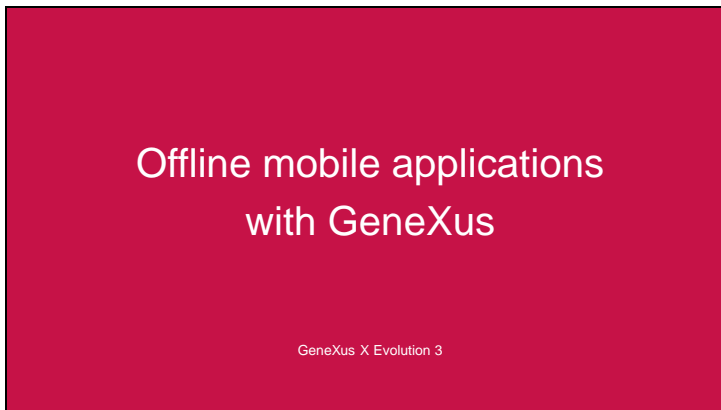
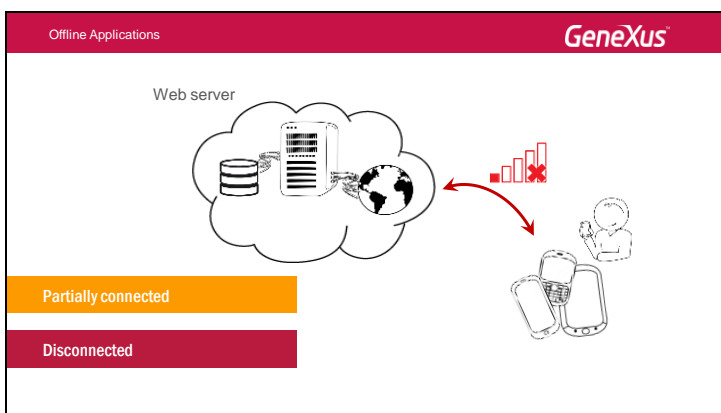


Offline Applications with GeneXus



So far, have developed applications for smart devices that were always connected through the Internet to the web server, which stored the necessary services and data for the application.

An important objective for smart devices is to allow the application, or part of it, to keep running when there is no Internet connection.



Some tasks, due to their sensitivity, must be validated in the central database. Tasks whose data change very often also require permanent access to the web server.

In our example, the login must be online. In addition, the panel showing tweets should also be connected to the Internet.

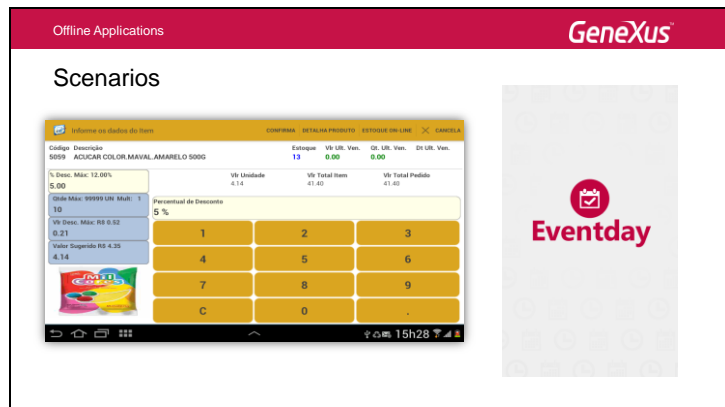
However, other tasks can be run offline and their data can be synchronized later.

Therefore, we have to select which objects of the application can be executed offline and which cannot.

These applications are called **PARTIALLY ONLINE** because they access local data and have the possibility to execute complex logic on the device.

Applications that don't have to be connected to the server once installed are called **OFFLINE** applications. These applications usually manage personal information and no data has to be sent to the server.

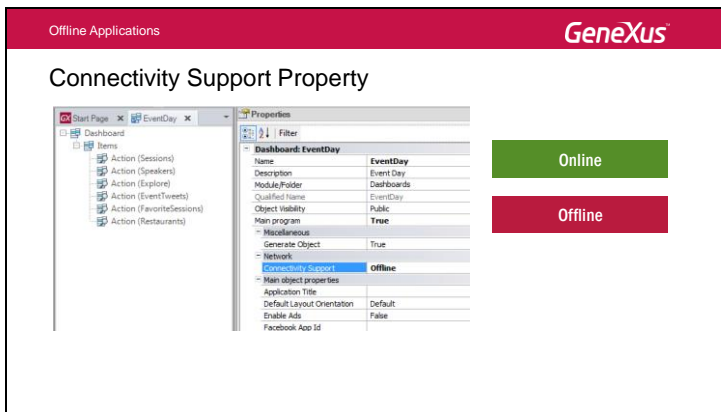
Partially online applications are the most frequently used. As examples of these applications we can mention Points of Sale or apps for Events.



In Point of Sale applications, sales people or distributors who work on the go are enabled to issue purchase orders and invoices, regardless of being connected or not. Then, when their Internet connection is restored, they synchronize their data with those of the company.

Or, as in this example, applications for events contain data that doesn't change very often. The database is loaded when installing the application in order to continue using it offline. On the other hand, certain information requires continuous access to the Internet, such as tweets.

So, how do we tell GeneXus that an application will be used offline?
With the Connectivity Support property, at the level of Main SD objects.
In our example, this object is the dashboard.



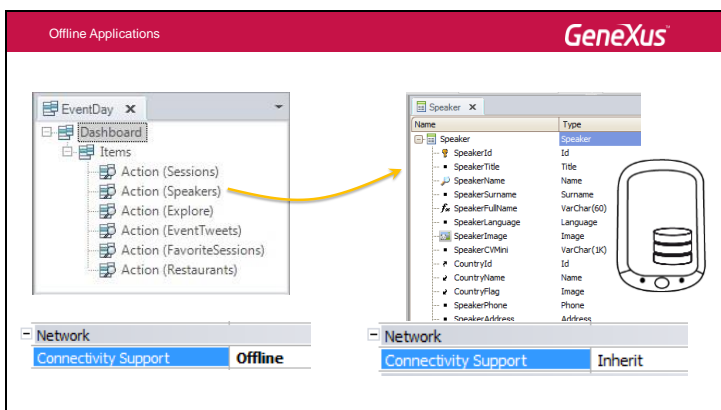
This property can be set as **Online** or **Offline**, indicating how the main object will behave.

In addition, all the objects that are not main and are invoked by the main object have the **Inherit** property to indicate whether they inherit the behavior of the main object that invokes them. This is valid for transactions with associated Business Components, WorkWithDevices objects, SD panels, Data Providers and procedures included in the invocations tree of the main object.

Let's see this with an example.

Suppose there's an application for an event.

What happens when the **Connectivity Support** property of the EventDay Dashboard is set to Offline?

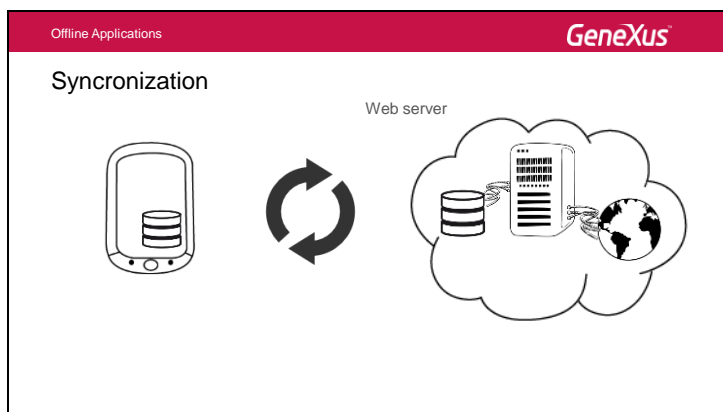


Transactions associated with business components used in WorkWithDevices objects invoked from the Main object (directly or indirectly) and which have the Connectivity support property set with the **Inherit** value, will generate tables that will be created in the local database of the device. In addition, all the tables required for referential integrity or by attributes mentioned in panels or prompts, will be created to make sure

that the offline application works in the same way as when it was connected to the Internet.

In those objects that we want to be always connected, we set the Connectivity Support property to **Online**.

When the application is installed on the device, the database will be created in the device itself together with the corresponding tables. In addition, the first time it is connected, it is possible to bring data from the server to the device to populate the tables.

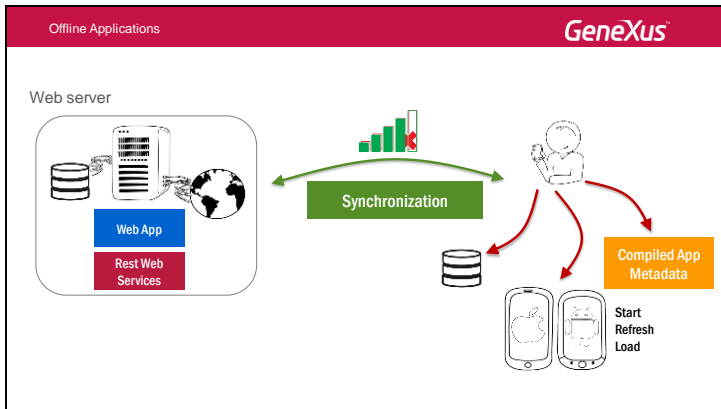


Once this is done, if the connection is lost by the device, the application will keep working with the data stored locally. When the device is connected again, the information locally stored is synchronized with the data saved in the server. The data on the server that has changed is sent to the device to be updated, at regular intervals or on demand.

Later on we will see that this behavior can be changed; for example, to never synchronize it or to synchronize it on demand.

Let's see the architecture of offline applications.





An offline application can be divided into two components:

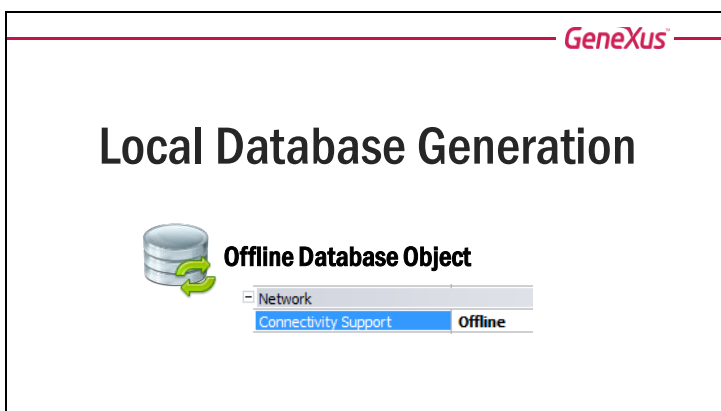
- A local component that includes a local database (normalized) with a subset of server tables and all the logic necessary to be run locally; and...
- A component on the server side, containing the application's backend, the database and the service layer for synchronization with the device.

Both components communicate via REST web services for synchronizing and loading the local database, and then sending modifications locally made to the server.

This communication is called Synchronization.

In order to install the local component on the device, the application must be compiled to include all the information necessary to access REST services.

For every main object that has the Connectivity Support property set to **Offline**, an object called **Offline Database** is created.

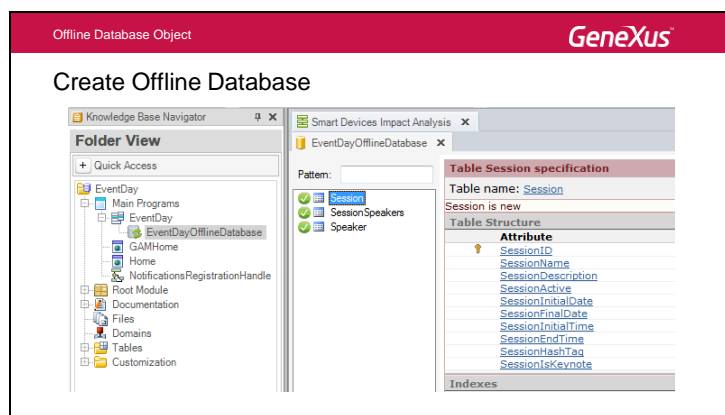


This object is responsible for determining which tables go to the local DB and which data is taken during synchronization, because obviously the local DB will not be the same as the server DB, but a reduced version.

The database created on the device is a SQLite database.

After changing the Connectivity support property of the main object of our application to Offline, the Rebuild All action must be performed.

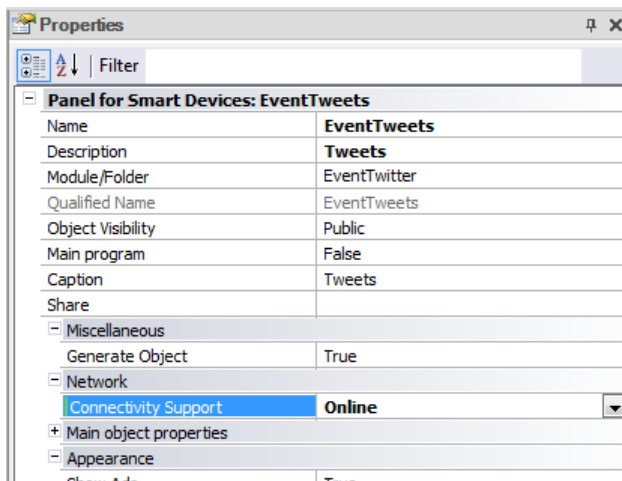
When Building the main object for the first time with the Connectivity Support property set to **offline**, the Offline Database object is created and an impact analysis is made to find out which tables will be created in the device, associated with the offline database object.



We will do this in GeneXus.

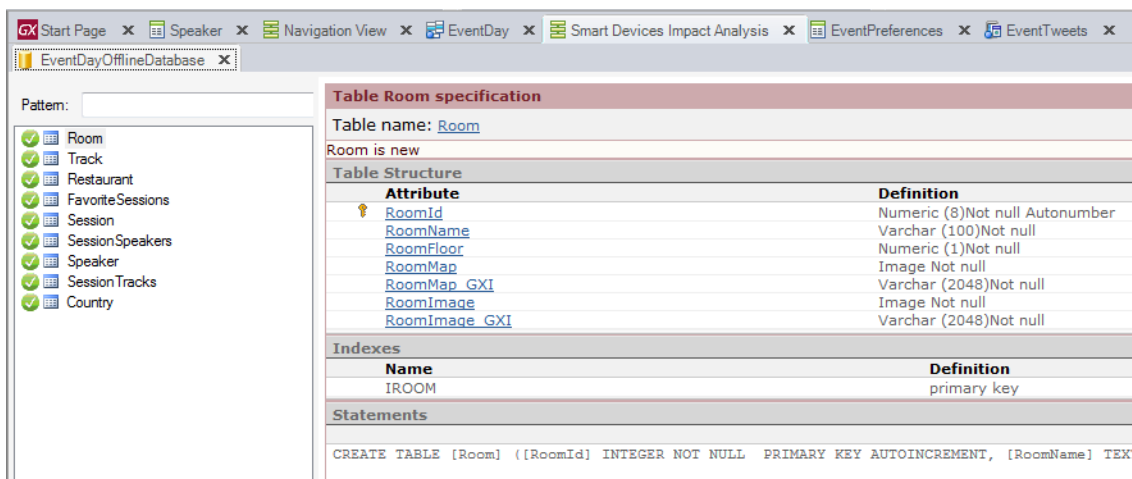
We open the EventDay dashboard object and change its Connectivity Support property from the Online default value to **Offline**. The objective is to make the entire application capable of running with no Internet connection, except for the screen showing tweets, which must work online due to the speed at which this information is updated.

To do so, we open the EventTweets object and change the Connectivity Support property value to **Online**.



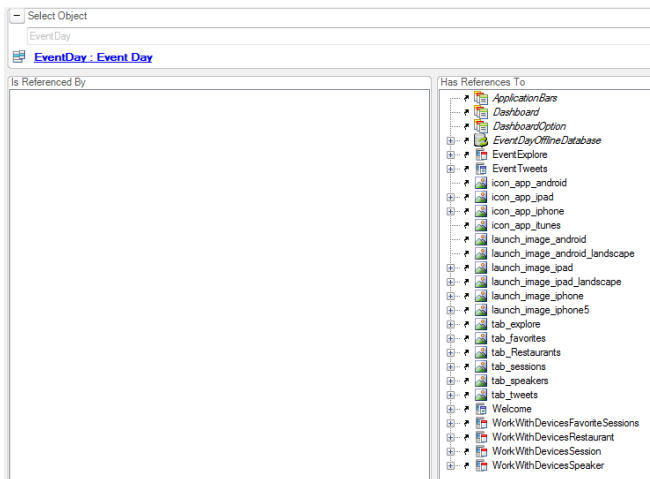
Now we press the Rebuild All button.

We see that the impact analysis report shows the creation of the EventDayOfflineDatabase object and the tables that will be created in the device, which in this case will be all of them because we didn't change the default value of the Connectivity Support property of any transaction, and left all of them with the **Inherit** value.



Note that the EventPreferences table is not included.

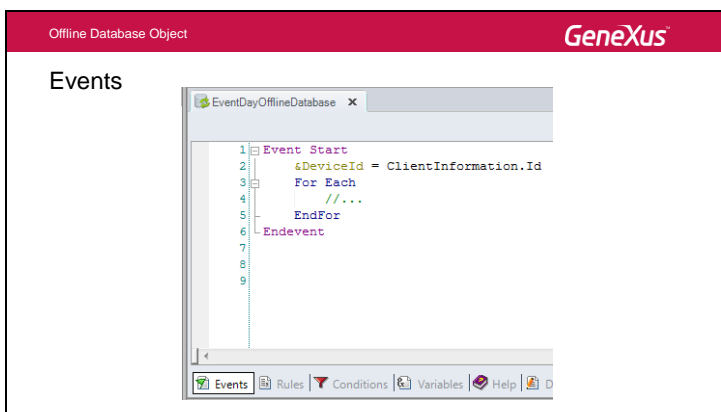
Right-clicking on the EventDay object and selecting the References option shows which objects are invoked by the dashboard. Also, we see that the EventPreferences business component is not invoked directly or indirectly.



When the application is installed on the device, the tables are created and the data is copied from the server, if there's an Internet connection.

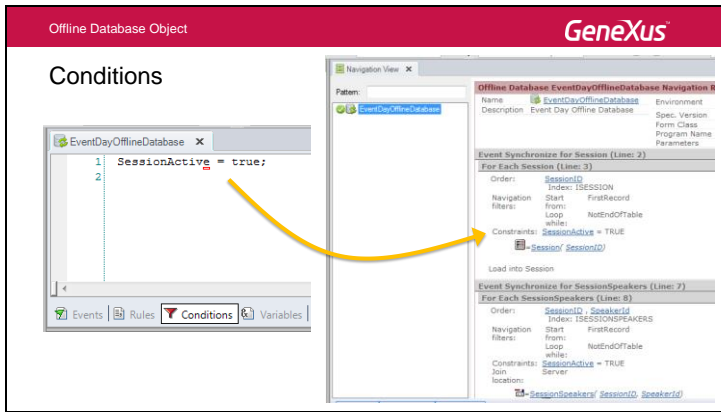
We return to the Offline Database object.

It is a simple object that only has events, conditions and properties.

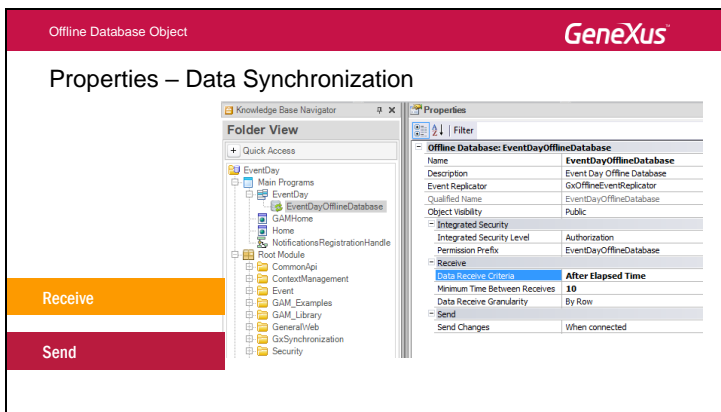


The only event that can be programmed is the Start event. This event is executed on the server, before every submission of data to the client for synchronization.

The Start event is designed to perform the initialization of variables and other procedures that must be run before synchronizing the tables.



The filters included in the Conditions tab are used as in any other GeneXus object. These filters are applied to the tables to know which data is sent to the device. They are global, so they are applied to all the corresponding tables. Conditions admit variables that can be loaded in the Start event.

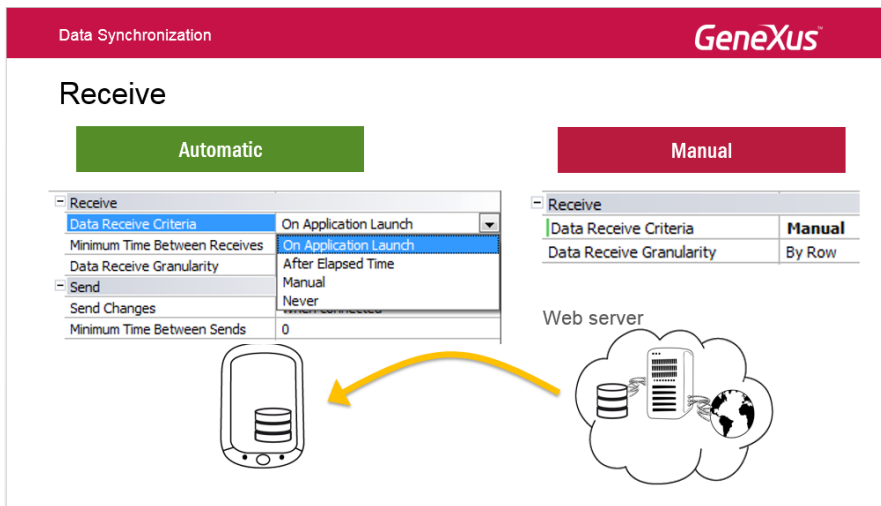


The OfflineDatabase object includes properties that can be used to define when to send or receive data to and from the server, respectively.

- Receive, which define how and when to receive a subset of data from the server.

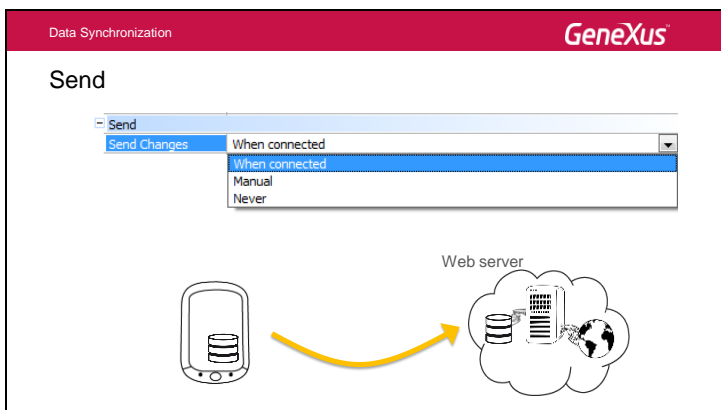
And in

- Send, which define how and when to send data from the device to the server.



When receiving data from the server, the synchronization process can be made in two ways:

- **Automatically:** it allows configuring the moment when the synchronization is made, which can be after opening the application for the first time, after a certain interval from the last time it was opened, always, at certain intervals, or never.
- **Manually:** the way in which data is received from the server has to be programmed using code.



The submission of data to the server can also be configured; it can be automatically made when the device gets connected to the Internet, it can be made manually depending on the programming, or never if no data is to be sent from the device to the centralized database.

In the event that the Receive and Send operations are manually made, the Synchronization API must be used to this end. This API is not found as an object in the SmartDevicesAPI folder; instead, it is part of the grammar.

Data Synchronization

GeneXus

Synchronization API

Properties

Offline Database: EventDayOfflineDatabase

Name

EventDayOfflineDatabase

Description

Event Day Offline Database

Event Replicator

GxOfflineEventReplicator

Qualified Name

EventDayOfflineDatabase

Object Visibility

Public

Integrated Security

Integrated Security Level

Authorization

Permission Prefix

EventDayOfflineDatabase

Security

Data Receive Criteria

Manual

Data Receive Granularity

By Row

Send

Send Changes

Manual

Manual Synchronization Only

Synchronization API

Receive

Send

```
Event 'Receive'  
  Synchronization.Receive()  
Endevent  
  
Event 'Send'  
  Synchronization.Send()  
Endevent
```


As an example, we can create an SD panel executed from the device, where the Send and Receive operations are programmed, as we can see in the image.

When data is sent to or received from the device, synchronization conflicts may occur because the application may be installed on several devices, and each one of them can insert data with the same identifiers but for different data.


Data Synchronization

GeneXus


Synchronization Conflicts



CountryId	CountryName
1	Uruguay
2	Brasil



CountryId	CountryName
1	Argentina
2	Paraguay



SpeakerId	SpeakerName	CountryId
1	Alejandro Cimas	2
2	Lucia Guedes	1

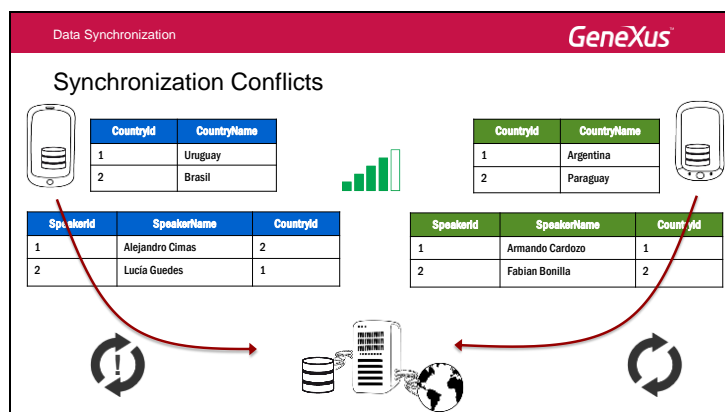
SpeakerId	SpeakerName	CountryId
1	Armando Cardozo	1
2	Fabian Bonilla	2

Let's see this in our example.

In the EventDay KB we have the Country transaction with the CountryId attributes and the Speaker transaction, which has the CountryId attribute as foreign key.



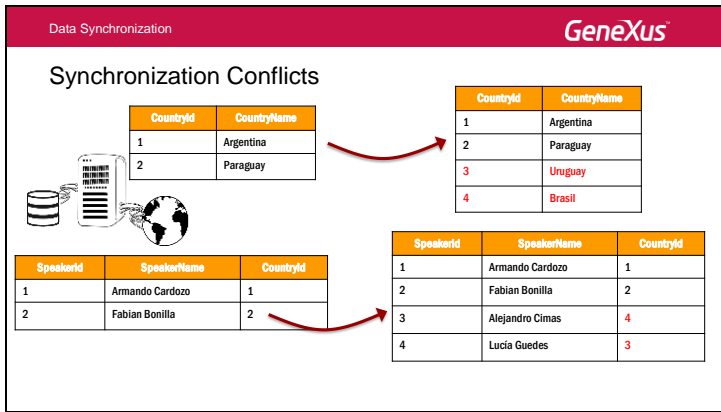
For example, here we can see that a device created the countries Uruguay with ID=1 and Brazil with ID=2, and that the countries Argentina with ID=1 and Paraguay with ID=2 are inserted in another device. Both devices are working offline, with no Internet connection, so all the data is saved in the local database of each device.



Once an Internet connection is obtained, one of the devices is synchronized with the server, sending all the operations made through Business Components over the local database.

When the second device tries to synchronize, a conflict occurs because the country key is repeated.

GeneXus solves the conflict because autonumbered keys are used.



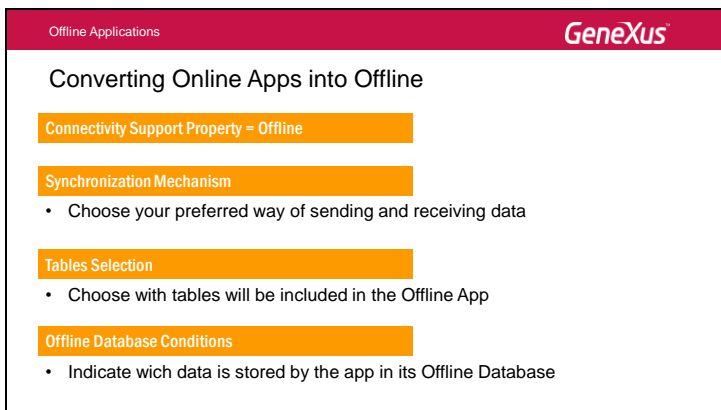
How does it do it? By solving the conflict in the server.

GeneXus generates a new key for the repeated countries, and updates the data on the server table with this newly generated ID.

Next, it updates the Speakers table so that the foreign keys match.

Lastly, it updates the data on the device, so that they correct keys are used in the device as well.

To sum up what we have seen, to convert an online application into an offline application, we need to follow these steps:



1. Set the Connectivity Support property to Offline
2. Select the synchronization mechanism
3. Indicate which tables will be created in the device
4. Add conditions to filter which records will be sent to the device.

If there is an Offline application that uses GAM, we need to take into account that credentials will always be saved on the server; therefore, the login operation can only be made online.

Offline Applications

GeneXus

Offline Apps + GAM

The credentials remain on the server

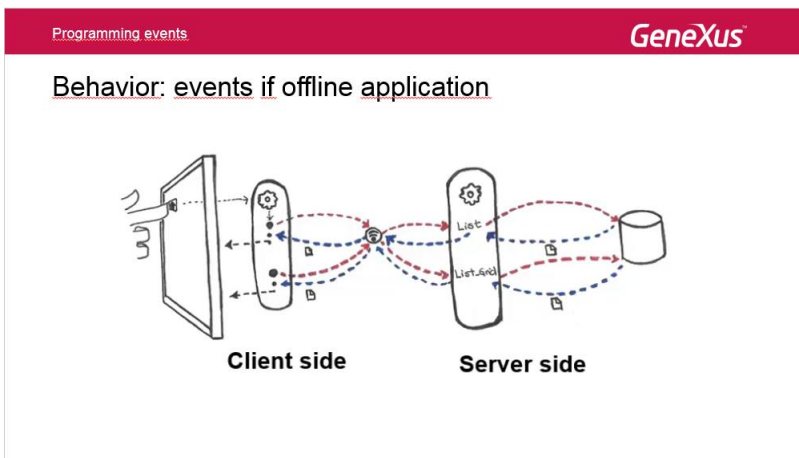
Login only Online

- Authenticated user is still authenticated when the app goes Offline

Only Authentication

Once logged in, users can work offline.

The programming of events in an offline application has certain characteristics which are different from online applications.



We will see the events of an offline application in another video.

For more information about offline applications, click on the link below:

<http://wiki.genexus.com/commwiki/servlet/hwiki?pageid?22228>

<http://wiki.genexus.com/commwiki/servlet/hwikibypageid?22228>

GeneXus X Evolution 3