number of games.

Define a web panel and, view the previous query as a graph using the *QueryViewer* extended control.

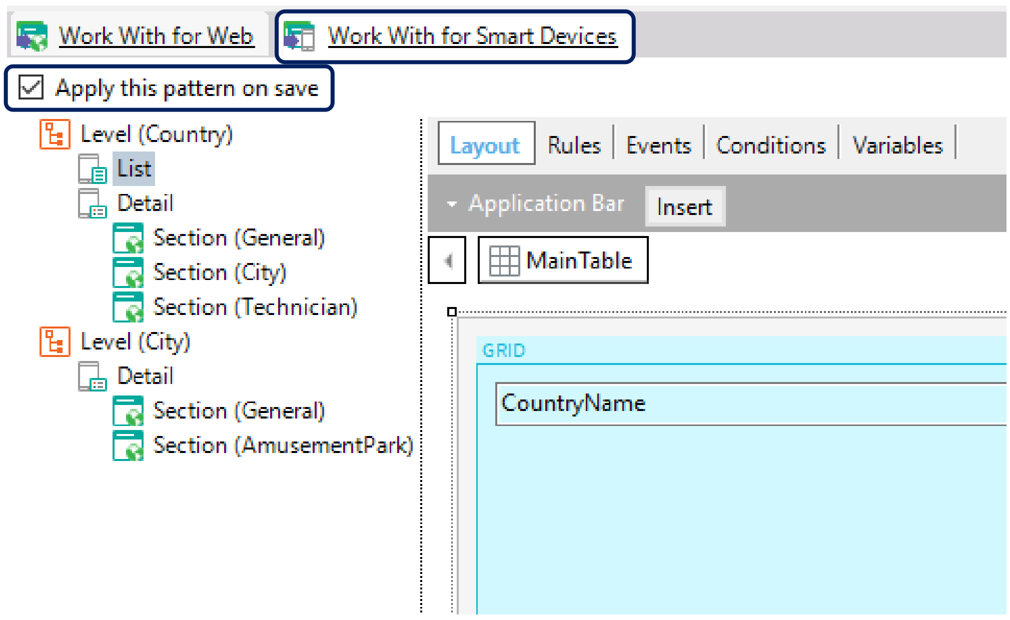


SMART DEVICES SECTION

The company also wants to offer customers a little app for smart devices to be used by end users.

The idea is to allow anyone to make queries from their smart devices regarding all the countries they may visit, and the amusement parks and games available in each country.

To do that, we must apply the *Work With for Smart Devices* pattern to the Country transaction:



And then, just simply save, to later create a *MenuforSmartDevicces* object and add to it the name object: *WorkWithDevicesCountry* generated by the pattern.

Remember that if you created objects for Smart Devices in the knowledge base, when you press F5, the emulator for *Android* will be automatically executed, with the possibility of accessing the web app from the *Developer menu* as well.

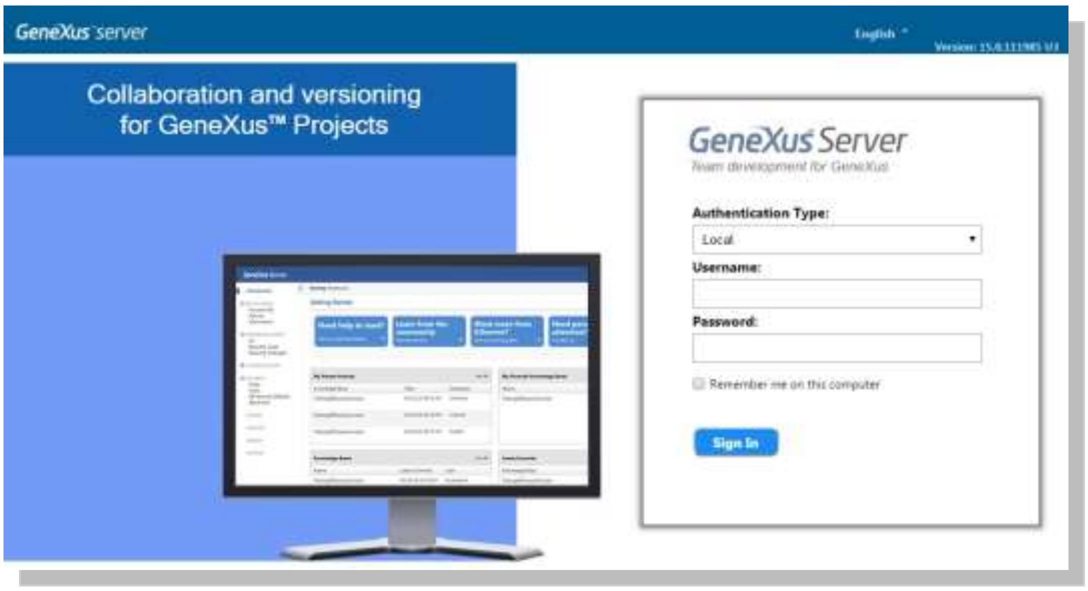
GENEXUS SERVER

Publish the knowledge base in the server http://sandbox.genexusserver.com/v16

Create a new web panel showing the list or registered amusement parks.

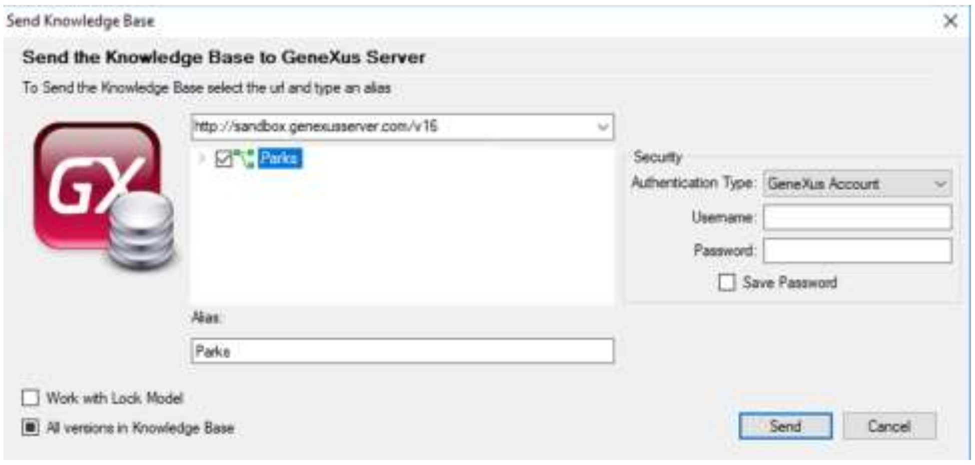
Send this new object to the server so as to integrate it with the centralized knowledge base.

Access the web console and verify the final status of the *KB*.


GeneXus server
English
Version: 15.0.111985 U3
Collaboration and versioning for GeneXus™ Projects
GeneXus Server
Team development for GeneXus
Sign In

Authentication Type: Local
Username:
Password:
Remember me on this computer

Close the previous knowledge base and create a new one synchronized with the previously published *KB*. This will enable you to locally receive a copy of the *KB* managed by GeneXus Server.



Send the Knowledge Base to GeneXus Server

To Send the Knowledge Base select the url and type an alias

http://sandbox.genexusserver.com/v16

Parks

Security
Authentication Type: GeneXus Account
Username:
Password:
Save Password

Alias: Parks

Work with Lock Model
All versions in Knowledge Base

In this new local copy, edit the Country transaction and define the new *CountryFlagImage* attribute, of the *Image* type. Send the change to the server.

Close this *KB* and open the initial *KB* again. Perform the *Update* operation to receive the change made before.