

Practical exercises

GeneXus Core Course

GeneXusTM 18

October 2022

© GeneXus. All rights reserved. This document may not be reproduced by any means without the express permission of GeneXusTM. The information contained herein is intended for personal use only.

Registered Trademarks:

GeneXus is trademark or registered trademark of GeneXus S.A. All other trademarks mentioned herein are the property of their respective owners.

CONTENTS

CONTENTS.....	2
PROBLEM.....	4
NEW PROJECT, NEW KNOWLEDGE BASE.....	4
INITIAL TRANSACTIONS.....	5
“Employee” transaction	5
“AmusementPARK” and “country” transactions, related.....	7
Related data: How is integrity maintained?.....	11
‘SHOW’ Transaction	12
‘Game’ Transaction.....	12
‘Category’ Transaction	12
“Employee” and “AmusementPark” transactions, related	14
ADDING cities to the ‘Country’ transaction	15
“AmusementPark” Transaction: ADDING the city.....	16
ADDING BEHAVIOR TO TRANSACTIONS (RULES)	17
PATTERNS: IMPROVING THE INTERFACE FOR WORKING WITH INFORMATION	17
“REPAIR” AND “TECHNICIAN” TRANSACTIONS AND THE NEED TO DEFINE SUBTYPES.....	21
FORMULAS	22
CREATING THE SECOND LEVEL	23
INDEXES.....	24

PDF LISTS 25

PASSING PARAMETERS..... 32

Listing of parks in a certain range 32

BUSINESS COMPONENTS 32

Price increase for repairs 32

Screen to delete all repairs..... 33

PROCEDURES TO UPDATE RECORDS..... 34

Price increase for repairs 34

Deleting all repairs 35

Initializing database information [optional]..... 36

WEB PANELS 38

Multiple Web Panels [OPTIONAL]..... 39

Types of Web Panels 40

DESIGN SYSTEMS 41

Paging in Grids 45

GENEXUS SERVER 47

PROBLEM

A multinational company that manages amusement parks hires you to develop a system to store and handle the information it works with. Suppose that the system is made up of two modules:

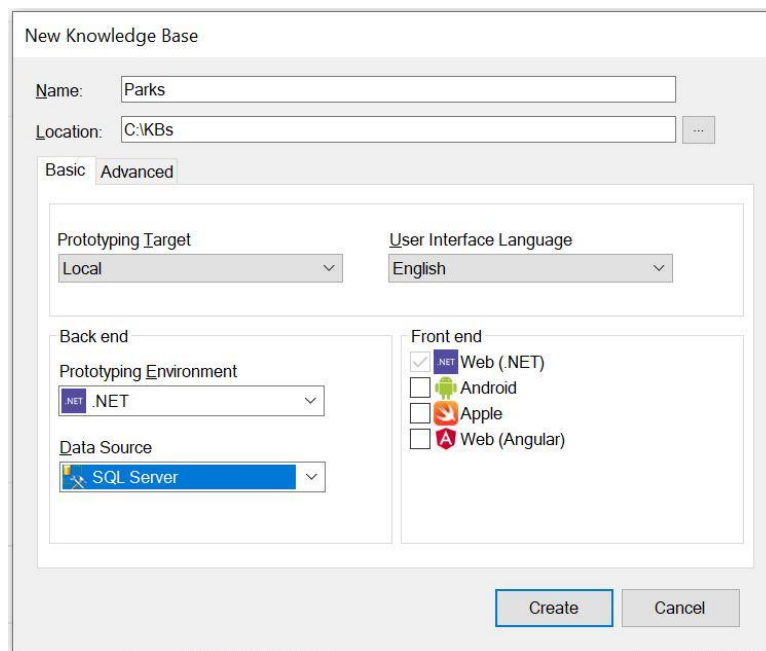
- **Backend:** part of the application that will run on a web server, so that the company's employees can handle the information from any location with an Internet connection.
- **Simple application for mobile devices:** part of the application that will be downloaded by the company's clients, and will allow them to search for the available countries, as well as the main amusement parks offered by each city and its attractions.

NEW PROJECT, NEW KNOWLEDGE BASE

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

Tips:

- Select .NET as a development environment. Make sure everything you need is installed (including SQL Server). If you are using GeneXus Trial, the generation environment with .NET and SQL Server is already predefined, prototyping in the Amazon cloud.
- Do not create the knowledge base in “My Documents” folder or any other folder under “Documents and Settings” because these folders have special permissions granted by Windows.



Take a few minutes to **get to know** the **IDE (Integrated Development Environment of GeneXus)**. Try **moving windows**, and **viewing specific windows you want** (View and View/Other Tool Windows), and take a close look at the contents of the **KBExplorer** window (Knowledge Base Explorer). You will see that **domains**, and some **objects**, **images**, etc. are already initialized.

Tip: keep the properties window open (**F4**) because you will use it very often.

INITIAL TRANSACTIONS

When you meet with the company's representatives, they explain the following:

“We record the data of amusement parks, to manage both their employees and their games and activities offered to visitors.”

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

“EMPLOYEE” TRANSACTION

We ask: What data do you record about the company's employees? The answer is as follows:

Their **name** (up to 20 characters), **last name** (up to 20 characters), **address**, **phone** and **email**.

With these details, you can already create the *Employee* transaction.

Remember that:

- To create objects, several options exist:
 - Do it from the menu: File/New/Object
 - Ctrl+N
 - Toolbar icon
- You will need an attribute that identifies each employee (EmployeeId).
- By typing a period (“.”) when you enter a new attribute, it is initialized with the name of the transaction.

The transaction structure should look as follows:

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** automatically assigned to the attributes that are defined with the words Address, Phone or Email in their name, respectively.
- When defining the data type of the identifier attribute, instead of using Numeric(4.0) directly, define the **Id domain** inline: **Id=Numeric(4.0)**. Set the **Autonumber** property of that domain to **True**, so that all attributes based on it are automatically numbered, with no need for the user to be concerned about it.

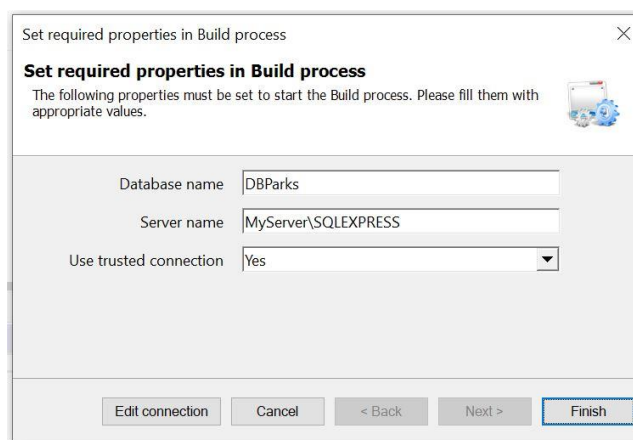
The next step is to test the application running. Make sure you have the GeneXus *Output* window enabled and visible. (*View / Other Tool Windows / Output*).

Now, **try** the application at runtime by pressing **F5**.

What will happen?

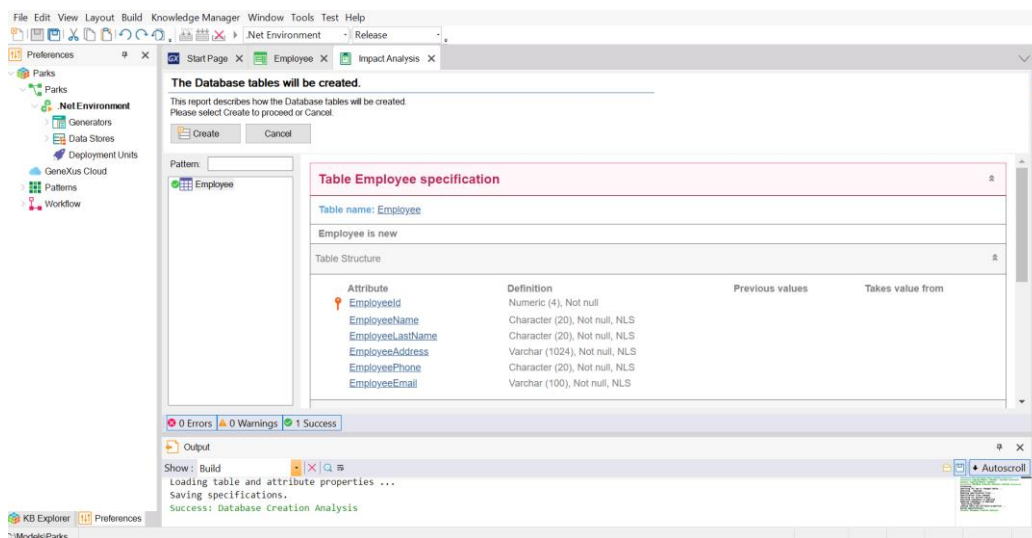
Remember that:

If you are using the *Full* version or the *Learning* version of GeneXus, and decide to create the database and programs **locally**, a window like the one below will open for you to enter the database, server and connection method information. Remember that if there is no database with the name you indicated on that server, GeneXus **will create** it.



But if the database and programs will be created in the cloud, the above dialog is not displayed because GeneXus knows the details of the cloud server and automatically sets the database name and all the database connection information.

An **Impact Analysis** is then displayed detailing that the database and the *EMPLOYEE* table will be created within it:



If you click on the **Create** button, GeneXus will run the program that will create it. At the end of the process, in the browser set as default, the menu with links to execute the defined objects will be opened. In this case, only one: the *Employee* transaction.

Enter some employees into the system. Next, **change** some data of an employee previously entered and **delete** an employee.

Also, try using the little arrows available to go from one record to another and the *SELECT* option, which offers a “selection list” to view the list of registered employees and select one.

Now let's identify and create the next transaction. Remember what we had been told, to which some additions were made:

“We record the data of **amusement parks** of different **cities** in different **countries**, to manage their **employees** as well as the **rides** and **activities** they offer to visitors.”

Before moving on with the development, let's publish the KB on GeneXus Server (<http://sandbox.genexusserver.com/v17>).

Remember that:

- You need to authenticate with your GeneXus account.
- The alias used to publish the KB must be unique. This means that there can't be two KBs with the same alias on the Server.

“AMUSEMENTPARK” AND “COUNTRY” TRANSACTIONS, RELATED

We ask the company's employees: what data do you record about the amusement parks you work with? The answer is as follows:

Their **name** (up to 40 characters), **web site** (up to 60 characters), **address** and **photo**.

With these details, you can already create the *AmusementPark* transaction.

Name	Type
AmusementPark	AmusementPark
AmusementParkId	Numeric(4,0)
AmusementParkName	Character(20)
AmusementParkWebsite	Character(60)
AmusementParkAddress	Address, GeneXus
AmusementParkPhoto	Image

We will create a transaction to record the countries to which the amusement parks belong.

Remember that:

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

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

Name	Type
Country	Country
CountryId	Id
CountryName	Name

Now, return to the *AmusementPark* transaction to add the *CountryId* and *CountryName* attributes.

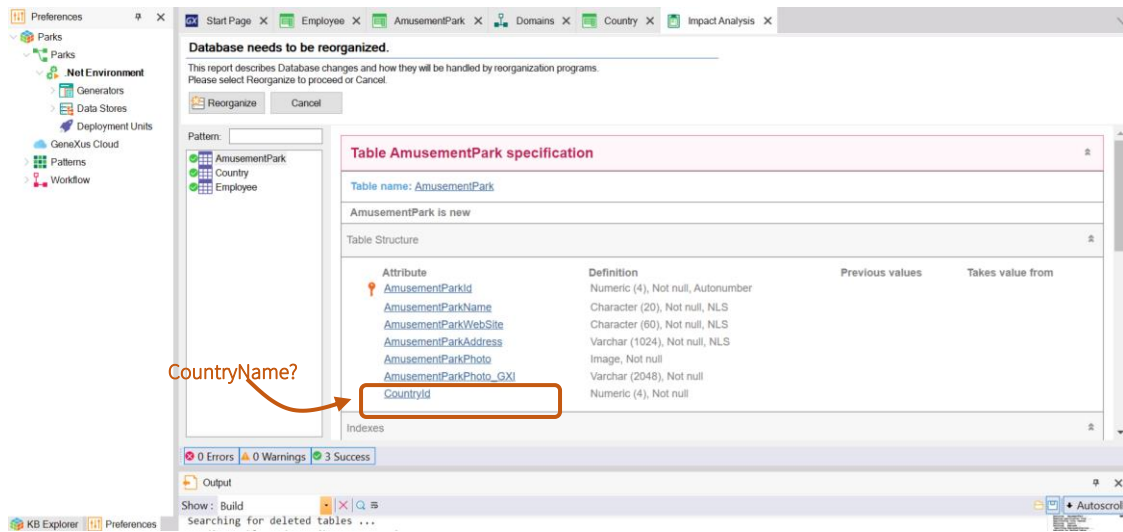
Name	Type
AmusementPark	AmusementPark
AmusementParkId	Id
AmusementParkName	Character(20)
AmusementParkWebsite	Character(60)
AmusementParkAddress	Address, GeneXus
AmusementParkPhoto	Image
CountryId	Id
CountryName	Name

Name	Type
Country	Country
CountryId	Id
CountryName	Name

Why was the *CountryName* attribute added to *AmusementPark* in addition to *CountryId*?

Answer: The *CountryName* attribute must be displayed on the transaction screen to show the name of the country, which is the data we remember best about it, instead of seeing only its identifier. It is also necessary to add the attribute in the transaction structure if you want to use it later, for example, within a rule.

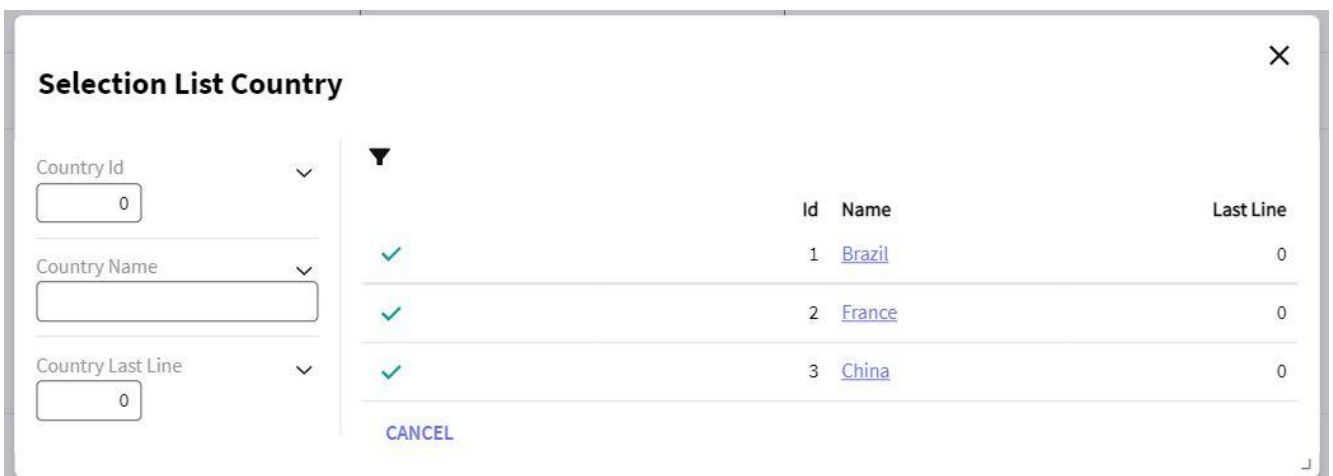
Run it (F5) and the following report will be displayed:



Why isn't the *CountryName* attribute included in the *AmusementPark* table, which GeneXus informs that should be changed in the database? That is to say, why will the physical table not contain it, when it is in the transaction structure?

After examining the report, if we agree, we click on **Reorganize** to effectively carry out what is reported. The browser will open showing the menu with links to the 3 programs corresponding to each of the transactions (*AmusementPark*, *Country*, and *Employee*).

Enter these **countries**: Brazil, France, and China. Note that if you leave the value 0 as a value in the identifier, when you save it is automatically assigned the number that comes after the last one assigned (in fact, it is auto numbered).



Enter this amusement park: “Beto Carrero World,” which is in Brazil. If you don’t remember Brazil's identifier in the system, how do you enter the country? An arrow icon is provided next to *CountryId* to open a “Selection list” of countries automatically created by GeneXus. The reason is that *CountryId* plays the role of foreign key in this transaction (that is, it is “pointing” to another table).

RELATED DATA: HOW IS INTEGRITY MAINTAINED?

AmusementPark and *Country* are related. When *CountryId* is placed in the structure of *AmusementPark*, since it has exactly the same name as the attribute that is a primary key in the *Country* transaction, GeneXus understands that the *CountryId* attribute is a foreign key in *AmusementPark* and automatically maintains the integrity of the data. For example:

- Try to enter an amusement park with a country ID that does not exist. Can you save that country?
- Choose a park previously entered (for example, ‘Beto Carrero World’) and change the country to one that does not exist. Are you able to save the change?
- Try to delete a country (using the *Country* transaction) that has an associated park (e.g. Brazil). Can you do it?

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

Return to GeneXus and change the data type associated with the *AmusementParkId* attribute to the *Id* domain. Run the application. What does the Impact Analysis show? Click on **Reorganize**.

'SHOW' TRANSACTION

They inform us that the amusement parks can offer different shows (musical, magic, educational, etc.) to visitors on certain dates and times. Regarding the shows, we need their names and a photo. The same show can be hired by several amusement parks.

'GAME' TRANSACTION

As initially requested, the system must allow entering the games available in each park, so we must create a transaction with their names and the amusement park to which they belong.

'CATEGORY' TRANSACTION

We have to complete the information of the *Game* transaction. The employees said that they also record the category (kids, radical, fun, etc.) to which each game belongs. Therefore, we need to create a transaction to record that information, and add the category to the *Game* transaction.

In addition, we have been informed that it is not mandatory to register the category to which a game belongs since it is being manipulated. It can be left empty. If we know that GeneXus automatically controls integrity, how can we do it?

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 finish creating the *Game* transaction, let's add the missing data: a photo.

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

Build the application from GeneXus and try it at runtime (F5).

Note what the Impact Analysis informs. The *Category* and *Game* tables will have to be created (don't worry about understanding why you need to store two values per image).

Reorganize and run.

Enter categories (such as kids and radical) and rides (such as roller coasters and carousels).


Note that in this case the category can be left empty (because the **Nullable** property has been set to **Yes** in the transaction structure).

However, if you try to set the *CategoryId* of the game to a non-existent value, it will not let you save it.

Don't forget to update the changes in GeneXus Server.

GeneXus™
Application Name

Game

Id	0
Name	<input type="text" value="Big Tower"/>
Amusement Park Id	<input type="text" value="1"/>
Amusement Park Name	Beto Carrero World
Category Id	<input type="text" value="2"/>
Category Name	Radical
Photo	

“EMPLOYEE” AND “AMUSEMENTPARK” TRANSACTIONS, RELATED

As we were told at the beginning of the application development, the system will manage the amusement parks and their employees. Therefore, we need to link the registered employees with the park in which they work.

To this end, select the *Employee* transaction and add the attributes *AmusementParkId* and *AmusementParkName*.

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 have been informed that it is not mandatory to register the park where the employee works; that is, it can be left empty. What should we do?

Run it (F5) and the following report will appear indicating that the AmusementParkId attribute now allows you to leave an unspecified value:

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.

Reorganize Cancel

Pattern:

Employee

Table Employee specification

Table name: Employee

Employee needs conversion

Table Structure

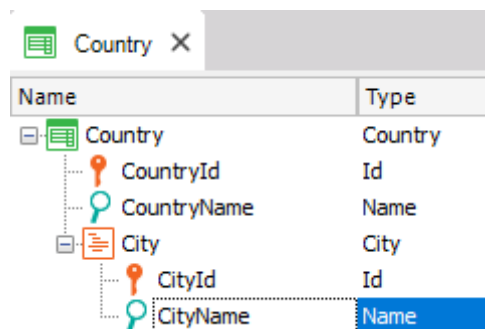
Attribute	Definition	Previous values	Takes value from
EmployeeId	Numeric (4), Not null, Autonumber		Employee. EmployeeId
EmployeeName	Character (20), Not null, NLS		Employee. EmployeeName
EmployeeLastName	Character (20), Not null, NLS		Employee. EmployeeLastName
EmployeeAddress	Varchar (1024), Not null,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

Indexes

0 Errors 0 Warnings 1 Success

ADDING CITIES TO THE 'COUNTRY' TRANSACTION

In addition to the countries, we need to record information about their cities. Therefore, we need to add a second level to the *Country* transaction, with the city identifier and name.



Remember that:

- By selecting the *CountryName* attribute, right-clicking, and selecting **Insert Level**, you can add the sub-level.
- Once you give a name to the new level by typing quotes (") instead of a period, the attribute you define will be initialized with the name of the level.
- The cities will be identified by their own ID combined with that of the country. In other words, you cannot identify a city without first providing information about the country in question. Thus, there could be a city 1 Rosario for both Uruguay and Argentina:
Country: 1 (Uruguay) - City: 1 (Rosario)
Country: 2 (Argentina) - City: 1 (Rosario)
- Or even Rosario for Argentina could be identified with another number:

Country: 2 (Argentina) – City: 4 (Rosario)

Reorganize and run (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.

Reorganize Cancel

Pattern: CountryCity

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, NLS		

Indexes

Name	Definition	Composition
ICOUNTRYCITY	primary key Clustered	CountryId CityId

0 Errors 1 Warnings 0 Success

Note that the Navigation List will inform you that:

- The **Autonumber** property for the case of **CityId** will be ignored. This means that at runtime the user will have to 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 for the cities.

Enter cities for the countries you have already registered.

“AMUSEMENTPARK” TRANSACTION: ADDING THE CITY

In the *AmusementPark* transaction, add the city in which the park is located. What should you do if the company informs you that this value may not be known or relevant for a given park at a given time?

Build the application and try it (F5 and Reorganize).

Update the changes in GeneXus Server.

ADDING BEHAVIOR TO TRANSACTIONS (RULES)

After testing with us the application that we have been developing, we're told that for the employees there is some specific behavior that we must enforce when handling the information through the program (*Employee* transaction).

What is this behavior?

We're told that:

- “The system must not allow entering employees without a name and a last name.”
- “The user should be warned if they leave the phone blank, in case it was an oversight.”
- “The employee's date of entry into the system must be recorded (**EmployeeAddedDate**) and the default value for that attribute should be today's date.”

Specify this behavior and try it (F5 and Reorganize).

Remember that:

- Rules end with a semicolon “;”.
- The **IsEmpty()** method applied to an attribute returns True when the attribute is empty and False otherwise.
- The **&Today** variable is a system variable and has the value of the day's date loaded.

To write a variable inside the **Rules** screen, when you type "&" all the variables defined so far are displayed for you to select the one you need. Another possibility is to use **Insert / Variable**.

Try entering a new employee by leaving the name empty. Does it allow you to save or move to the next field?

The same with the last name. Does it also happen with the phone number?

If you are then informed that the entry date into the system should not be manipulated by the user, but only viewed. How do you configure this behavior?

Specify and try it at runtime.

In the Country Transaction, add a rule that allows the CityId attribute to be autonumbered.

Specify and try it at runtime.

PATTERNS: IMPROVING THE INTERFACE FOR WORKING WITH INFORMATION

When showing the client what has been done so far, we are told that they would like to be able to handle the information on countries, amusement parks, employees, shows, categories and games in

a more powerful and attractive way (offering queries, filters, and the possibility of inserting, modifying and deleting data, etc.).

To this end, you will have to apply the *Work With for Web* pattern to transactions. Try it at runtime.

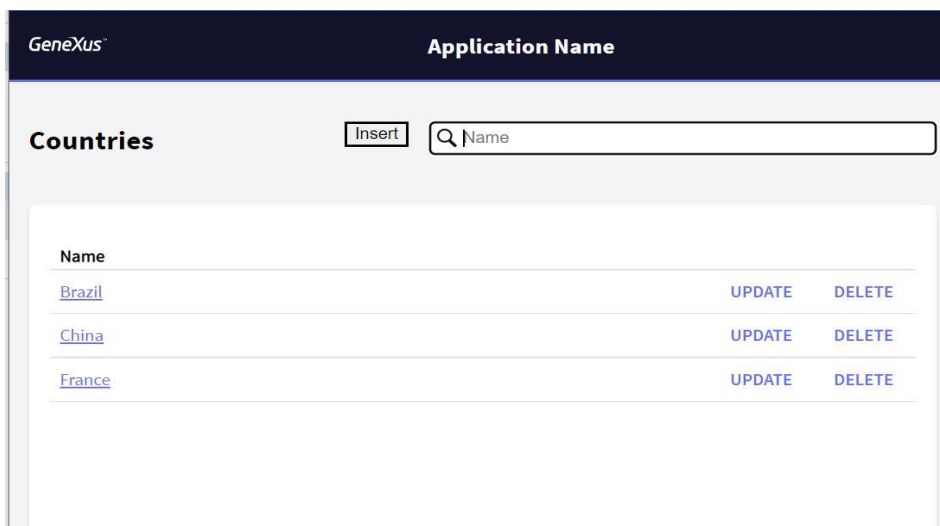
Note that:

- There is also a Work With for Smart Devices. But the one you should apply is the one that corresponds to the web application you are building.
- GeneXus will automatically create several objects per transaction, to implement the element "Work with" that entity.

Why do the transactions *Country*, *AmusementPark*, *Employee*, *Show*, *Category* and *Game* no longer appear in the *Developer Menu*?

Try the following:

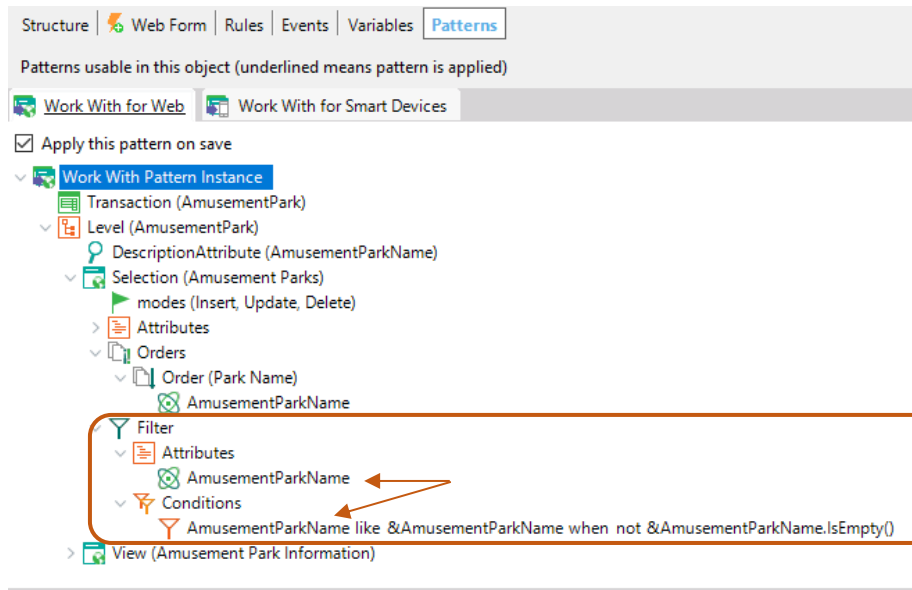
1. Enter a new country.
2. Make changes to an existing country (for example, by adding a city).
3. Delete an existing country,
4. View the information about a country.
5. Make a search by country name.



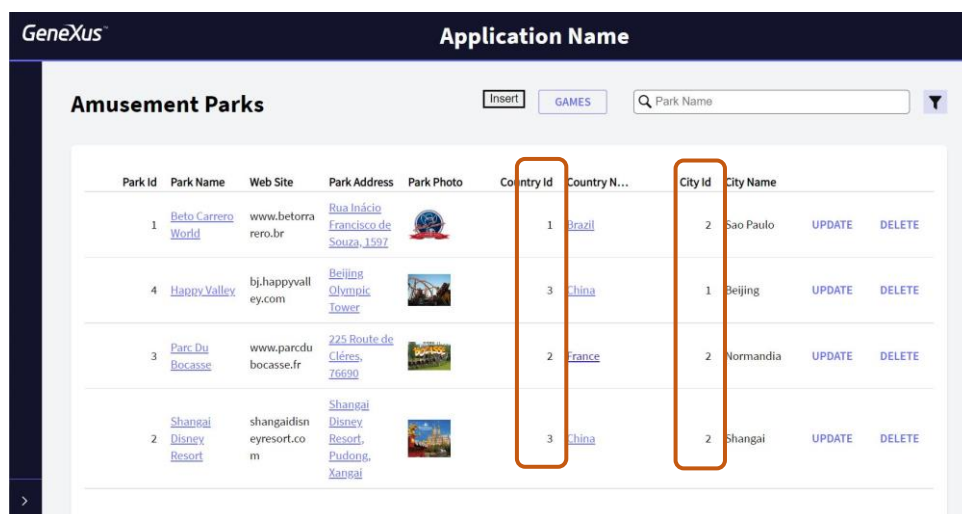
6. Enter a couple of amusement parks (For example: Shanghai Disney Resort, in China/Shanghai, Parc Du Bocasse in France/Normandy, Happy Valley in China/Beijing).

7. Filter the amusement parks whose names start with a P. What if now you want to view all the parks in China? This possibility is not included, so you will have to customize the *Work With* pattern of this transaction, to add it. Do it in **GeneXus** and **try it at runtime**.

Tip: Note how the filter that does exist is specified, by name of the amusement park. Try to define the requested filter in the same way.



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



9. If now you want to allow the user to choose whether to view the amusement parks sorted by park name or by country name, **implement** it and **try** it.

The screenshot shows a web application interface for 'Amusement Parks'. At the top, there is a search bar labeled 'Park Name' and a dropdown menu for 'Ordered By'. The dropdown menu is open, showing three options: 'Park Name', 'Country Name', and 'Country Name'. A blue arrow points to the 'Country Name' option. Below the search bar is a table with the following columns: Park Id, Park Na..., Web Site, Park Ad..., Park Ph..., Country..., and City Name. The table contains four rows of data, each with a small image icon and 'UPDATE' and 'DELETE' buttons.

Park Id	Park Na...	Web Site	Park Ad...	Park Ph...	Country...	City Name		
1	Beto Carrero World	www.betorrarero.br	Rua Inácio Francisco de Souza, 1597		Brazil	Sao Paulo	UPDATE	DELETE
4	Happy Valley	bj.happyvalley.com	Beijing Olympic Tower		China	Beijing	UPDATE	DELETE
3	Parc Du Bocasse	www.parc-dubocasse.fr	225 Route de Clères, 76690		France	Normandia	UPDATE	DELETE
2	Shanghai Disney	shangaidisneyresort.com	Shanghai Disney Resort		China	Shanghai	UPDATE	DELETE




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

It is now necessary to record the games that go into repair status. Each repair case has an identifier, a date from which the game is no longer available for use, the estimated number of days for repair, the game identifier, its name, a regular technician and a substitute technician. Also, each repair has a cost. For the cost, create a domain called *Cost* of the type Numeric(8.2).

Create a transaction to record the technicians who will work on the repairs. Each technician has an identifier, a name and a last name, a phone number, a country and a city in which he is located.

How is it defined that each repair has a regular technician and a substitute technician?

Remember

- 1) That in the transaction structure:
 - An upward arrow icon  informs that the attribute is a foreign key; that is, it points to another table.
 - A downward arrow icon  informs that the attribute is inferred from another table.
 - An  icon indicates that the attribute is a subtype.
- 2) Regarding subtype groups:
 - They are defined in the same way as any other type of object.
 - Each group of subtypes must contain a subtype of a primary attribute (which is the primary key of a table) or a set of attributes that make up a primary key.
 - In each group of subtypes, you need to include all the subtype attributes that have to be known, and that belong to the base table and/or extended table of the primary key of the group.

Run it and **check** that when trying to enter a repair, an error is triggered if the regular technician you are trying to assign to the repair does not exist. Do the same for the substitute technician.

A repair whose regular technician is the same as the substitute technician should not be allowed to be entered. Implement this behavior and try it at runtime.

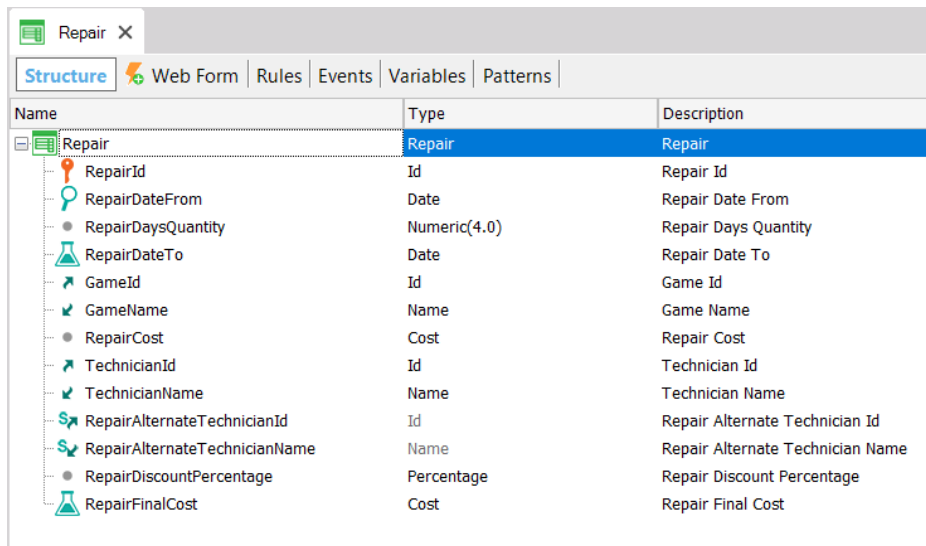
Update the changes in GeneXus Server.

FORMULAS

You need to be able to record the **current discount for each repair**. Define a new attribute in the *Repair* transaction **to store this data**. Enter the name *RepairDiscountPercentage* for the new attribute, whose data type must be a *Percentage* domain, numeric of 3 digits.

The final price of the repair should be displayed with the discount applied. To solve this, define one more attribute, named *RepairFinalCost*, which is a **global formula** and automatically calculates the final price of the repair.

Add a new field called *RepairDateTo*, which will be a sum between the repair start date and the number of days the repair will take.



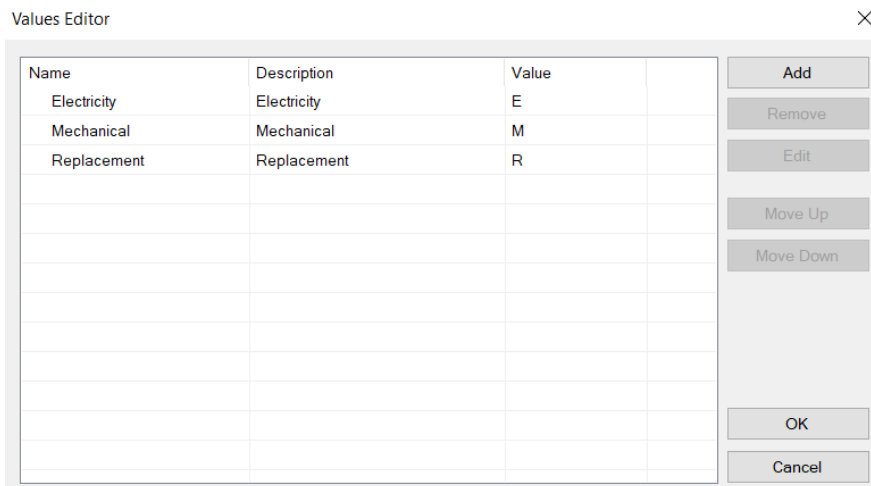
Name	Type	Description
Repair	Repair	Repair
RepairId	Id	Repair Id
RepairDateFrom	Date	Repair Date From
RepairDaysQuantity	Numeric(4.0)	Repair Days Quantity
RepairDateTo	Date	Repair Date To
GameId	Id	Game Id
GameName	Name	Game Name
RepairCost	Cost	Repair Cost
TechnicianId	Id	Technician Id
TechnicianName	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, check in the Impact Analysis which attribute will be physically created and which will not, reorganize, and test the application at runtime.

CREATING THE SECOND LEVEL

A second level needs to be created in the *Repair* transaction to save the details of the type of problem found for repair.

To do so, first create a domain called *KindName*, Character(1). Restrict the possible values for the domain: the values “E,” “M,” and “R” should be valid (editing its **Enum Values** property as seen below).



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

Create a second level in the *Repair* transaction called *Kind* to record the type of repair. This level will have these three attributes:

- *RepairKindId* – Numeric(4) (it will be the key in this second level)
- *RepairKindName* – Based on the *KindName* domain (GeneXus will automatically suggest it).
- *RepairKindRemarks* – Character(120), will contain some comments about the problem, such as a short description of the problem found or the part to be replaced.

Remember that to define a certain attribute as part of the primary key, you must right-click on the attribute, and the context menu will offer you the **Toggle Key** option. In this case, it will not be necessary since 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 involved in a repair, create a new attribute in the first level of the *Repair* transaction, named *RepairProblems*, Numeric(1) and define it as a global formula. This attribute should count the types of problems found.

A repair could involve electrical or mechanical problems or the need to replace a part. There could even be more than one problem of the same type, such as two records of electrical problems, for example. If there were many, you may add more details with the comments attribute, which allows you to write a short text.

The number of problems should be viewed on the web form and it should be checked that between 1 and 3 lines of problem types are entered.

This control should be performed when you finish entering data in the second level 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 usual thing would be to give it values starting from 1. Do what is necessary to autonumber this attribute (remember that it is **not possible** to autonumber this attribute with the *Autonumber* property).

Update the changes in GeneXus Server.

INDEXES

Select the *View* → *Tables* menu and place the necessary controls so that the name of a country or amusement park cannot be repeated.

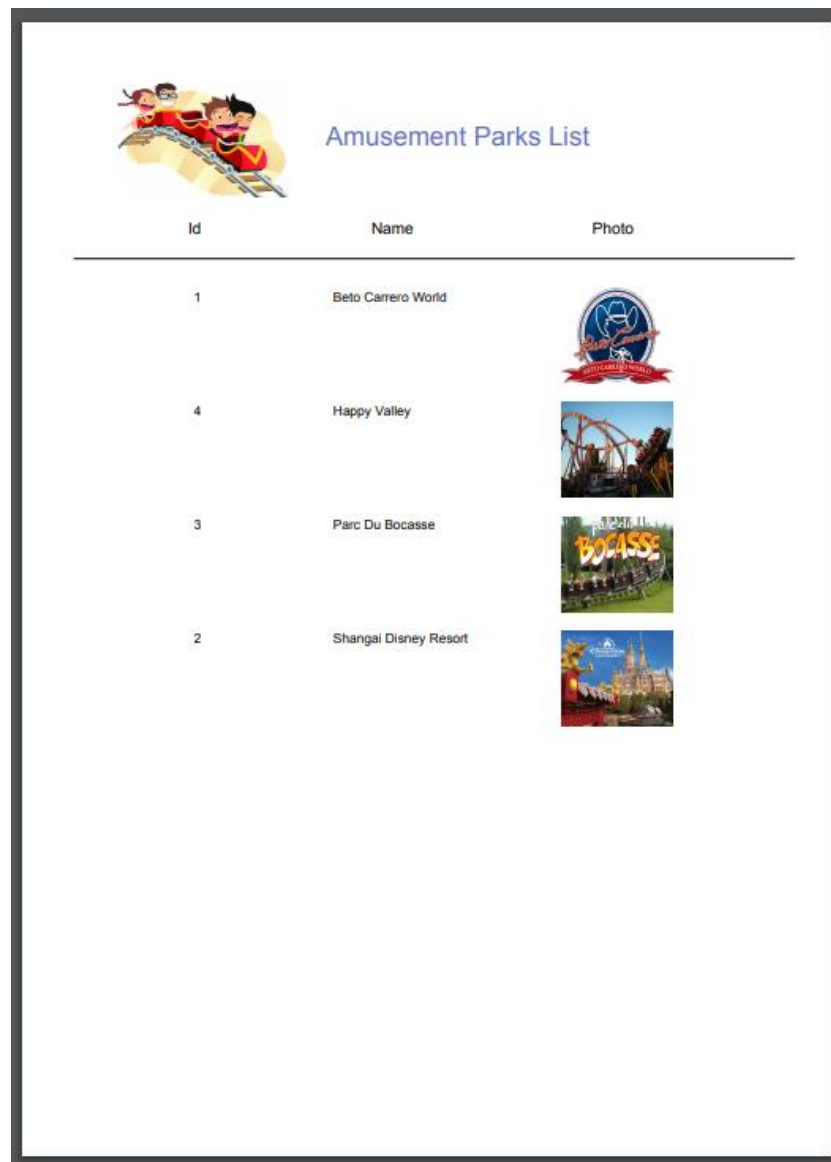
In what section are these controls specified?





What message is displayed in the browser if you try to add a country/amusement park whose name is repeated?

PDF LISTS

Now let's suppose that as part of the application you have to implement the possibility of displaying PDF lists with the required information at the user's request. For example, suppose you need a list showing in alphabetical order the amusement parks stored in the database.

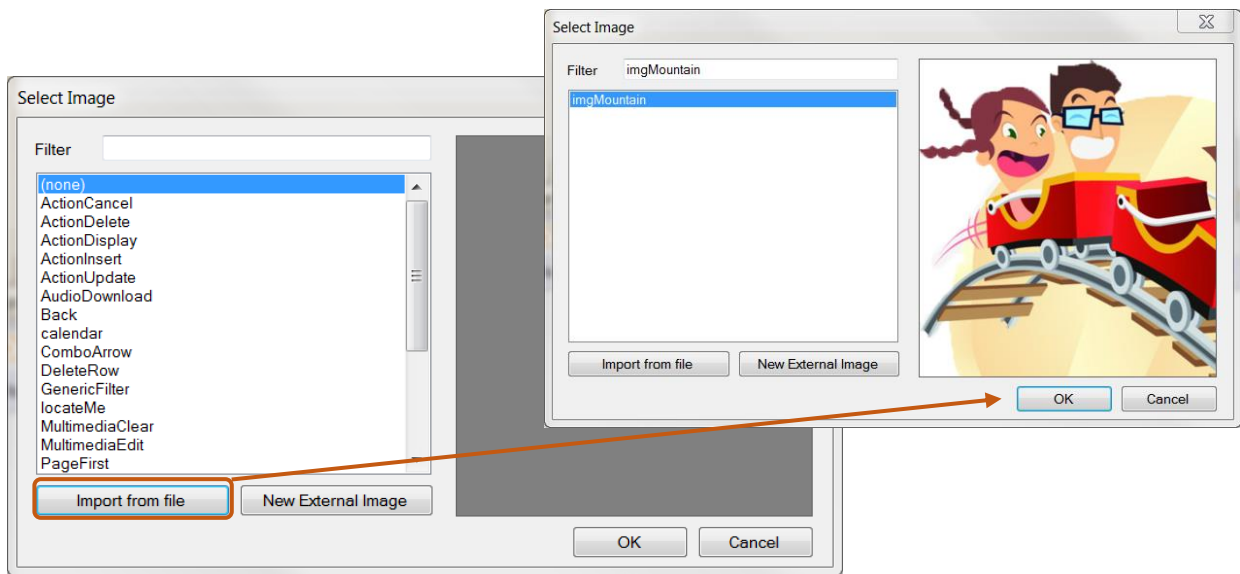
We know it should look similar to this:



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

Suggestion to put an image in the title: Use the *Image* control in the *Toolbox* to display an image next to the title of the list.

This image should be integrated into the knowledge base. To do this, select the *Import from File* option, search for the image and give it a name.



Implement it in GeneXus.

Remember that in order to view a list directly from the browser, it must be generated as a PDF. For this, you must set the following properties of the procedure object:

- Main program = 'True'
- Call Protocol = HTTP

And the following rule:


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

To run the list, on the object tab, right-click / **Run with this only**



Have you read the information in the navigation list of the procedure?

What if the list now has to be sorted by country name? Implement it, check the information in the navigation list and test it.

What if now you only need to list the amusement parks in China? Try it (looking at the navigation list).




Amusement Parks List

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

In each case, figure out which table in the database is being run through to perform the *For Each* command query.



A list like the one below is also needed (showing each category, and for each of them, their games). Implement it and test what you have 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	

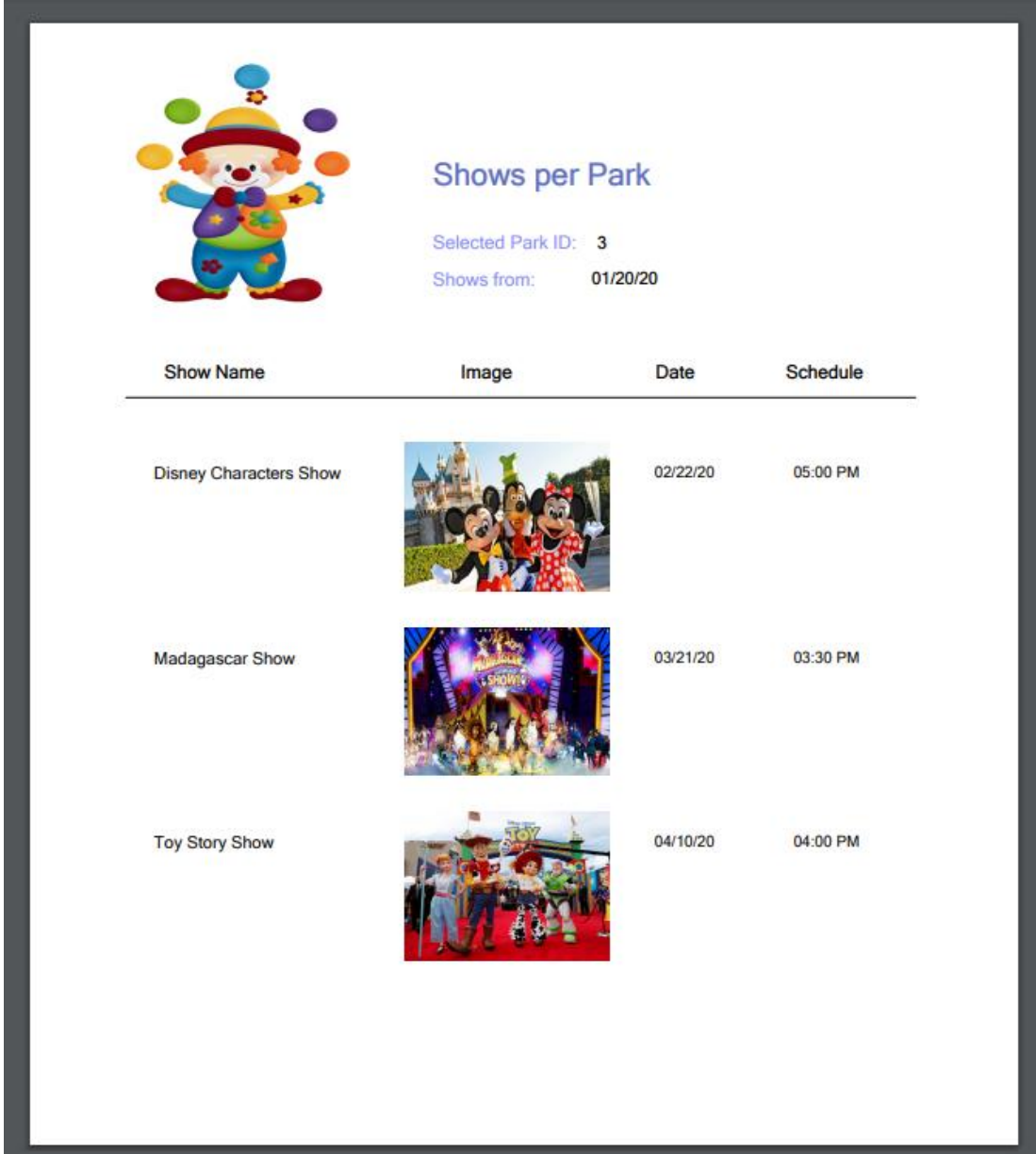
Add a new category to the system, for example Water games. Run the previous list again. Was the category listed?

Modify the above list so that no categories that do not have related games are listed.



What changes did you find in the navigation list?

Answer: The word *break* has been added to indicate that a **control break** is made, also showing that the base table of the external *For Each* command is the same as that of the nested *For Each* command.

Another necessary list is that of the shows offered from a date indicated by the user, in a certain amusement park also selected by the user.



The screenshot displays a web application interface for "Shows per Park". It features a clown mascot on the left and a title "Shows per Park" on the right. Below the title, there are two filter fields: "Selected Park ID: 3" and "Shows from: 01/20/20". A table below lists three shows with their respective images, dates, and schedules.

Show Name	Image	Date	Schedule
Disney Characters Show		02/22/20	05:00 PM
Madagascar Show		03/21/20	03:30 PM
Toy Story Show		04/10/20	04:00 PM

The company has also requested a list that shows all country names and for each country, the number of amusement parks they offer.



Countries list

Country	Quantity
Brazil	2
France	1
China	2
United States	3

In addition, they have requested another list that shows all the countries that have more than 2 parks to visit:



Countries list

Country	Quantity
United States	3

PASSING PARAMETERS

Many times, we need an object to receive parameters so that it executes its logic based on them. For example, it is one thing to make a PDF list of all the amusement parks in the database, and another to make a list of those whose name is within a given range.

LISTING OF PARKS IN A CERTAIN RANGE

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

Implement a screen that asks the user for the values of that range, and invokes this object, passing it those values by parameter. Test it at runtime.

Remember that:

- If you define a variable based on an attribute, it will be linked to the attribute's data type; that is, if the attribute is modified, the variable's data type will also be modified, according to the change.
- The variables used to perform an invocation in the calling object and those used to declare the parameters received in the called object do not need to have matching names, but they must have compatible data types.

Tips:

- Define the corresponding Parm rule in the procedure *AmusementParksFromTo* so that it receives the values to query.
- Create a Web Panel with two variables for the end user to enter the values to be queried so that they can be sent by parameter to the procedure.

BUSINESS COMPONENTS

We will perform some operations on the database through *Business Components*.

PRICE INCREASE FOR REPAIRS

The company is entitled to increase the prices of repairs by a certain percentage from a certain point in time. In order to save this change on a massive scale we will try this functionality. To do this, you need to implement a **screen** for the user to specify that **increase percentage**, and give the **order** to apply it to all the repairs in the database. **Implement** it and **test** it.

Remember that:

- The Web Panel object allows you to implement flexible screens for information input and output.
- To enter information (to allow the user to enter values on the screen), you can insert from the *Toolbar* an *Attribute/Variable* control in the Form and assign a variable to it.
- To edit menu bars, from GeneXus at the top of the bar, right-click to insert, for example, the *Formatting* bar.
- For the Web Panel to perform a certain action buttons can be inserted and the associated "event" can be programmed.
- *Business Components* are data types created when configuring the property called *Business Component* of the transaction object to Yes. In doing so, to insert, modify or delete records from the corresponding tables, you can use, in addition to the transaction, a variable of the *Business Component* data type in any other object (for example, a Web Panel) and perform the same operations through the *Load()*, *Insert()*, *Update()*, *Save()*, and *Delete()* methods.
- To make the operations performed through the *Business Component* permanent, you must then execute the *Commit* command.
- To increase a value X by 20%, it is enough to type $X = X*(1+20/100) = X*1,20$

Tips:

- Create a Web Panel with a &percentage variable of this data type: Numeric(3.0), and drag a control of Button type to the Form.
- In the event associated with the button, program the logic to be executed when the user presses the button.
- Remember that all repairs should be run through, and for each of them, using the concept of Business Component, their price should be modified by increasing it by the percentage indicated by the user in the variable on the screen.
- Also, remember that since operations to save or delete from the database (through the *Insert()*, *Update()*, *Save()*, and *Delete()* methods) may cause errors, it is important to know what happened. To this end, we have the *Success()* method that will return True if there were no errors, and False otherwise.
- To indicate that the changes should be made permanent, the Commit command must be executed. Otherwise, to undo the changes, the Rollback command must be executed.

Update the changes in GeneXus Server.

SCREEN TO DELETE ALL REPAIRS

Save the previous Web Panel with another name (to do so, click on the tab, right-click and select *Save as*) and change the Form so that it only contains a button with the text *Delete Repairs*; at runtime, this Web Panel should look as shown below:



When the user clicks on the button, all repairs should be deleted from the database.

What should you change in the *Confirm* event that you had programmed?

Note: If you click on the button and view its properties, you can modify the text of the button in the *Caption* property.

Tips:

- Use the *Delete()* method that is applied to variables of Business Component type (simple or collection ones).
- Remember that with the *Msg* command you can display a message in the Web Panel Form.

Update the changes in GeneXus Server.

PROCEDURES TO UPDATE RECORDS

PRICE INCREASE FOR REPAIRS

Suppose that there are thousands of repairs to which you must increase their price by a given percentage. Knowing that the price increase is a simple procedure that will not cause any integrity failures, try to solve it with a procedure without using *Business Components*.

Remember that:

- With the *For Each* command within a procedure, you can update the attributes of its extended table through simple assignments.
- Updating "directly" through procedures does not control the integrity of the information.
- Every object must declare the parameters it receives and the parameters it returns. If it does not declare them, it will neither receive nor return values.
- Parameters are declared through the ***parm*** rule.
- Variables are local to the object where they are used. This means that if I want to receive a value as a parameter in an &X variable, I have to declare it in the object.

DELETING ALL REPAIRS

What if you now wanted to delete all repairs, as was previously done in the practical course, but this time through a procedure?

Tip:

- Create a procedure called *RepairsDeletion* that does not receive parameters, and runs through and deletes all repair records. Use the Delete command.
- Do a Save as of the Web Panel you had implemented and program the deletion through the Business Component.

What if you want to delete all the information from the database?

Tip

- Create a procedure called *DeleteAll* to run through and delete all the records of the various tables associated with the transactions defined.

Remember that procedures do not validate data consistency, but the database does. That is, the database checks the consistency of the interrelated data, so the order in which you try to delete the data is important.

For example, if you try to delete the countries before the parks, the database will prevent it, the program will cancel, and it will not be user friendly.

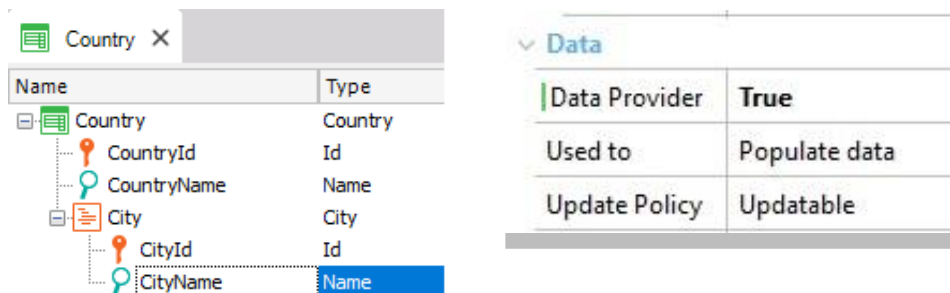
INITIALIZING DATABASE INFORMATION [OPTIONAL]

When the application you are developing is deployed to production (i.e. it starts to be used by the company) all data of countries, parks, shows, categories, games, technicians, and repairs should be loaded. Use the features that the *Transaction* object offers for this, initialize those tables with information provided by the company and try it out.

Remember that:

- Transactions have a **Data Provider** property that allows you to have a Data Provider associated with the transaction, so that when the table is created in the database, it is initialized with the values that you define in that Data Provider. To this end, the **Used to** and **Update Policy** properties will also have to be configured.
- Insert the images in the KB to be able to use them. One way to do so is to open the *KB Explorer* window, select the *Customization* node, choose *Images* –where all the images currently stored in the *KB* are listed– insert a new one (from a file), and give it a name.

Tip: To try out different options, populate the countries and categories using the Data Provider property of the transactions.




Remember that: The Data Provider associated with the transaction will be triggered again:

- Every time the corresponding table is reorganized.
- Every time the Data Provider's content is edited.

Therefore, it is necessary to make sure that the information is not duplicated. This can be achieved as follows:

- If the primary key is auto numbered, a unique index can be defined over the descriptor attribute.

- The auto numbering feature can be turned off for the primary key. In this way, its value will be assigned when creating the Data Provider and it will not be duplicated.
- Load the desired data in Country_DataProvider (you can see it below the Country transaction in the KBExplorer window). When created, the table will be loaded with the indicated data.



```
1 CountryCollection
2 {
3   Country
4   {
5     CountryName = "Brazil"
6     City
7     {
8       City
9       {
10        CityId = 1
11        CityName = "Rio de Janeiro"
12      }
13    }
14   Country
15   {
16     CountryName = "France"
17     City
18     {
19       City
20       {
21        CityId = 1
22        CityName = "Paris"
23      }
24       City
25       {
26        CityId = 2
27        CityName = "Versailles"
28      }
29     }
30   }
31 }
```

- Likewise, load the categories.
- Update the changes in GeneXus Server.

WEB PANELS

A page has been requested that shows all country names and, for each country, the number of amusement parks they offer.

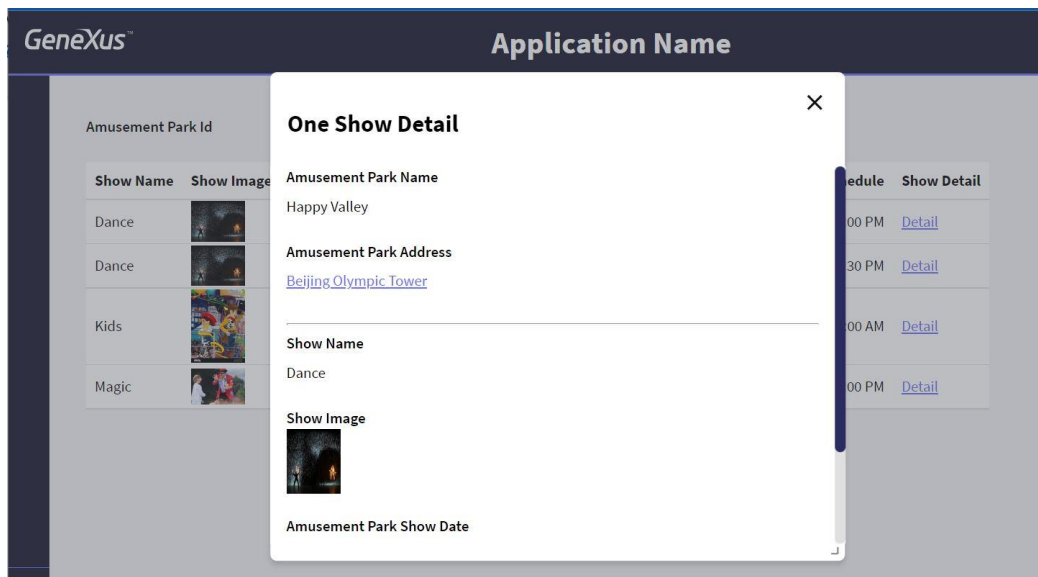
Remember that the **Load event in a Web Panel with a base table that has a Grid** is executed just before each line is loaded into the grid. This event is suitable to assign to the variable the calculation that returns the number of amusement parks of each country that is being navigated and about to be loaded in a line of the grid.

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

Another web page is requested to show the information of the Shows: name of the show, an image representing it, the park to which it is associated, date, and schedules; ordered by the name of the Show.

It is necessary to filter the information according to the park where the Show will be presented; if no park is selected, it must display all the Shows.

Another requirement is a link that allows showing another screen as a pop-up window where the information of the selected show is displayed.



TIP:

To display a pop-up window on the screen, use the following syntax: `Object.Popup(Parameters)`

It is necessary to display a ranking in a web screen according to the number of Shows of each Park.

NOTE: it is not required to define a formula attribute at the Park transaction structure level.

Amusement Park Name	Shows Quantity
Beto Carrero World	4
Happy Valley	2
Par Du Bocasse	1

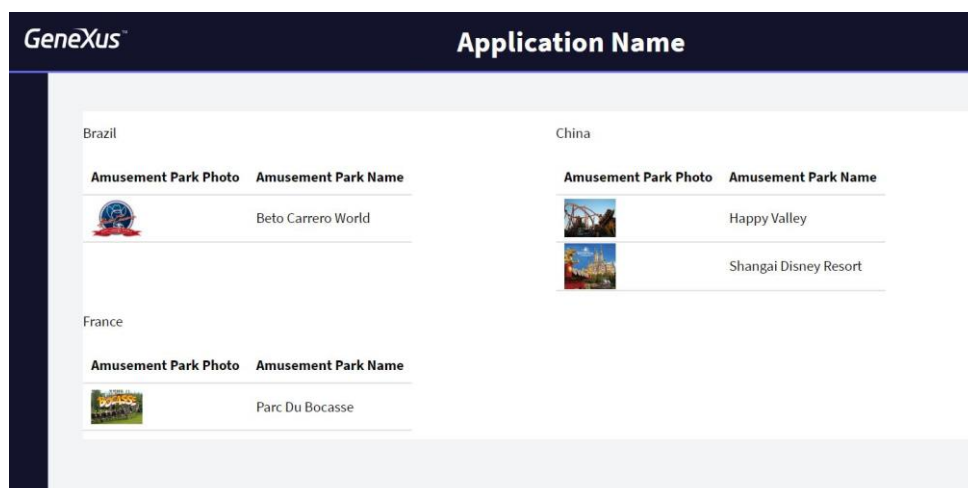
TIP:

Tip: Define an in-memory data structure that allows storing the collection of parks so that it can then be sorted from highest to lowest by the number of Shows and displayed in a Web panel.

MULTIPLE WEB PANELS [OPTIONAL]

You want to see a screen that lists the information of all the countries with their corresponding related parks.

In the Web Panel to be defined, add a Free Style Grid and within it, a Grid –each one with its corresponding attributes so that the information is displayed as follows:



What is the base transaction of the external Grid?

What is the base transaction of the nested Grid?

Add a new country and run the Web Panel again.

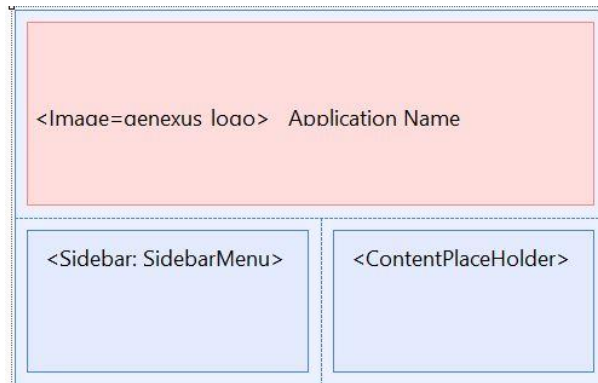
What changes should you make if you only want to see on screen the countries that have related amusement parks?

TYPES OF WEB PANELS

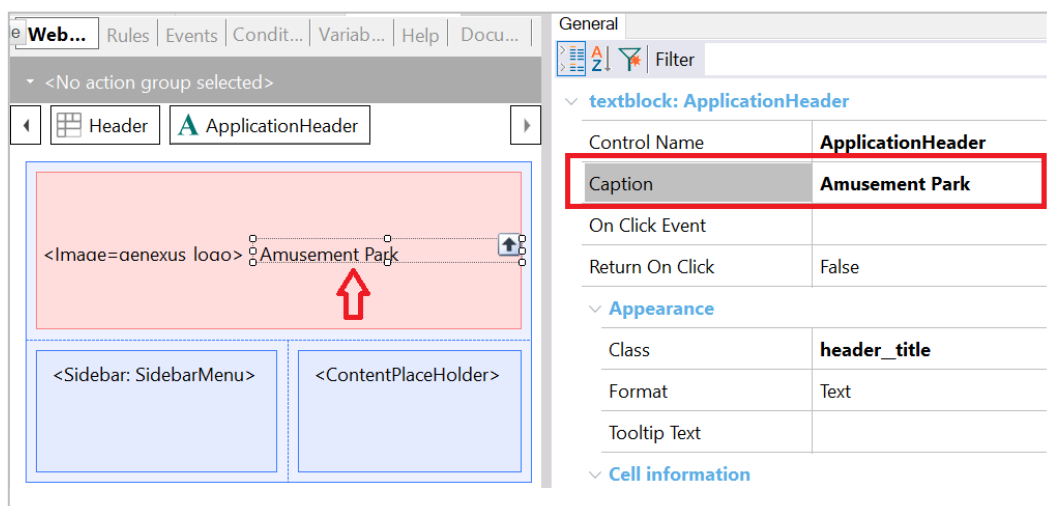
Open the KB Explorer and search in the filter “*MasterUnanimoSidebar*” This is the Master Page associated with all the GeneXus objects in our KB.

Note that it has three sections:

- A Header containing the name of the application and logo.
- A Sidebar User Control that offers the menu of the objects that have the WorkWith for Web pattern.
- A Content Placeholder where the objects of our application are displayed.



In the *Header*, select the *ApplicationHeader* control and change the Caption property to the name of the application: Amusement Park.



General	
Filter	
textblock: ApplicationHeader	
Control Name	ApplicationHeader
Caption	Amusement Park
On Click Event	
Return On Click	False
Appearance	
Class	header_title
Format	Text
Tooltip Text	
Cell information	

GeneXus™ Amusement Park

Amusement Parks

Park Id	Park Name	Web Site	Park Address	Park Photo	Country Name	City Name		
1	Beto Carrero World	www.betorrarero.br	Rua Inácio Francisco de Souza, 1597		Brazil	Sao Paulo	UPDATE	DELETE
4	Happy Valley	bj.happyvalley.com	Beijing Olympic Tower		China	Beijing	UPDATE	DELETE
3	Parc Du Bocasse	www.parcdubo.casse.fr	225 Route de Clères, 76690		France	Normandia	UPDATE	DELETE
2	Shanghai Disney Resort	shangaidisneyresort.com	Shanghai Disney Resort, Pudong, Xangai		China	Shangai	UPDATE	DELETE

DESIGN SYSTEMS

Open the Web Panel where the solution was built to show all the countries and, for each one, the number of amusement parks they offer.

This Web Panel should have the same format displayed in the WorkWithForWeb Grids.

GeneXus™ Amusement Park

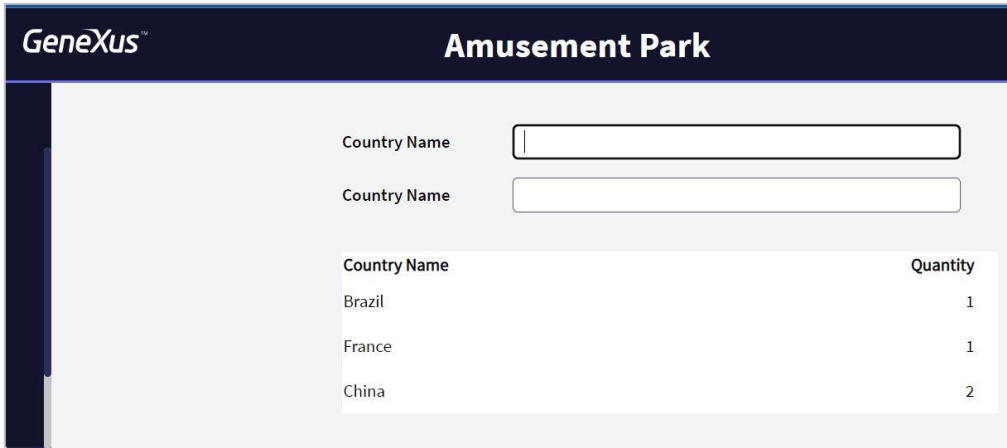
Country Name

Country Name

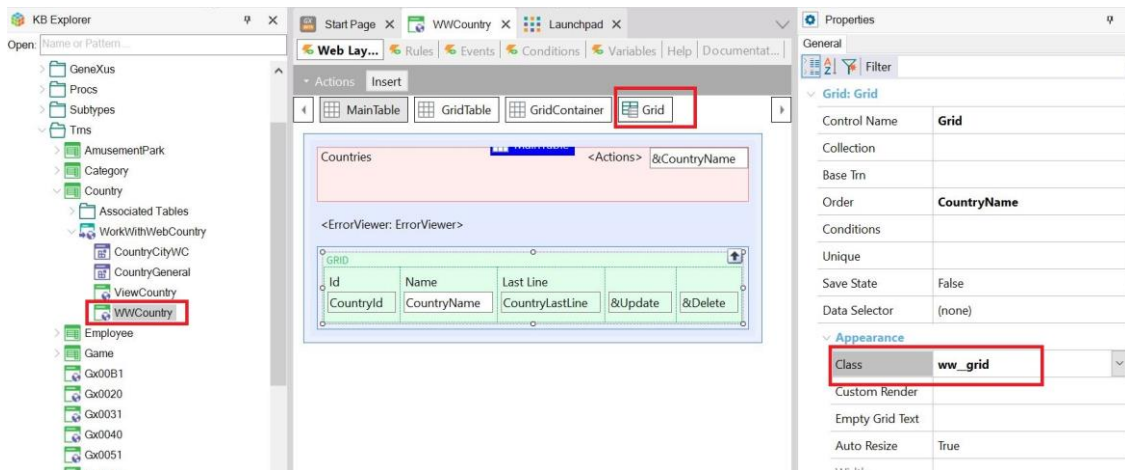
Country Name Quantity

Brazil	1
France	1
China	2

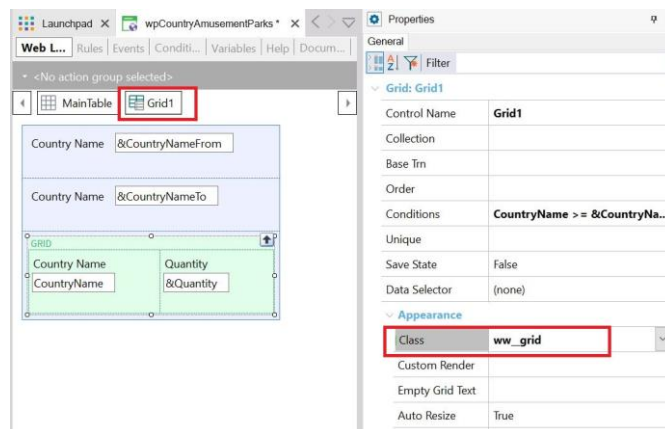
>



In a *WW* Web Panel associated with *WorkWith*, note that in the Grid properties there is an assigned class called *WorkWith*.

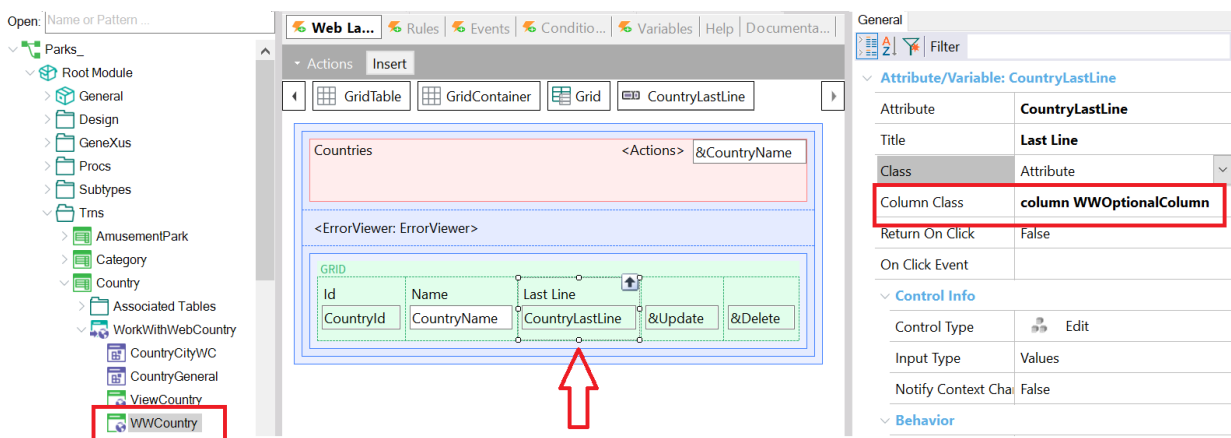
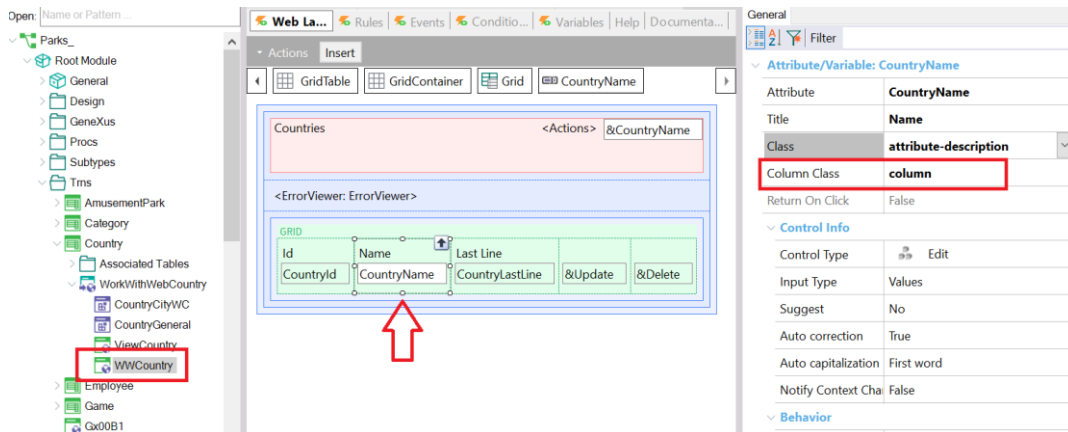


Apply this same class in the Grid control of the Web Panel to which a format will be applied.

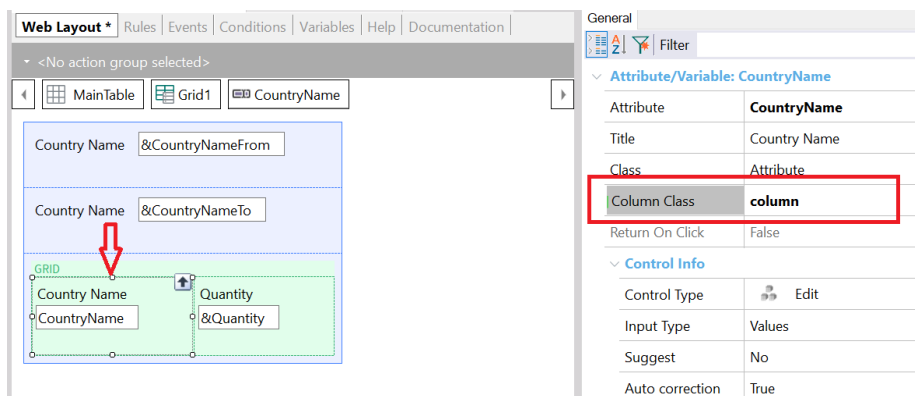


To maintain the same design, note that each column of the *WorkWith* Grid has an associated class:

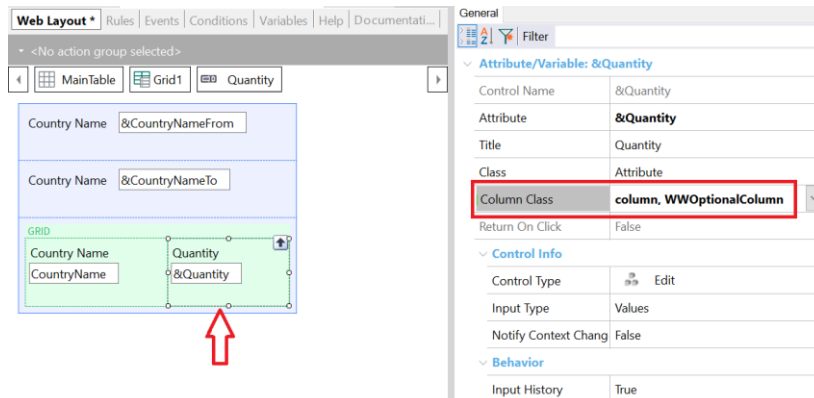
- *Column* for the most important attributes that are displayed at all times, regardless of screen size.
- *Column* and *WWOptionalColumn* for the attributes that are hidden when the screen gets smaller.



To achieve the same behavior in the Web Panel, choose the country name and in the *ColumnClass* property, place *Column*.

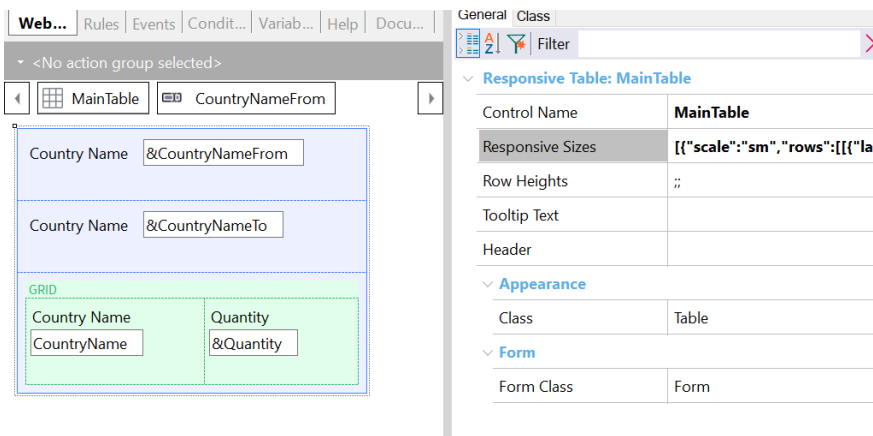


To hide the number of parks when the screen is smaller, select the control associated with the variable and in the *ColumnClass* property, place *Column* and *WWOptionalColumn*.



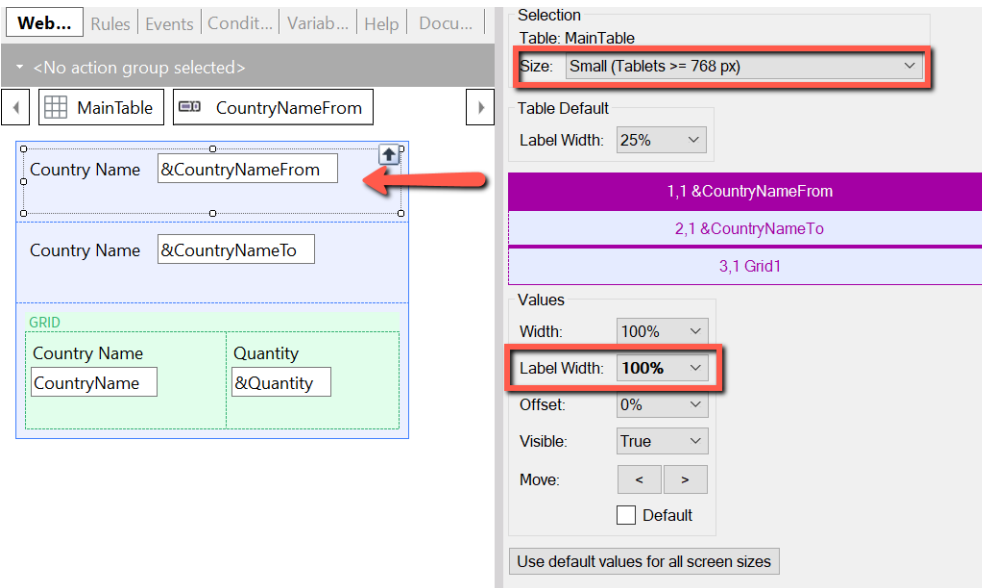
Run the application and test this behavior

Lastly, for the labels to appear above the editable box, open the box of the responsive sizes. If they are not displayed, select the *MainTable* of the Web Panel and click on the Responsive Sizes property.



Select the first variable, in the responsive sizes select the Small size and in *Label Width* place 25%. Change the size to Extra Small and configure the same in *Label Width*.

Do this same process for the second variable.



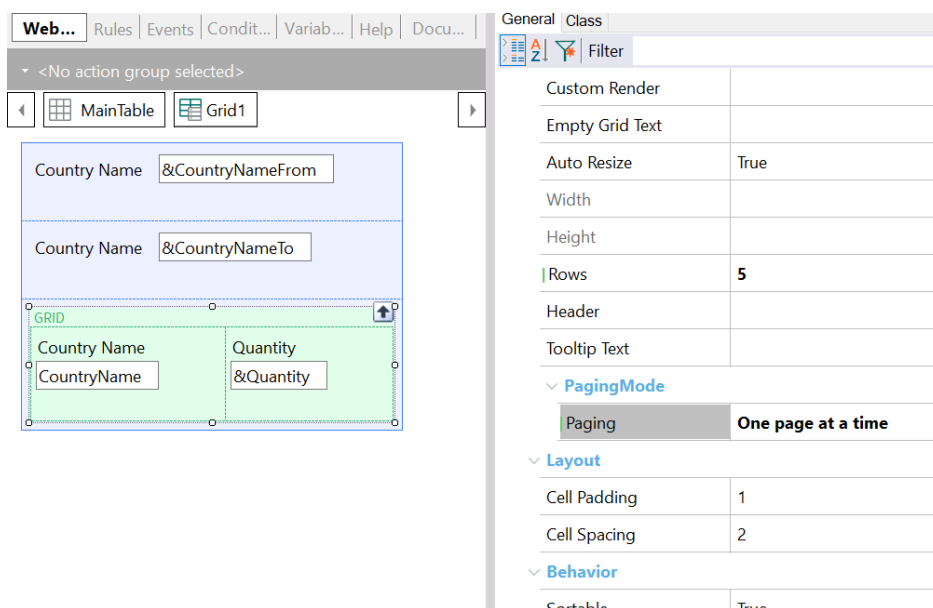
Run the application and test this behavior.

PAGING IN GRIDS

The previous screen is required to have Grid paging to better display the countries.

For this, add more countries.

In the Grid properties of the Web Panel, change the *Rows* property to 5 and in *Paging* verify that *One page at a time* is activated.



Run and note that now only 5 records per page are shown, and thanks to the navigation arrows on the Grid you can move around the countries.



What would happen if you didn't want to see the information in different pages, but simulate that the content is in a single screen and scroll to access the records? Go back to the Web Panel and change the *Paging* property to *Infinite scrolling*.

Run the application and note that you can scroll through the records.



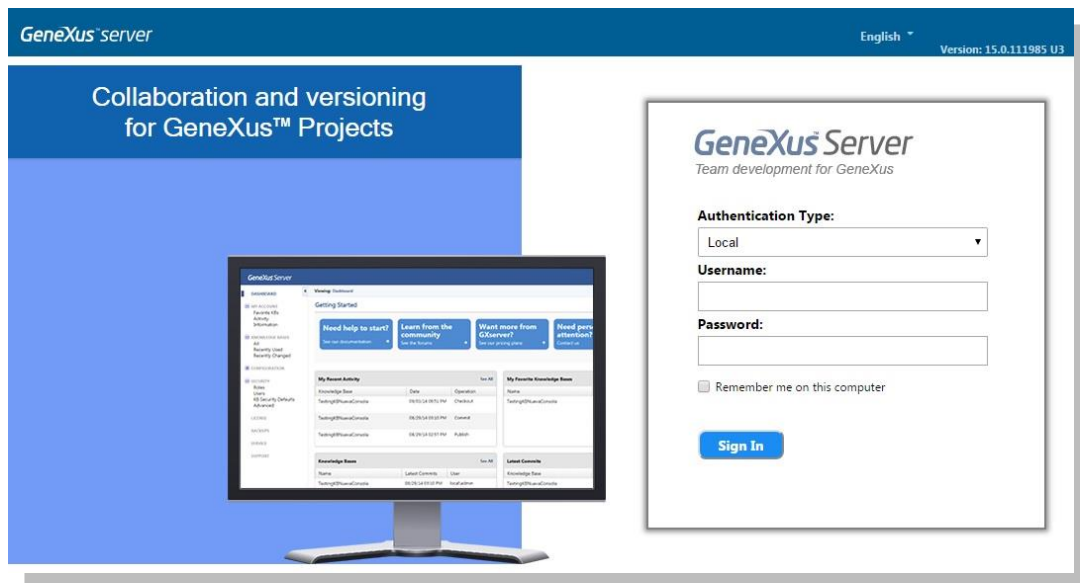
GENEXUS SERVER

Publish the knowledge base on the server at <http://sandbox.genexusserver.com/v17>

Create a new Web Panel that shows the list of registered amusement parks.

Send this new object to the server to be integrated into the centralized knowledge base.

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




Close the previous knowledge base, and create a new one by synchronizing with the previously published *KB*. In this way, you will receive a local copy of the *KB* managed by GeneXus Server.

Send Knowledge Base ✕

Send the Knowledge Base to GeneXus Server

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



http://sandbox.genexusserver.com/v17

Parks

Security

Authentication Type: GeneXus Account

Username:

Password:

Save Password

Alias:

Work with Lock Model

All versions in Knowledge Base

Send
Cancel

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

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