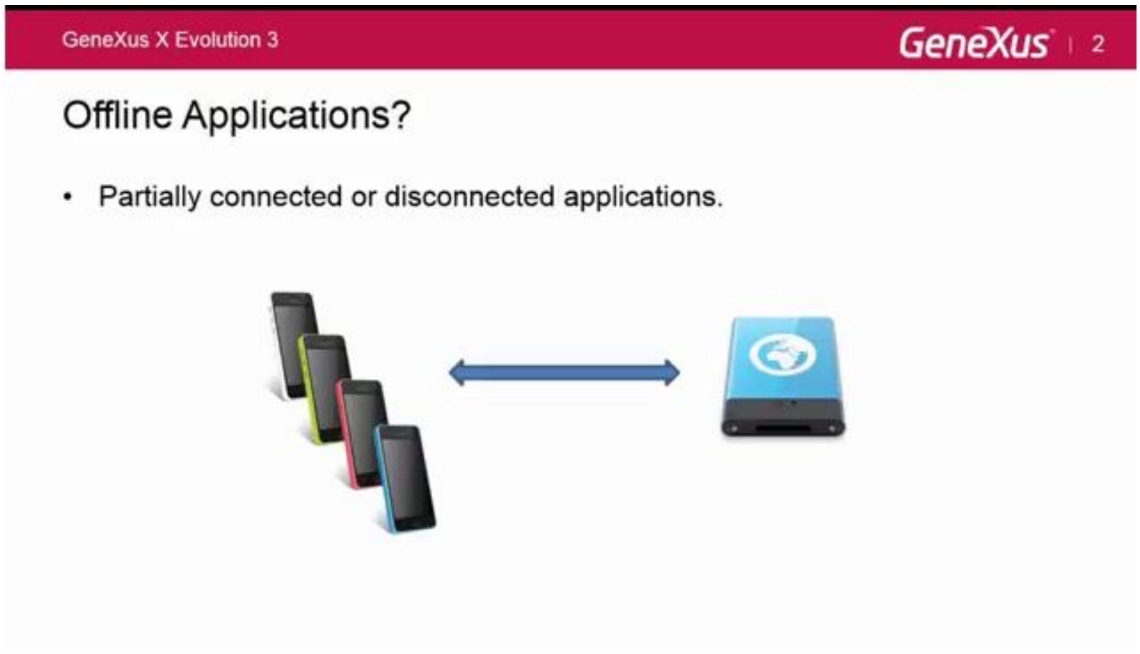
Offline Mode

Welcome!

My name is Martin Torrado and I work for Artech's support team. In this webinar, we will talk about offline SD applications for the new version of X Evolution 3.

So, let's start from the basics, that is: what are offline applications?



GeneXus X Evolution 3 GeneXus | 2

Offline Applications?

- Partially connected or disconnected applications.

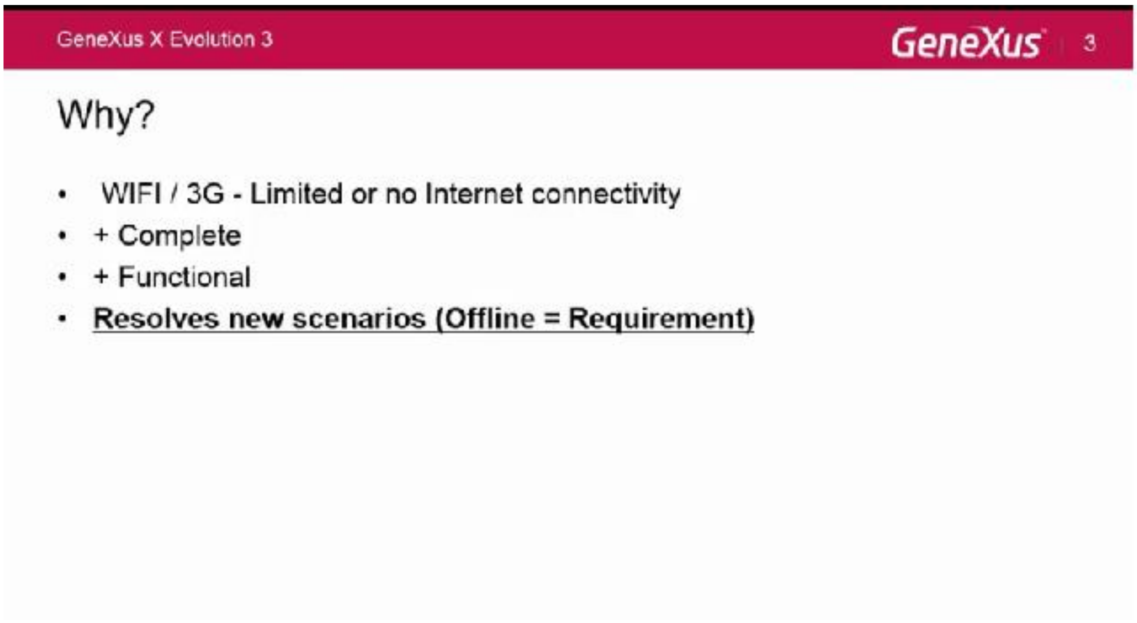
Offline applications are partially connected, autonomous applications.

And what does this mean?

This means that when I'm using my application, on my device connected to internet, when that connection is lost, I will still have my application running. And when I finally recover my connection to internet, a synchronization process with the centralized server's database will take place.

Therefore, we could say that these applications function at all times, regardless of whether we have a connection or not, and they access local data with the possibility of executing complex logic on the device's side.

Why do we have this type of applications?


GeneXus X Evolution 3 GeneXus 3

Why?

- WIFI / 3G - Limited or no Internet connectivity
- + Complete
- + Functional
- **Resolves new scenarios (Offline = Requirement)**

As a result, exactly of what we said, because we may not always have internet connections available to us. So, my application will continue to function even where I cannot get an internet connection.

We then say that our application are more complete and functional. And most of all –as I pointed out in underlined bold letters on the ppts as very important– because we may solve a wide range of new scenarios where offline applications, instead of something desired, become a system requirement.

And we have thousands of examples of this. For instance, sales force applications where my sales cannot depend on whether my device is connected to the Internet or not.

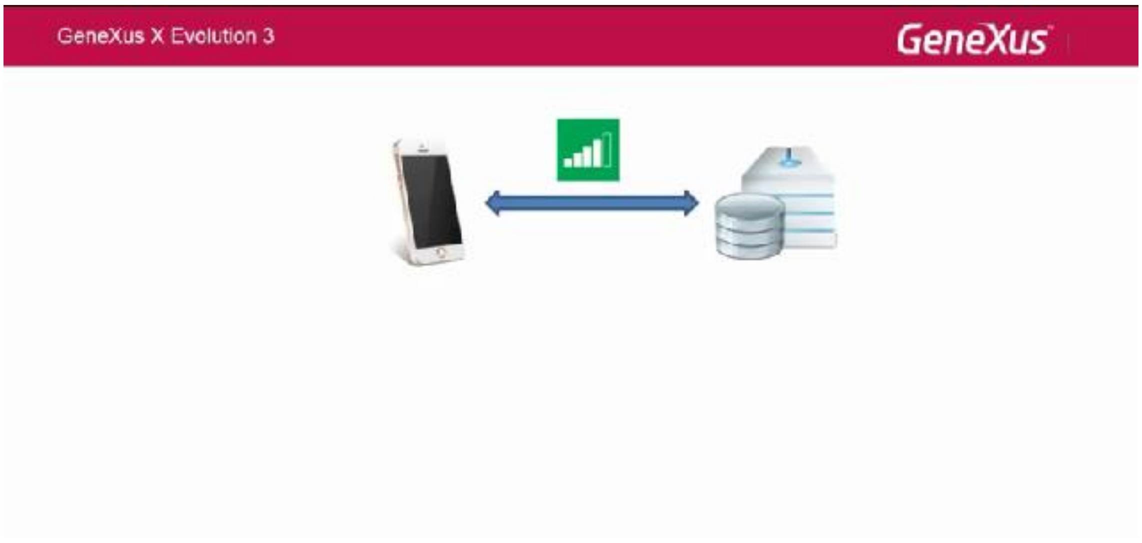
So far then, we have found out what offline applications are and why we use them.

Now I would like to refer to the Architecture…

GeneXus

Architecture

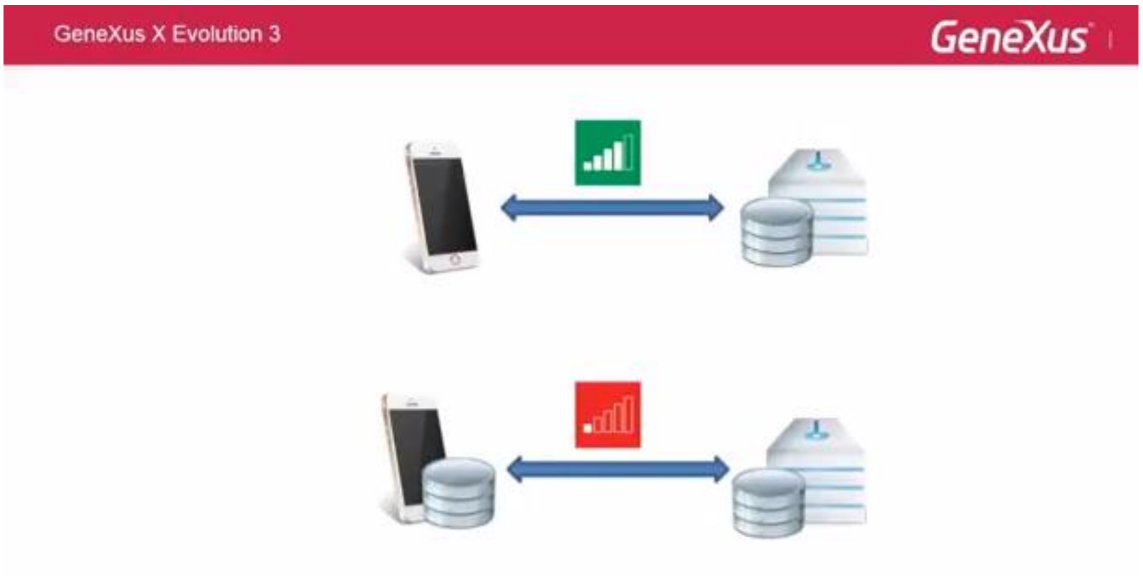
And the best way to do it is compare against a model we already know about: the model of architecture of SD applications in XEv2.



As you will recall, what we had up to the XEv2 was our Devices with an installed application that was connected to our centralized server’s database through a layer of rest services, to obtain data and execute “Insert, update, delete” on the server’s side.

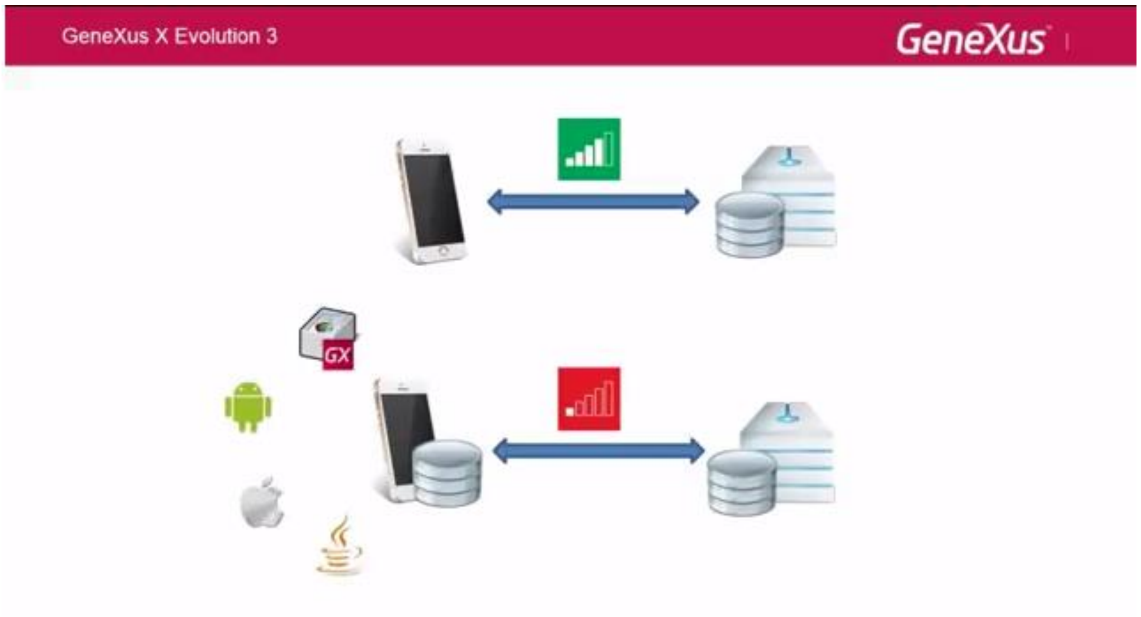
That was basically the architecture we had been working with up to the XEv2 version.

And though many things remain the same in the architecture of XEv3, we will now have a database on the client side.

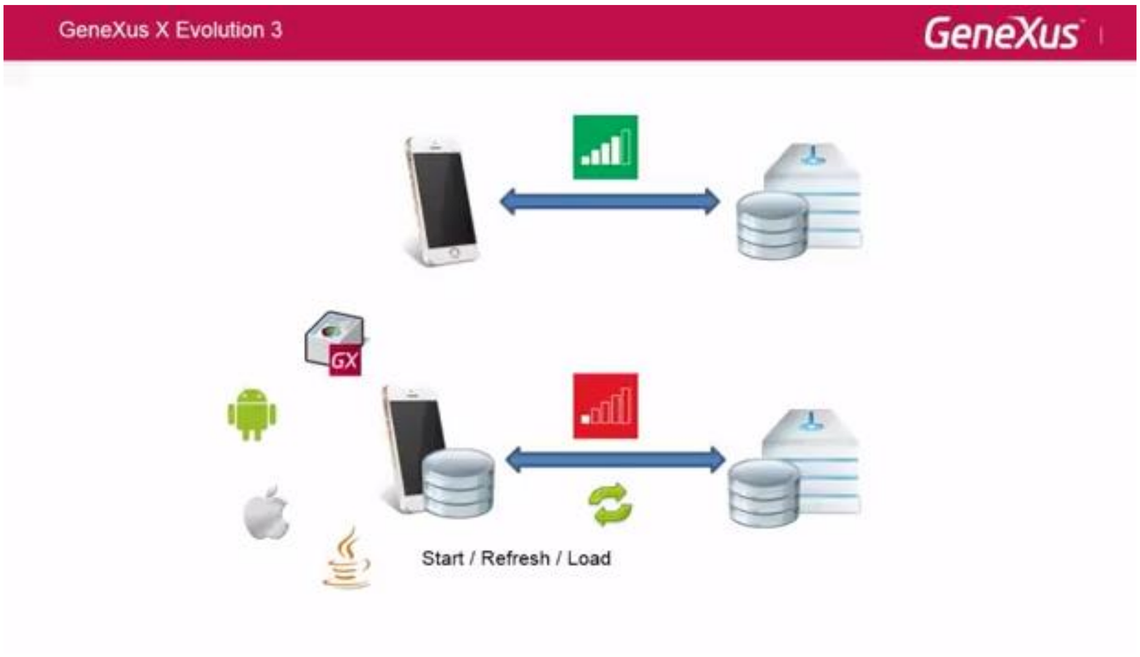


An sqlite database, already pre-installed in both Android and IOS, is a database engine that we have on the client side due to the fact that we will have more data processing requirements on the client side.

This means that a much more complex logic will be executed on the client side, and we will have to generate native code, which will be JAVA in Android and Objective-C in IOS.



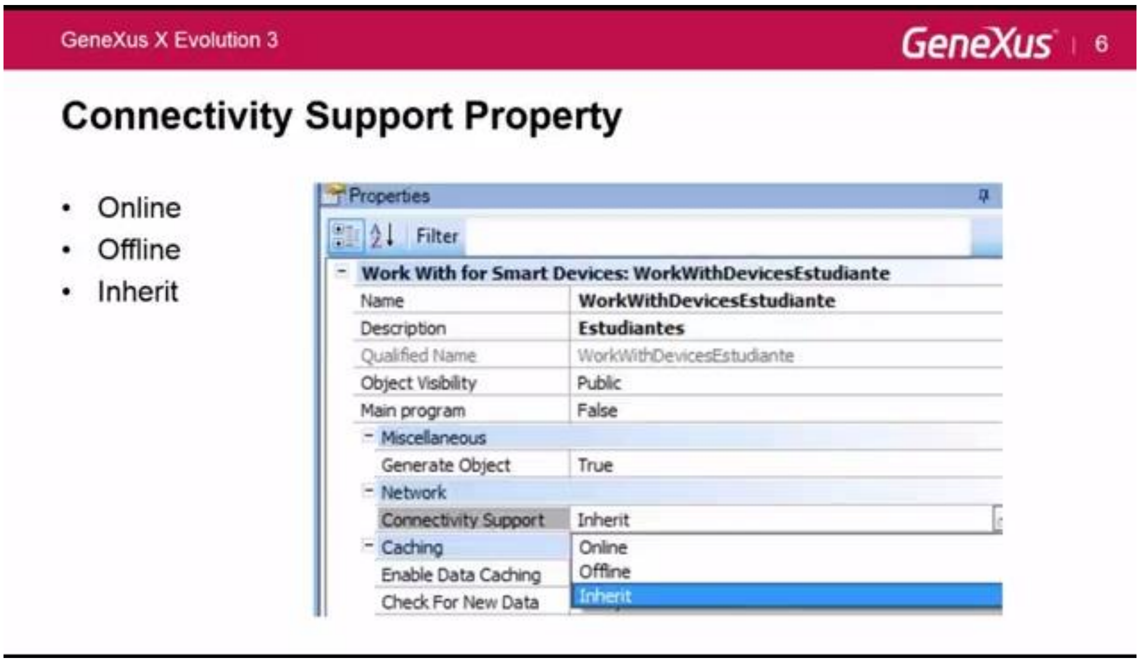
We should also mention the Start / Refresh / Load events, which previously -in online architecture- were executed on the server side, are now executed on the client side by accessing local data.



User events continue to be interpreted, with the exception that when offline applications call an offline object, they will do it locally without the need to go to the server to execute it.

Another outstanding aspect in this architecture is that both the client database and the server database will have to remain consistent with one another, through a process called "synchronization process" which we will see in detail further ahead.

Now let's see how we do this in GeneXus.



Connectivity Support Property

- Online
- Offline
- Inherit

Properties

Filter

Work With for Smart Devices: WorkWithDevicesEstudiante	
Name	WorkWithDevicesEstudiante
Description	Estudiantes
Qualified Name	WorkWithDevicesEstudiante
Object Visibility	Public
Main program	False
Miscellaneous	
Generate Object	True
Network	
Connectivity Support	Inherit
Caching	Online
Enable Data Caching	Offline
Check For New Data	Inherit

We have a property called Connectivity Support, available for all SD objects, which includes three options:

- Online
- Offline
- Inherit

What do these three options imply?

Online is the default option for SD main objects, and the way we have been working so far, including the XEv2, where our application is fully connected. The Inherit property is included by default in all SD objects that are not main objects. This means that they will inherit the behavior of the parent calling it. And the offline property is the one we are most interested in today.

What happens when I set an object's property with connectivity support offline?

Suppose we have an application for an event, and a dashboard (set up as main) for which I set up the property connectivity support offline.

What happens immediately afterwards? The next time I execute or compile, a new object called Offline DataBase Object will be created.

GeneXus

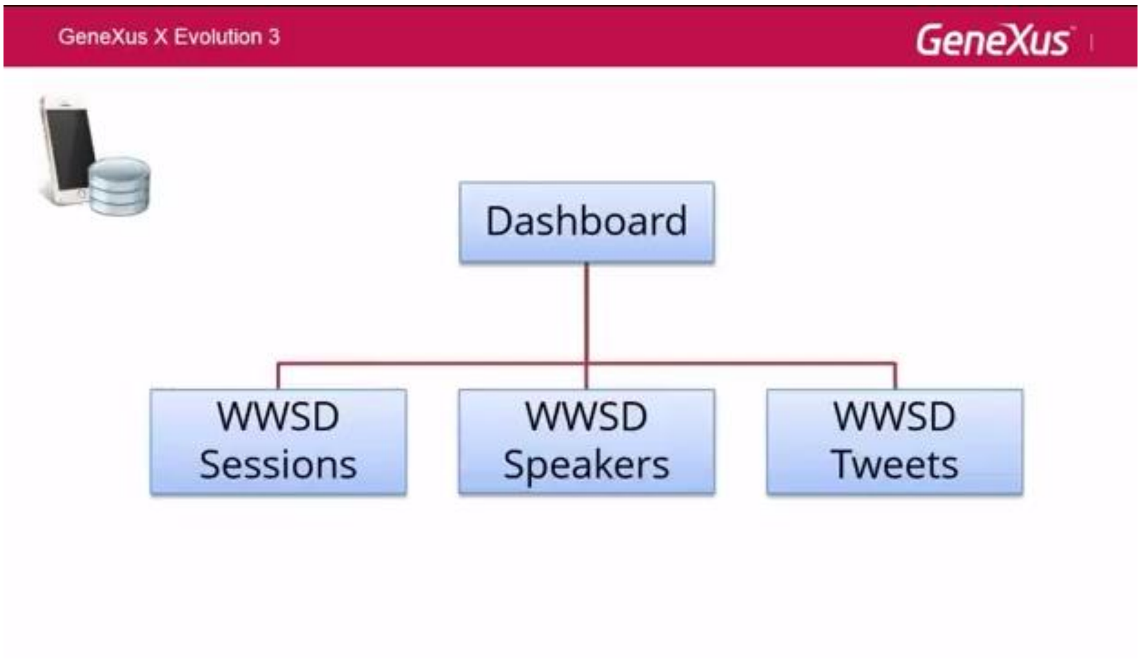
Local database generation

Offline DataBase Object

It will determine the tables that go to the client's database, and also the data taken for synchronization, because it is obvious that the local database will not be the same as the server database but rather a reduced version of it.

So, how do we determine the tables that will be taken to a client's database?

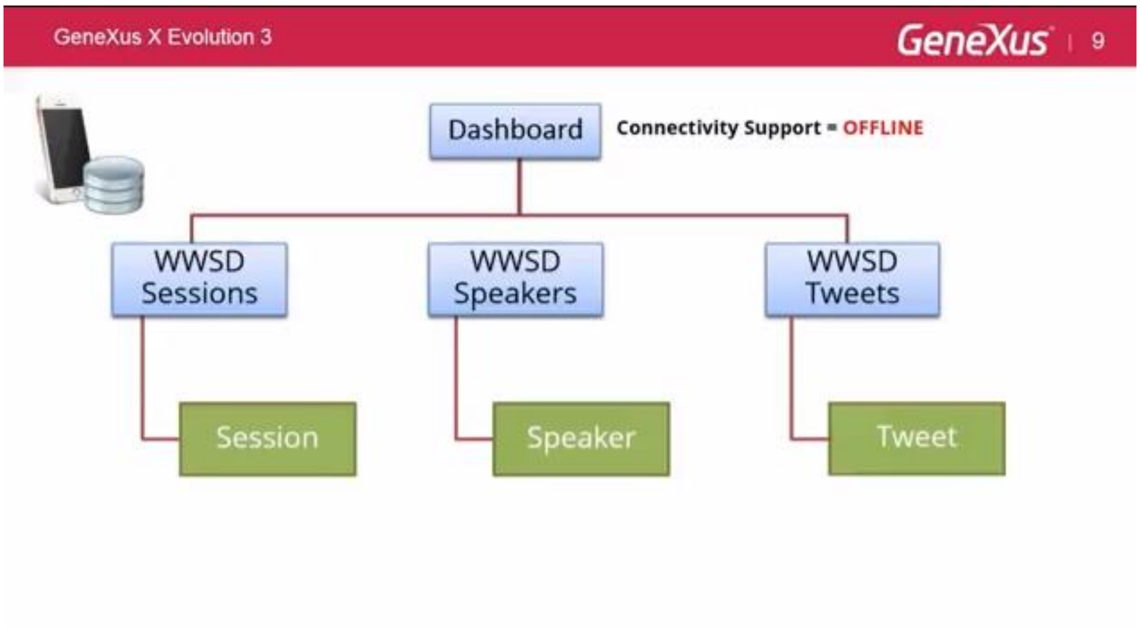
I brought this example to make the concept clear. And in line with what we just saw, it is an application for an event.



I have a dashboard calling a Work with sessions, a Work with speakers, and a Work with tweets.

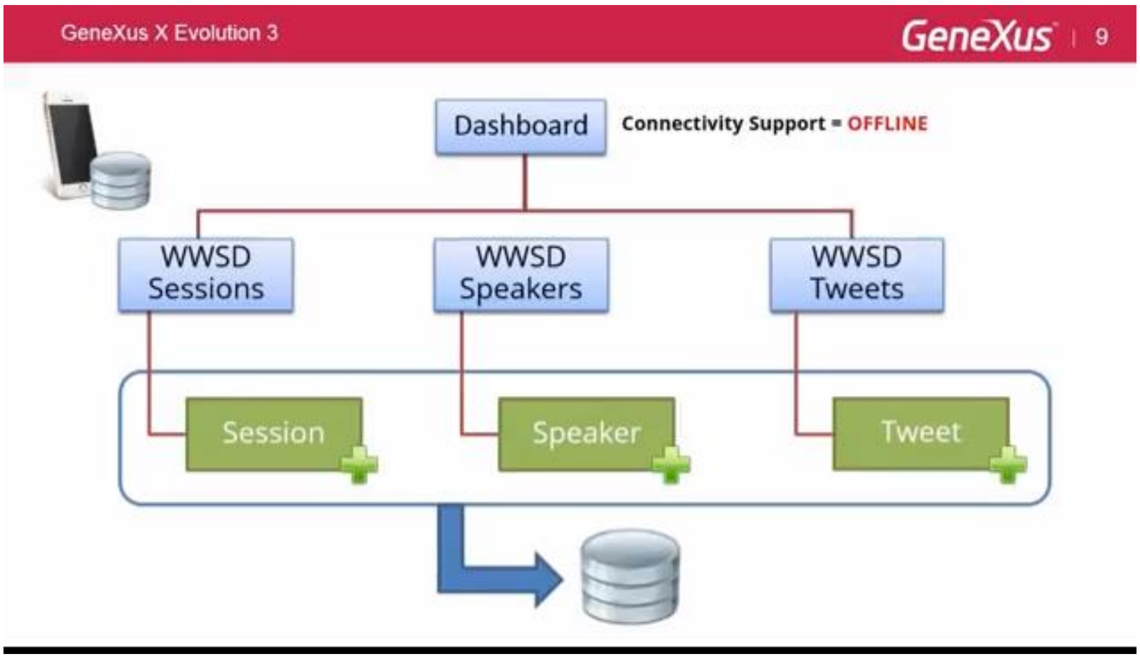
So, what tables will end up in the device's internal database?

For my dashboard, I define Connectivity Support offline. Since, as we said, my dashboard is an SD main object that has the property connectivity support online by default, I change it to offline. I indicate that, as of now, I want the application to function without connectivity.



As we know, we have the object Work with Sessions, speakers and tweets. And the property that these three panels will bring by default will be Inherit, so the behavior of the calling parent will be inherited, in this case the dashboard with the offline property.

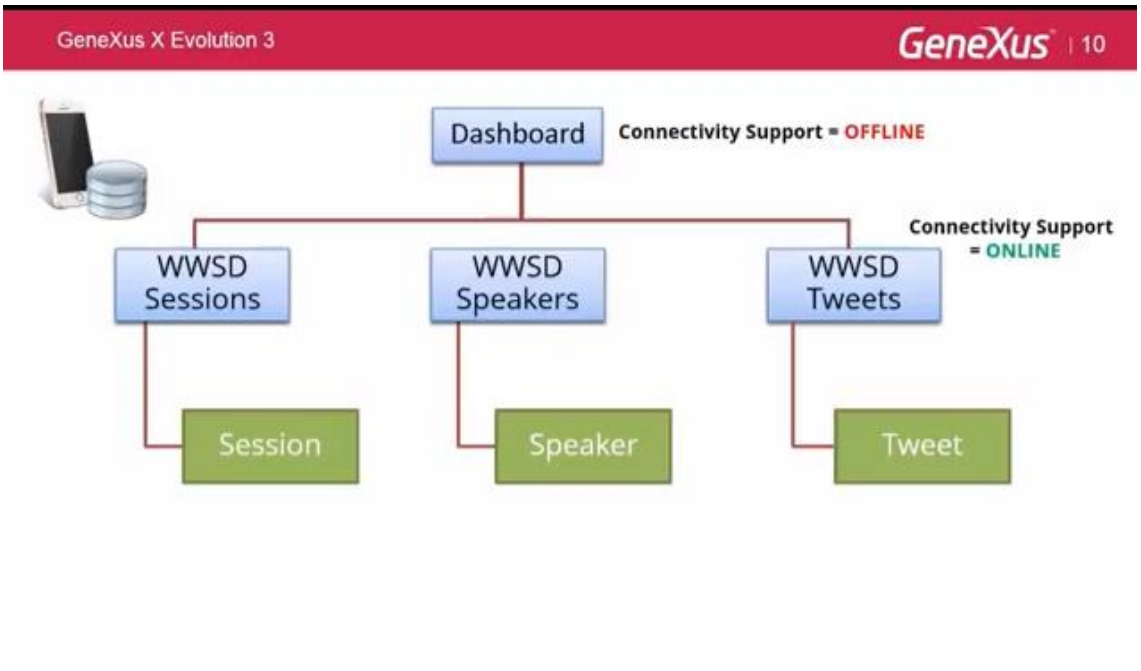
Therefore, they will inherit that offline behavior …



… and they will be added to the tables they access and to the device's database, that is to say that the addition will include the sessions, speaker and tweets tables.

And what if I want tweets to only be viewed with internet connectivity so that users always view updated tweets with internet connection?

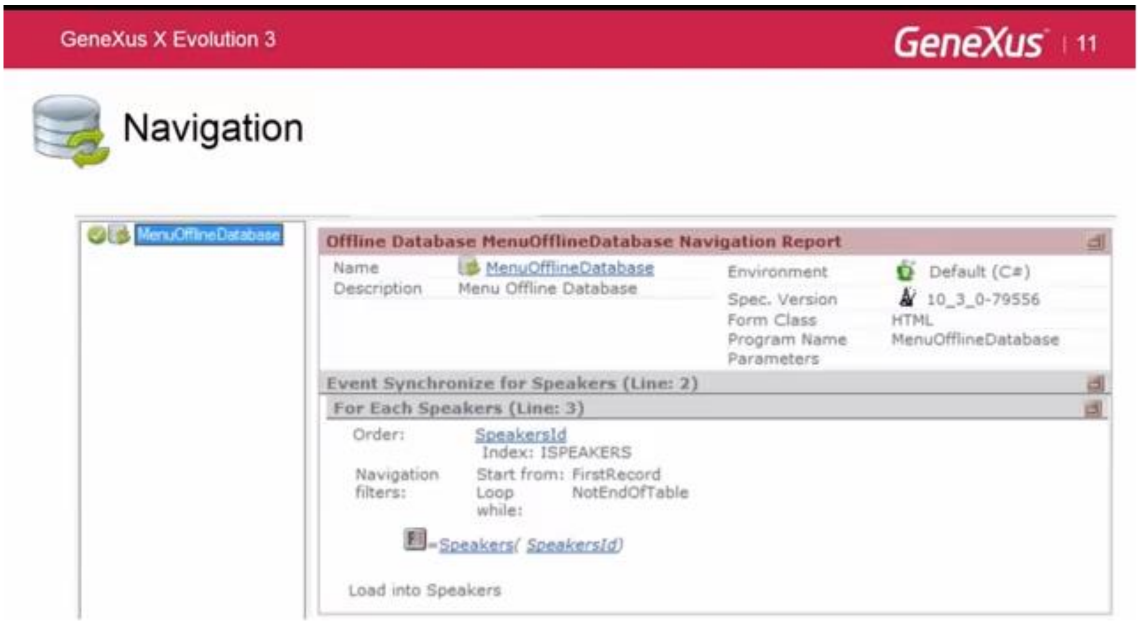
In that case then I will need to act on the connectivity support property, and set online connectivity support for the Work with SD panel.



By doing this I am determining that tweets will only be viewed with connectivity. So the tables taken to the device no will be session and speaker only.

We should also point out here that when a table has a Foreign Key to another table, the table referenced will also be taken to the device.

So, let's now see the navigation of this Database offline object.


GeneXus X Evolution 3 | GeneXus | 11

Navigation

MenuOfflineDatabase

Offline Database MenuOfflineDatabase Navigation Report

Name	MenuOfflineDatabase	Environment	Default (C#)
Description	Menu Offline Database	Spec. Version	10_3_0-79556
		Form Class	HTML
		Program Name	MenuOfflineDatabase
		Parameters	

Event Synchronize for Speakers (Line: 2)

For Each Speakers (Line: 3)

```
Order:        SpeakersId
               Index: ISPEAKERS
Navigation    Start from: FirstRecord
filters:      Loop        NotEndOfTable
              while:

        =Speakers( SpeakersId)

Load into Speakers
```

As you can see, there is a Synchronize event for each table involved. So for each of those events, which are basically For eachs to the table, we will see the conditions applicable.

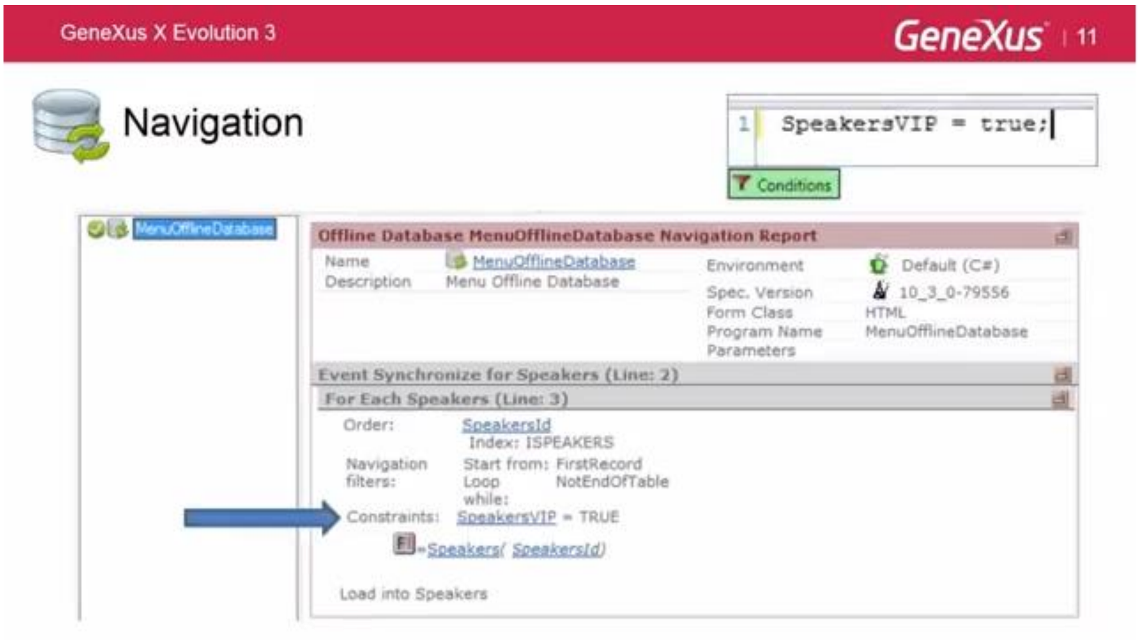
Conditions are used like any other GeneXus object, applied to tables in order to know which data is taken to the device.

And why are these conditions applied?

For one thing, there may be security reasons where I do not want a user to view all records, and also capacity reasons relative to the device's memory. I cannot take all the records of the server's database to my device because there may be thousands of records and my device's capacity may be exceeded.

Due to those two reasons, we will be applying filters. As we saw in previous ppts, we can filter by tables, differentiating or discriminating the tables we want taken from the server's database to the device's database. But, as we will see now, we can also filter by records.

And how do we do this?


GeneXus X Evolution 3 GeneXus | 11

Navigation

```
SpeakersVIP = true;
```

Offline Database MenuOfflineDatabase Navigation Report

Name	MenuOfflineDatabase	Environment	Default (C#)
Description	Menu Offline Database	Spec. Version	10_3_0-79556
		Form Class	HTML
		Program Name	MenuOfflineDatabase
		Parameters	

Event Synchronize for Speakers (Line: 2)
For Each Speakers (Line: 3)

```
Order:        SpeakersId
               Index: ISPEAKERS
Navigation    Start from: FirstRecord
filters:      Loop        NotEndOfTable
              while:
Constraints:  SpeakersVIP = TRUE
     =Speakers( SpeakersId)

Load into Speakers
```

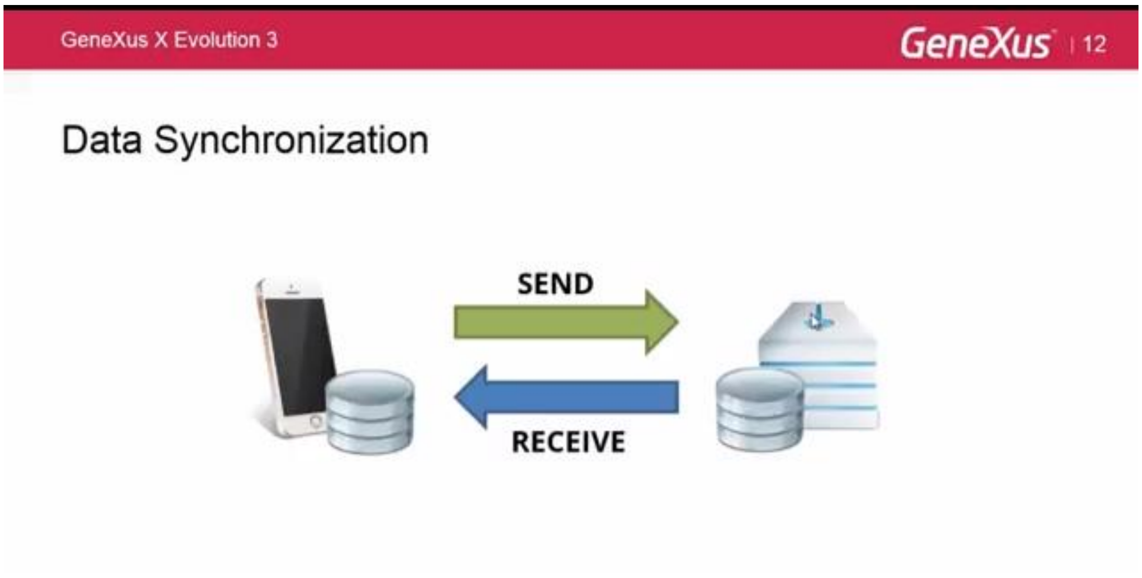
We add a condition. Suppose, in the same example we had, that I only want speakers in my event that I consider important to be taken to the device.

That is: VIP speakers = true

So, on the CONDITIONS tab I add that condition. This will add a Constraint to the object's navigation that the For each will continue doing on the speakers table. And also, I add another Constraint where VIP speakers = true.

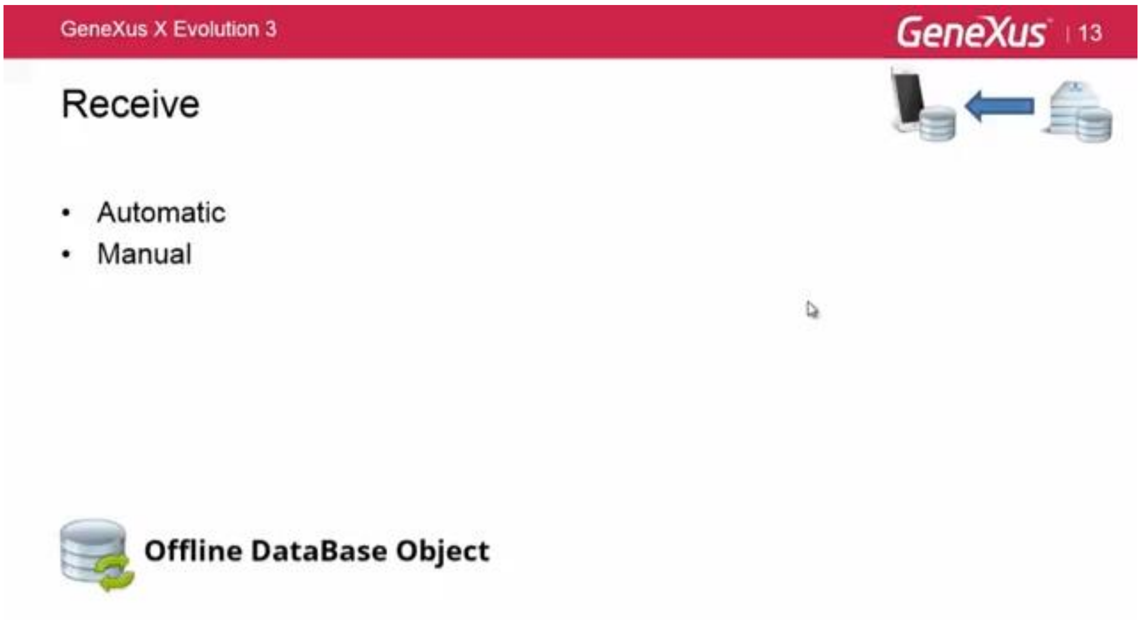
Something important to point out is that we do not have reorganizations, and the database does a create every time we modify the data model.

Now let's see something about data synchronization.



This synchronization will have a process for sending data from our device database to the server, and a process that receives data from the server to the device database, that is, the client database.

So, starting with the data reception process, there are two ways for synchronization: Automatic and Manual.

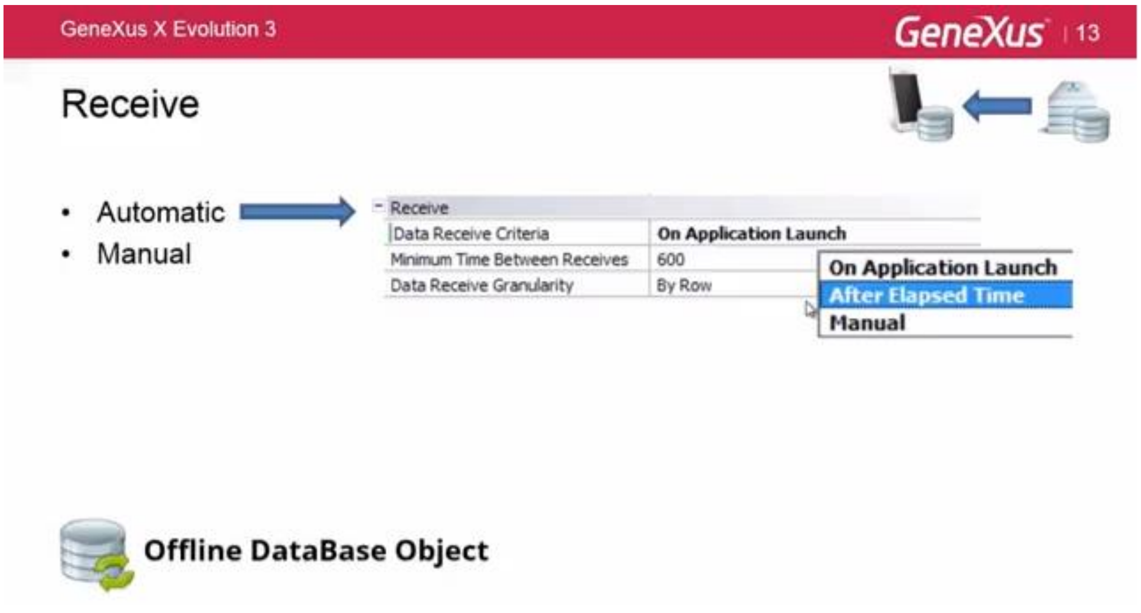

GeneXus X Evolution 3 | GeneXus | 13

Receive

- Automatic
- Manual

Offline DataBase Object

In automatic synchronization we have the following properties at the level of the offline Database object.


GeneXus X Evolution 3 | GeneXus | 13

Receive

- Automatic
- Manual

Receive	
Data Receive Criteria	On Application Launch
Minimum Time Between Receives	600
Data Receive Granularity	By Row

Offline DataBase Object

The Data Receive Criteria property, by default, will come in an On Application Launch, but there is also the After Elapsed Time property. And there is the manual way which we will see further ahead.

What happens when I set Data Receive Criteria with the On Application Launch property?

What will happen is that I will be receiving the data every time that the application does a Launch, that is, every time that I pick up the application when the minimum time between receives has elapsed, which is the second property, that is currently set at 600 seconds – a 10 minute period will pass. When I open the application, if 10 minutes have elapsed, then the application will be data Receive.

What happens when I set the Data Receive Criteria property in After Elapsed Time?

This property is supplementary to the other one, so a data receive will be done every time that 600 seconds have elapsed from the last time that I received data, so it supplements the other property.

And this data receive may take place in either one of two ways, which is the last property you see on screen, the Data Receive Granularity. It may be received by Row or by Table.

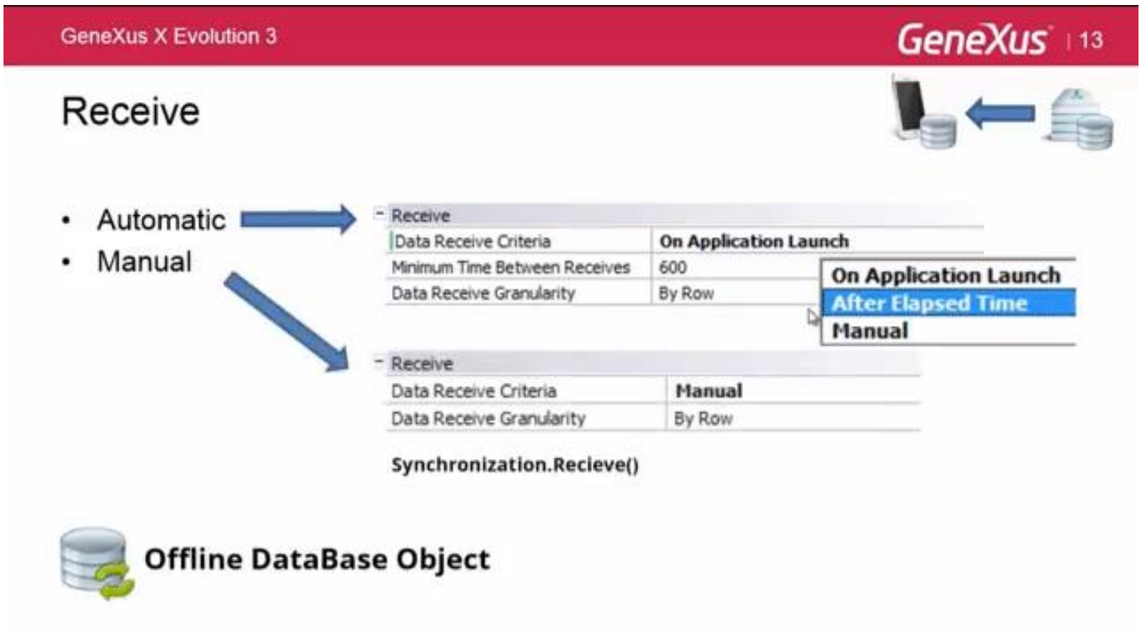
And what is the difference between them?

By Row, I will receive in the client database only the server data modified, that is the modified rows. By Table, I will be receiving the whole table every time that there is a change made. The table will be deleted on the client side for the whole table to be received again.

This implies advantages as well as disadvantages. With by Row the advantage is that the data transfer from server to client is small, because only modified data is sent. The disadvantage in this is that there is much processing on the server's side, since it has to search for the data that was modified.

And there is the opposite situation in by Table. There is little processing on the server's side because the whole table is sent upon every change made, but there is a lot of data transferred from server to client.

And, as we mentioned, there is also the manual property.


GeneXus X Evolution 3

GeneXus | 13

Receive

- Automatic
- Manual

Receive	
Data Receive Criteria	On Application Launch
Minimum Time Between Receives	600
Data Receive Granularity	By Row

Receive	
Data Receive Criteria	Manual
Data Receive Granularity	By Row

Synchronization.Recieve()

Offline DataBase Object

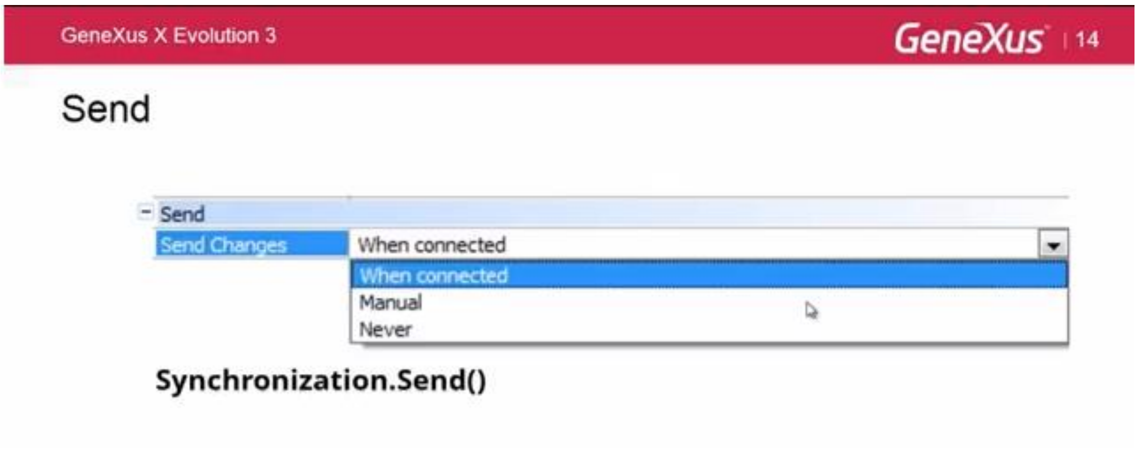
What will we decide in the Data Receive?

When we set up the property as Manual, the second property we had before, called minimum time between receive, disappears because the waiting time between receives does not make any more sense, since I will be executing them manually.

How is this done? As shown on screen, with Synchronization.Receive.

When I set the property as automatic, that is, with Application Launch or After Elapsed Time, there is also the possibility of using manual synchronization. I will be able to use, also, the Synchronization.Receive in user events, some buttons, etc.

This refers to receiving data.


GeneXus X Evolution 3 GeneXus | 14

Send

Send
Send Changes: When connected

Synchronization.Send()

Now let's see the Data Send, to send data from the local Database from Devices to the server database. To do this we have the property at the offline database object level, called Send Changes, which includes three options:

The default option is When Connected, which sends data as soon as I get my device connected. I work in offline mode, and when my connection occurs, that data is sent to the server automatically.

There is also the Manual option which validates the data receive. Using the Synchronization.Send method I will send the client data, when I consider appropriate at the programming level, to the server. I could have the When Connected option set up and use the manual method of my programming anyway with this Synchronization.Send method.

And the third option is Never, for when we don't send client data to the server. A case where this property is used is when we have a personal finance application that includes sensitive data that I do not want on the server.

Before we finish this presentation I would like to mention two things.

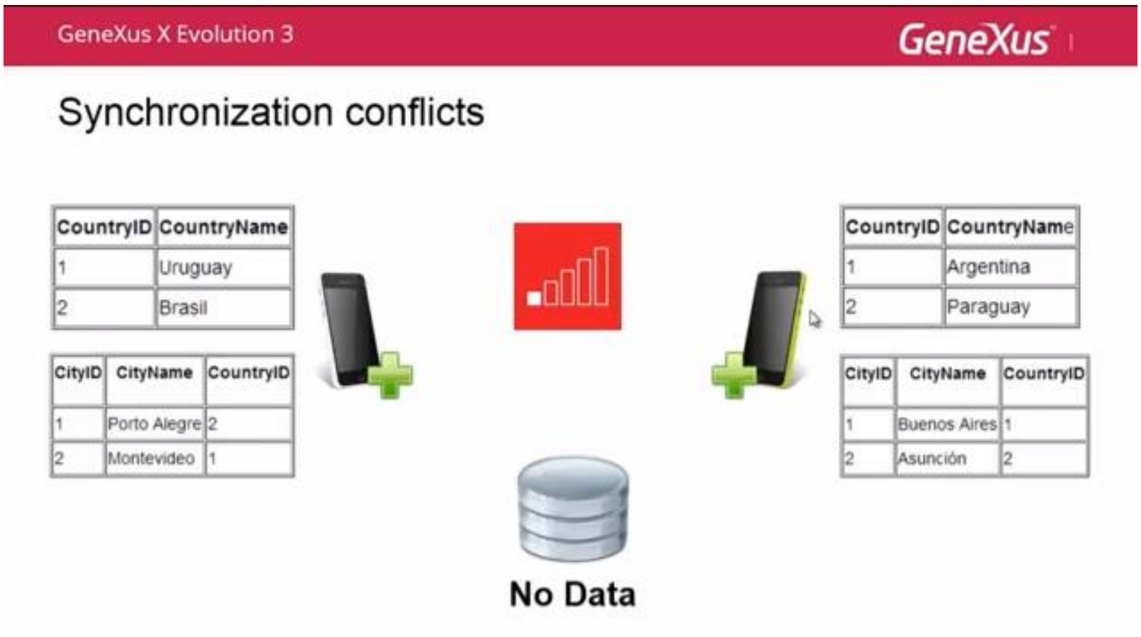
One is that when we create an offline application in GeneXus we install it on the Device, and a table called GXPendentEvent will be created, where all modifications made offline will be saved, in the same order in which they take place. When we obtain internet connection, those modifications are replicated in the same order in the server database, which makes the databases consistent with one another.

Additionally, we will have associated to each change, a state which we will be accessing through the Synchronization Event Api – an API that we make available to check possible errors in sending data and to make the necessary decisions in each case.

So we could say that this API enables us to make decisions and manage errors relative to synchronization.

The other thing I would like to mention is that everything to be synchronized in my offline applications must be done with business component, and the News and For eachs are not considered.

Having made clear these two aspects, we can now move on to consider conflicts in synchronizations.



Synchronization conflicts

CountryID	CountryName
1	Uruguay
2	Brasil

CityID	CityName	CountryID
1	Porto Alegre	2
2	Montevideo	1

CountryID	CountryName
1	Argentina
2	Paraguay

CityID	CityName	CountryID
1	Buenos Aires	1
2	Asunción	2

No Data

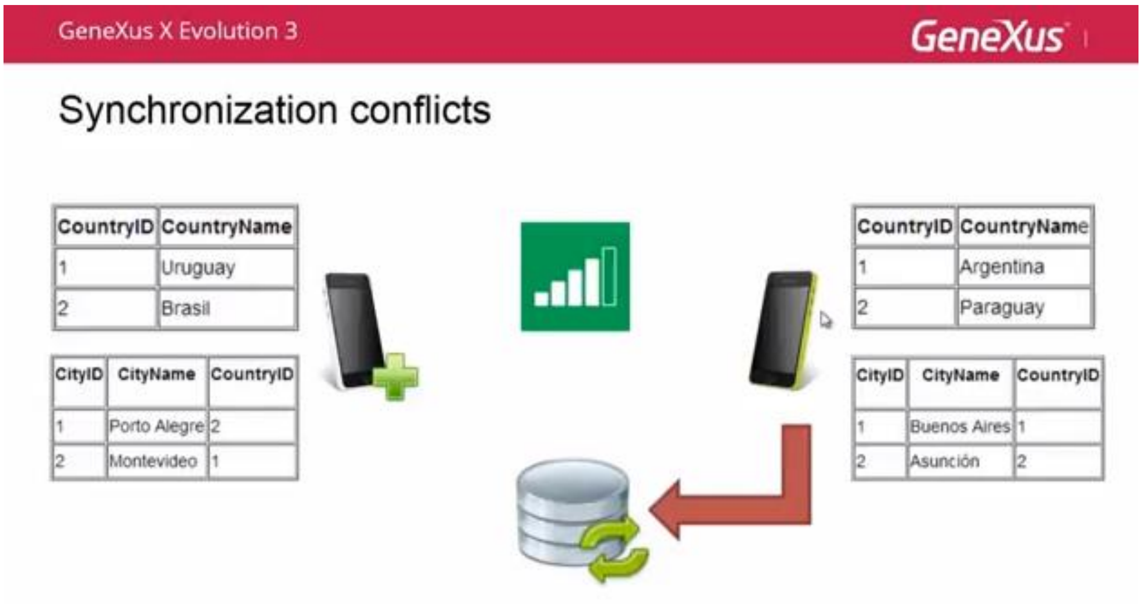
We will consider a particular case where we have two devices with the same application installed, both disconnected from the internet for the moment, and a centralized database in the cloud with no data to begin with.

So, Device1 does an insert of country Uruguay and of country Brazil with their corresponding cities - Porto Alegre and Montevideo – in the country tables and in the city tables.

We must point out that both the CountryID and CityId keys are autonumbered. This is important in this particular case where we will manage conflicts.

As we said, Device1 does that insert of Uruguay-Brazil, and Device2 does an insert of Argentina and Paraguay in the same tables, also with tables CountryId and CityId autonumbered.

What happens when those two devices are connected to the internet and must be synchronized with the centralized server database?



GeneXus X Evolution 3

GeneXus

Synchronization conflicts

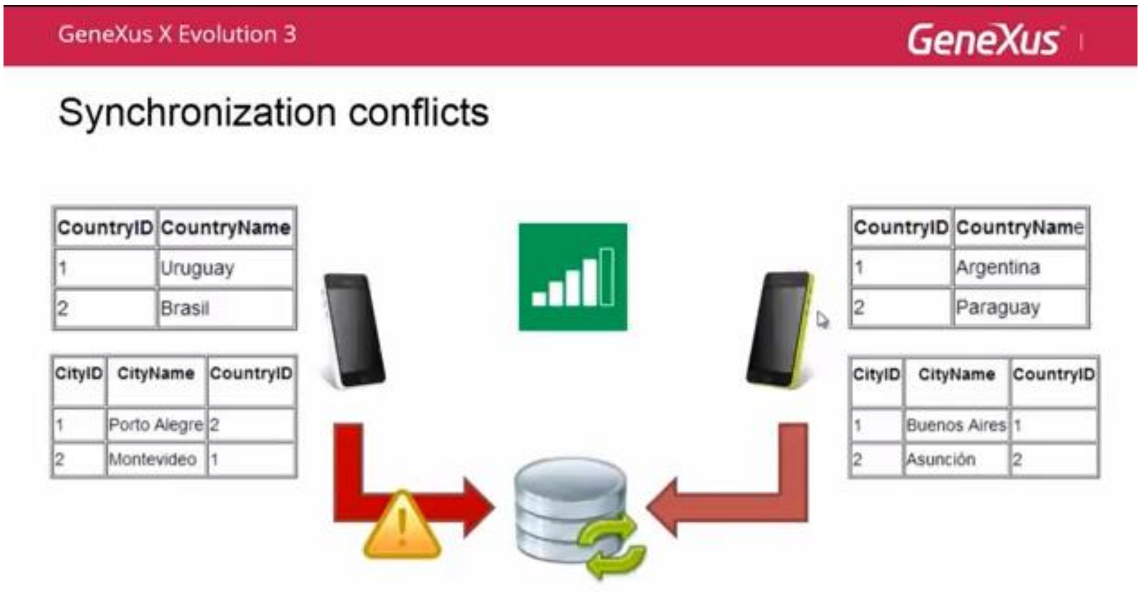
CountryID	CountryName
1	Uruguay
2	Brasil

CityID	CityName	CountryID
1	Porto Alegre	2
2	Montevideo	1

CountryID	CountryName
1	Argentina
2	Paraguay

CityID	CityName	CountryID
1	Buenos Aires	1
2	Asunción	2

Suppose that Device2 is the first one connected and synchronizes its data with the server. Then, it will send "Argentina-Paraguay" "Buenos Aires-Asuncion" to the server's tables.



GeneXus X Evolution 3

GeneXus

Synchronization conflicts

CountryID	CountryName
1	Uruguay
2	Brasil

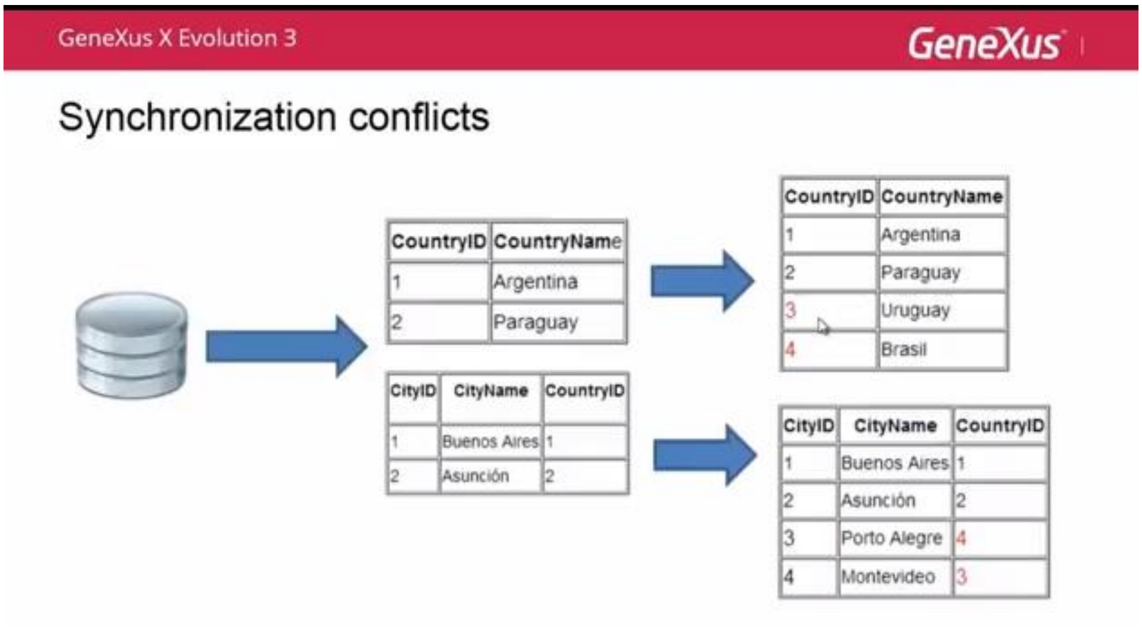
CityID	CityName	CountryID
1	Porto Alegre	2
2	Montevideo	1

CountryID	CountryName
1	Argentina
2	Paraguay

CityID	CityName	CountryID
1	Buenos Aires	1
2	Asunción	2

When Device1 tries to synchronize there will be a conflict because it will be trying an insert in the same tables with the same ID. Therefore, when it calls for Uruguay with CountryID1, it will find that CountryID1 is already Argentina, as entered by Device2.

So, how is that conflict solved?



GeneXus X Evolution 3

Synchronization conflicts

CountryID	CountryName
1	Argentina
2	Paraguay

CityID	CityName	CountryID
1	Buenos Aires	1
2	Asunción	2

CountryID	CountryName
1	Argentina
2	Paraguay
3	Uruguay
4	Brasil

CityID	CityName	CountryID
1	Buenos Aires	1
2	Asunción	2
3	Porto Alegre	4
4	Montevideo	3

The server will solve this automatically. Since the keys are autonumbered, it continues adding those numbers, taking Uruguay and Brazil and assigning the value following the last one existent. Because Uruguay 1 and Paraguay 2 already existed, it will change the ID of Uruguay to make it Uruguay 3, and the same applies to Paraguay 4.

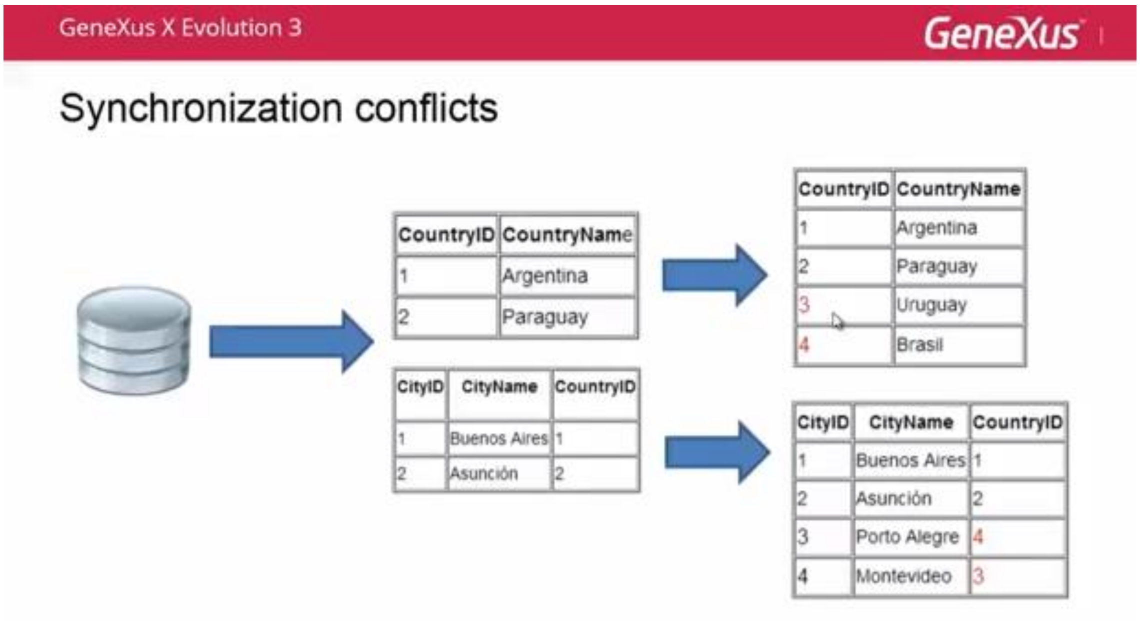
The interesting thing here is that modifications may also be done in cascade, that is: Uruguay with ID3 and Brazil with ID4 are also modified on the City table. Because it has foreign key, they are modified there for consistency.

And what about the data I had on my device (Uruguay1 and Brazil2)?

What I does here is a backwards modification. After the modification on the server is ready and consistent, it makes the modification in Devices, making the whole environment consistent, with the final values you now have on screen.

This is how conflicts with autonumbered keys are solved.

Now we will go into some considerations regarding Prototyping and GAM



GeneXus X Evolution 3

GeneXus

Synchronization conflicts

CountryID	CountryName
1	Argentina
2	Paraguay

CityID	CityName	CountryID
1	Buenos Aires	1
2	Asunción	2

CountryID	CountryName
1	Argentina
2	Paraguay
3	Uruguay
4	Brasil

CityID	CityName	CountryID
1	Buenos Aires	1
2	Asunción	2
3	Porto Alegre	4
4	Montevideo	3

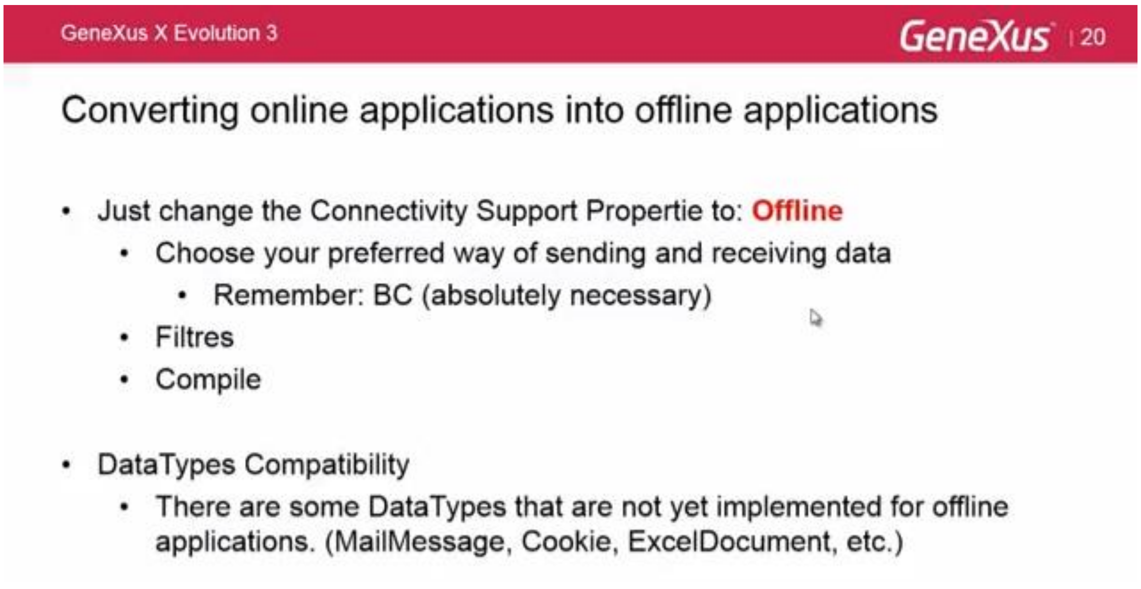
In regards to prototyping, I would like to say that KBN will not be available for testing offline applications.

Since the KBN only interprets metadata, and we will be generating our offline applications and complex logic which imply the generation of programs in native code -JAVA in Android and Objective-C in IOS–, the KBN will not be applicable for executing this type of logic and therefore not applicable for testing offline applications.

We will then have to compile and execute those applications on the devices.

Regarding the GAM, I wanted to mention two things: one is that credentials are kept on the server due to security reasons. We do not take user credentials to device databases. And as a result, the other thing to mention is that the login is done online only. I will only login to my application when I am connected to the internet. In third place we could say that user authentication may be done only online, but if we lose the connection after the authentication takes place, we may continue to access our applications screens offline.

And one last thing to mention is how we convert the applications we used to work with in XEv2 (online applications) into offline applications in XEv3.


GeneXus X Evolution 3 GeneXus | 20

Converting online applications into offline applications

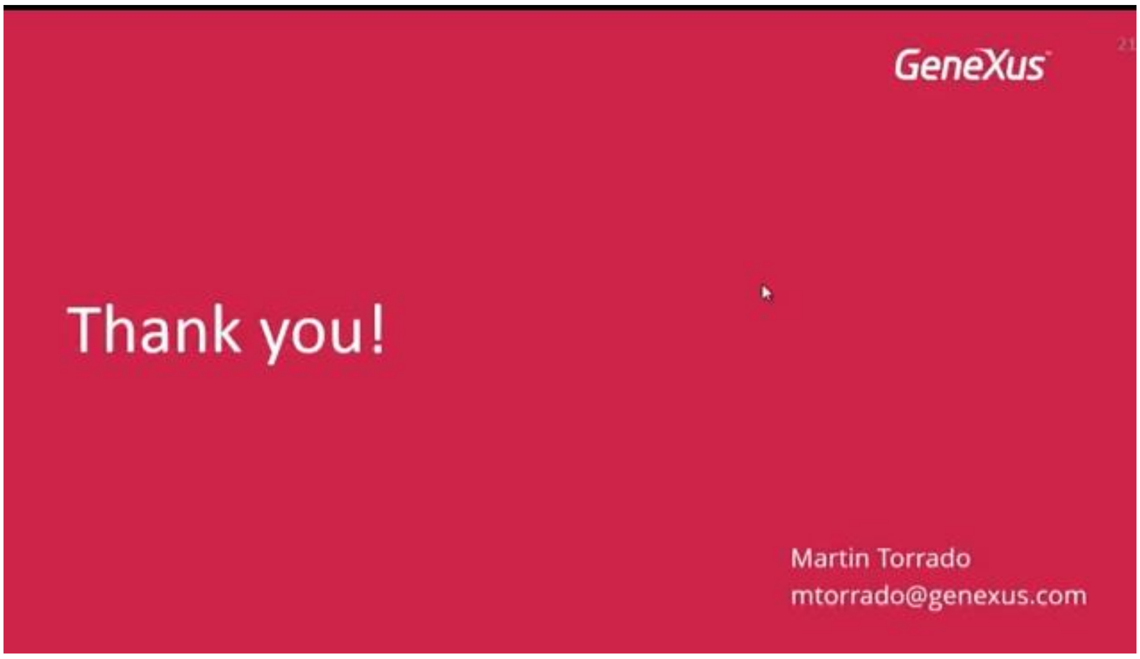
- Just change the Connectivity Support Propertie to: **Offline**
  - Choose your preferred way of sending and receiving data
    - Remember: BC (absolutely necessary)
  - Filtres
  - Compile
- DataTypes Compatibility
  - There are some DataTypes that are not yet implemented for offline applications. (MailMessage, Cookie, ExcelDocument, etc.)

As we mentioned before, what we must do is change the connectivity support property to offline. By default, this property is set as online. After we change it to offline we will select the way in which we want the data to be sent and received. As we saw in previous screens, we have the options of automatic or manual, and whether it is by row or by table. We have to set up those options and also remember that is necessary for all changes we want sent to the server database must be done with Business Component.

And bearing that in mind, also apply filters. As you may recall, we can decide which tables we want sent to the device's database, and from those tables, which records we want sent to us. So we apply those filters and then do a Rebuild all of the main object. We must also remember that there are some non-compatible DataTypes.

There are some examples like MailMessenger, Cookie, ExcelDocument, and others. You have further information at the wiki, but the main conclusion now is that changing our online applications to offline is quite easy to do.

This is about all I wanted to transmit to you today regarding offline SD applications. I welcome you all to download Evolution 3, test it, and benefit from all the advantages we showed here. And you also have my email for any queries you might need to make.



GeneXus

Thank you!

Martin Torrado
mtorrado@genexus.com