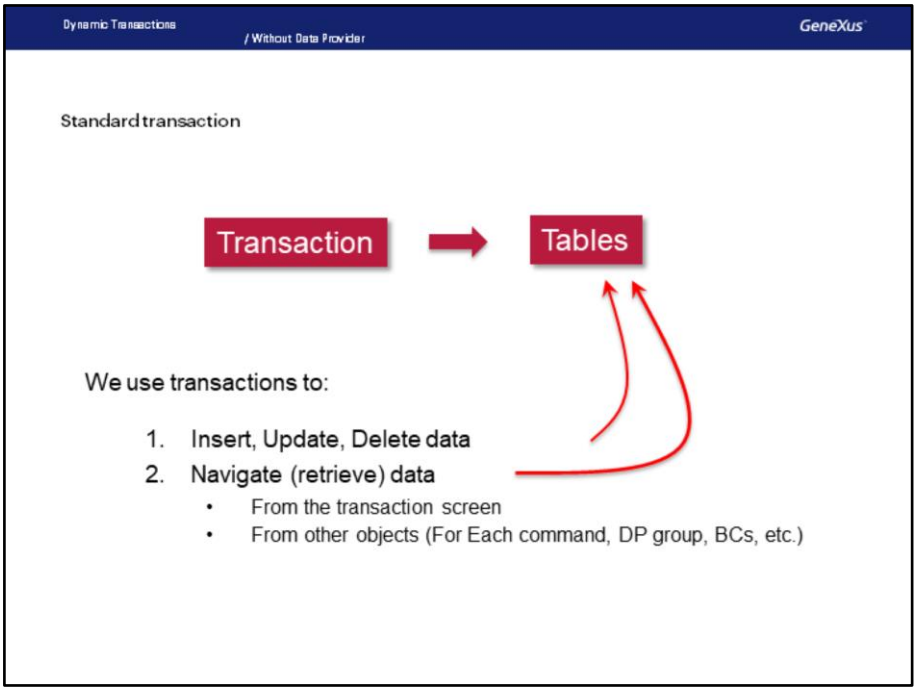
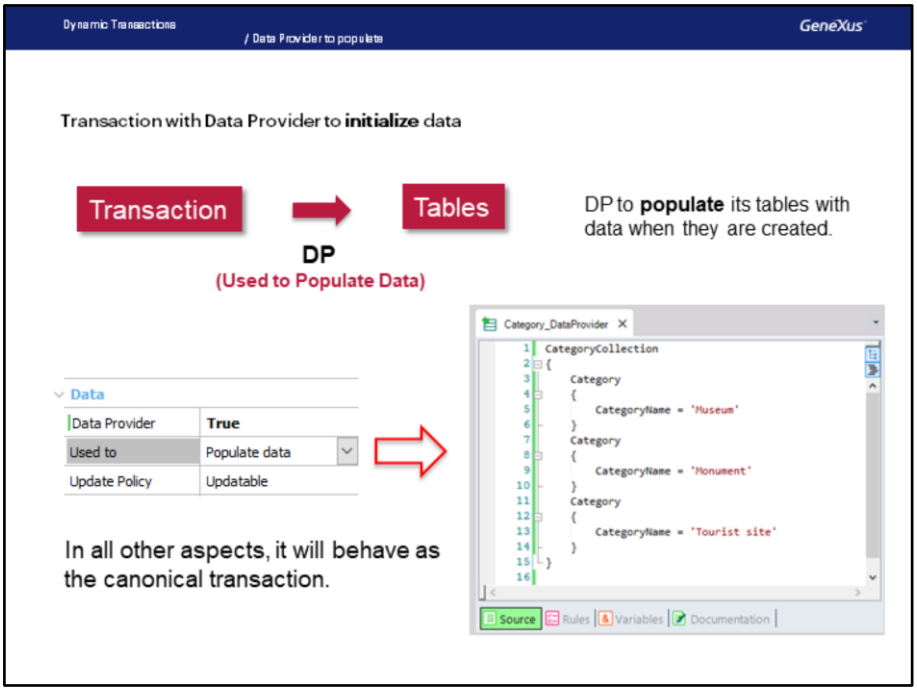
Dynamic Transactions
On-demand Data Retrieval
Transactions as “Views”

GeneXus 16



So far, we've seen that for every transaction object, a table for each level is created to store its data and retrieve it later.

The transaction, in its canonical form, is used to perform the **insert, update** and **delete** operations on these tables through its screen, as well as to navigate (**retrieve**) the data in these tables.

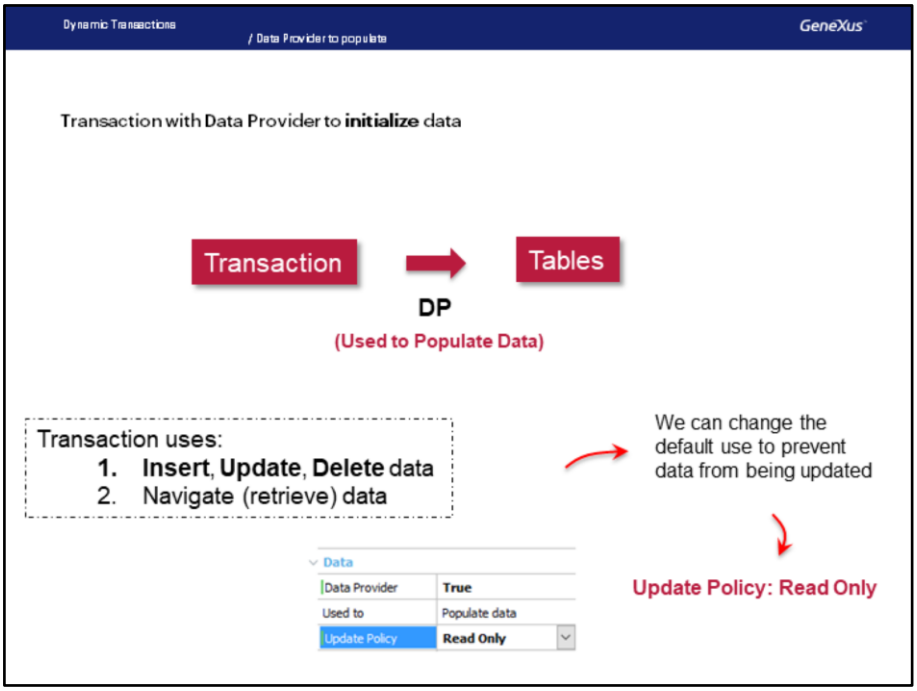


We had already seen how to **associate a Data Provider with the transaction** in order to **populate** its table(s) with data.

Remember that the Data Provider “Used to” populate the tables with data (“Populate data”) will be run in the reorganization, when the tables associated with the transaction are created.

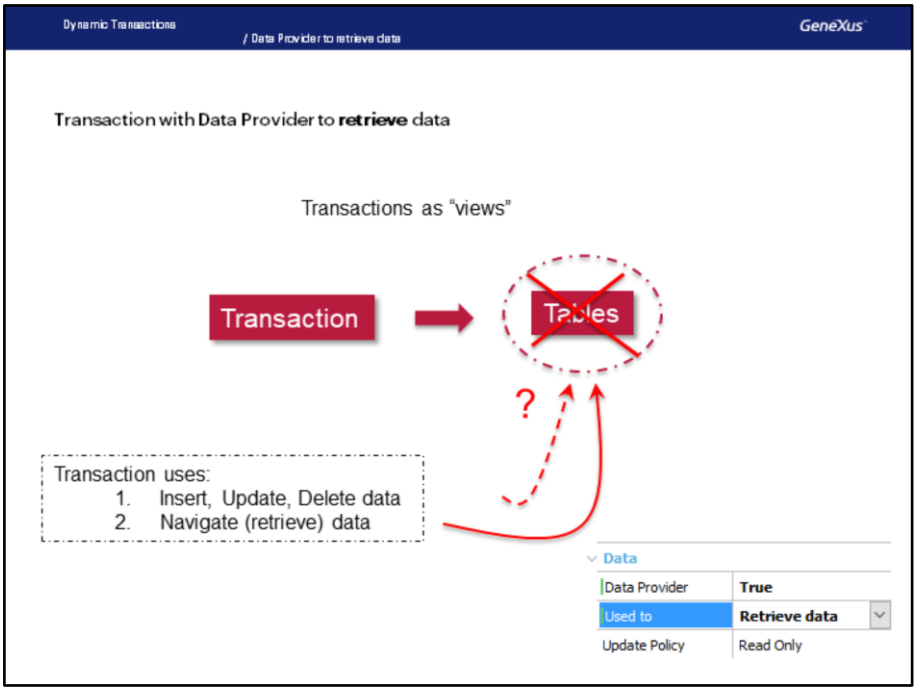
Note: If this Data Provider is changed anytime later, it will be run again in the next F5. Therefore, the data that is already included in the tables must be handled with care.

The Data Provider is only used for initialization purposes. Next, the transaction will behave as the canonical one; that is to say, it will access its tables as usual to retrieve the data, and will allow **inserting, updating** and **deleting** records in the usual way. Pay attention to the **Update Policy** property and its **Updatable** value.



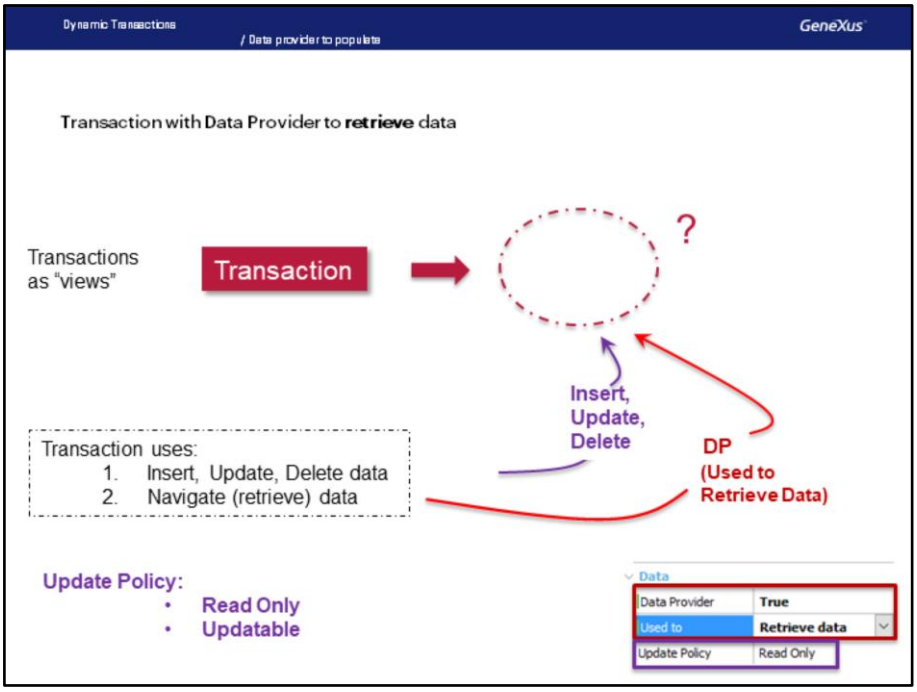
We can also change the default use of the transaction in order to prevent its data from being updated. That is to say, once the tables have been initialized with data, it will not be possible to change existing data or create new data.

To this end, after indicating that the transaction will have an associated Data Provider used to “Populate Data”, the update policy that is “Updatable” by default will be changed to “Read Only”.



Here we will see that it is possible to keep the transaction uses —insert, update, delete, and navigate (retrieve) its data— without storing data in canonical tables. In sum, we will have transactions that don't create tables in the application database.

If tables are not created, we will have to indicate where the information will be retrieved from every time that their data is navigated. In addition, we will have to indicate what to do when the user enters data on the screen and wants to “insert” it into the “tables” (or “update” or “delete” them).

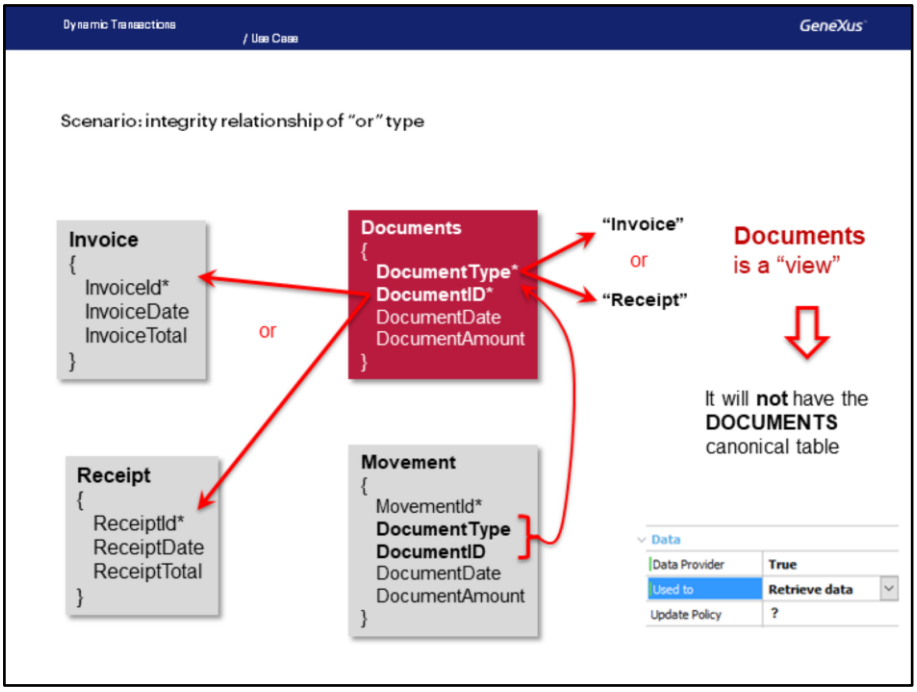


To Insert, Update or Delete data, we will have to explicitly program three events with these names.

To Navigate (retrieve) the transaction data, we will have to program the Data Provider associated with the transaction.

Just like when we used a Data Provider only to populate the tables with data, we could avoid updates by setting a property to indicate that the process would be read-only; here we may also want to use the transaction only to retrieve its data, not to update it. The property used will be the same: it is called Update Policy and accepts the two values that we've shown.

Next we will see an example to explain what to do and how to do it.

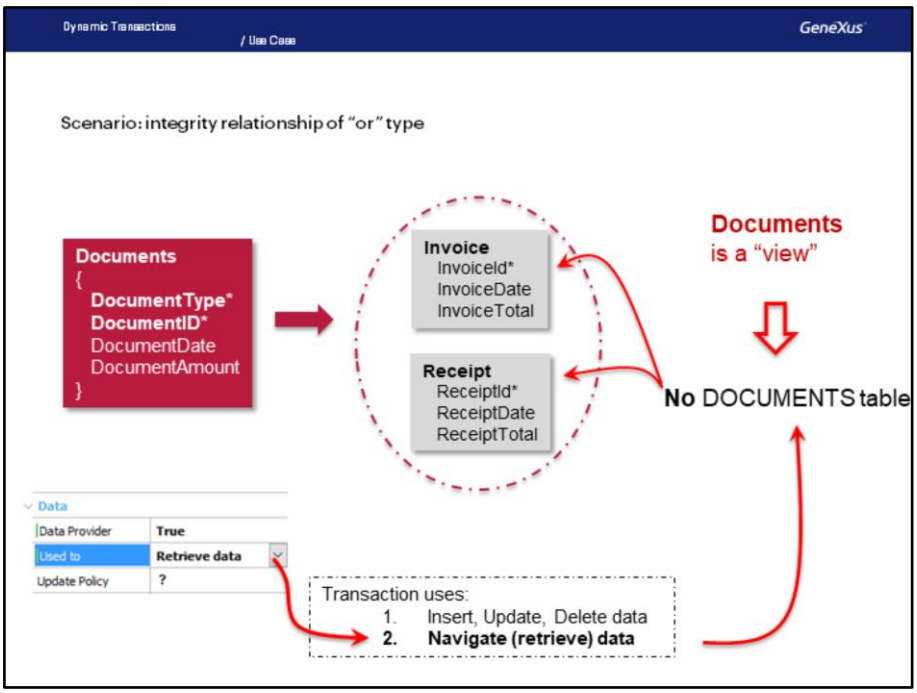


Let's suppose that we have two standard transactions:
Invoice, to represent the invoices issued by the travel agency to its clients for purchasing tickets, trips, and so on. These invoices are identified with a sequential number.
A Receipt that is used to represent the receipts issued by the travel agency to its clients for their purchases. Receipts are also identified with sequential numbers.

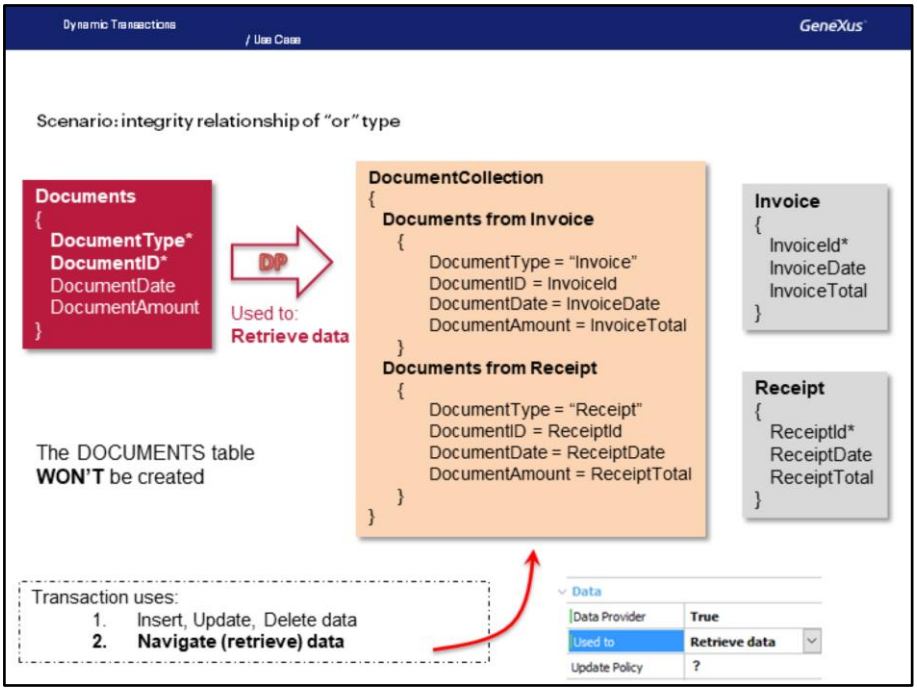
The travel agency's accounting system will have to handle invoices and receipts in "movements". In movements, receipts are a type of document, as are invoices. Movements are identified with a single, autonumbered ID. Note that movement 1 can belong to invoice 1, and movement 2 can belong to receipt 1. The movement transaction unifies the data of invoices and receipts.

That's why the Documents transaction is created with an identifier made up by DocumentType and DocumentID. This transaction will be like a "view" that combines the information included in the Invoice and Receipt tables. That is to say, it will not create a table to contain the data; instead, it will take it from the tables corresponding to Invoice and Receipt, which have identical names. Then, the Movement transaction will be a standard transaction that will generate a MOVEMENT table with DocumentType, DocumentID as pseudo-foreign key.

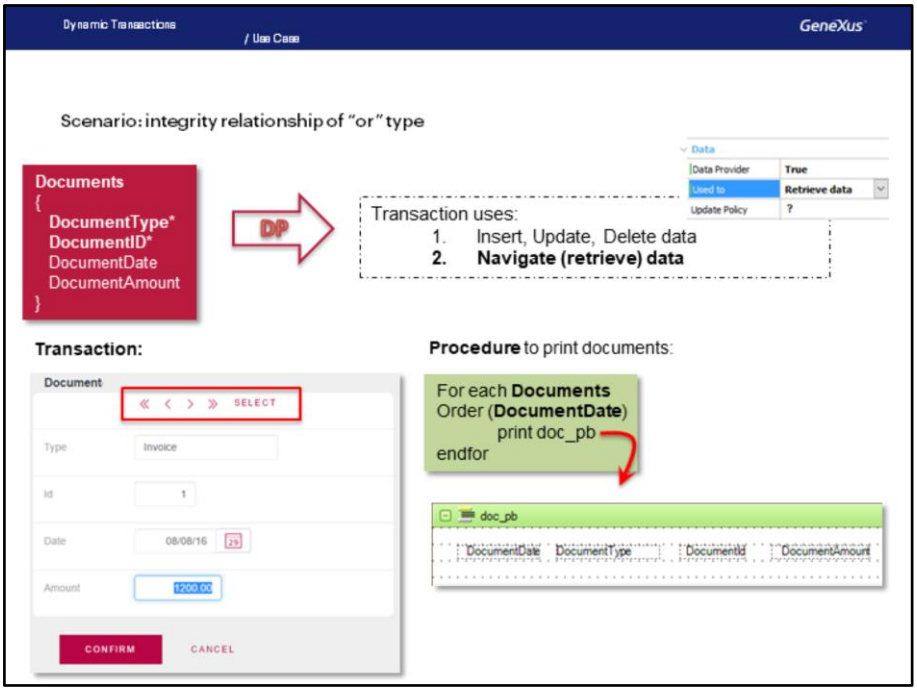
Why is it a "pseudo" foreign key? Because there won't be a DOCUMENTS physical table with DocumentType, DocumentID primary key to make reference to. However, it will exist at the logical level and, as we will mention later, the referential integrity controls will be made.



When we indicate that the transaction will have an associated Data Provider for the data, the "Used to" property is enabled (the "Update Policy" is always enabled). If we indicate that this Data Provider will be used to **retrieve the data** (property Used to: Retrieve Data) GeneXus will automatically understand that the table associated with the transaction must not be created because this Data Provider will be used to indicate where to obtain the data from. In this case, it will be from the INVOICE and RECEIPT tables, which are associated with the transactions that have the same name.

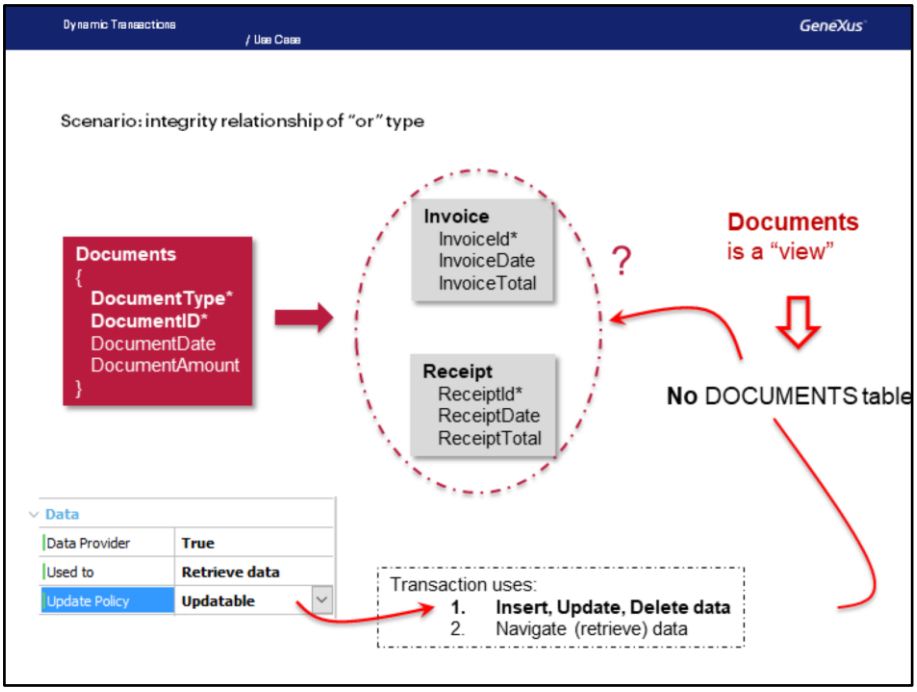


The Data Provider Source will be stated in this way. We have a Documents group to retrieve all the documents which are invoices, and another group to retrieve all the documents which are receipts.

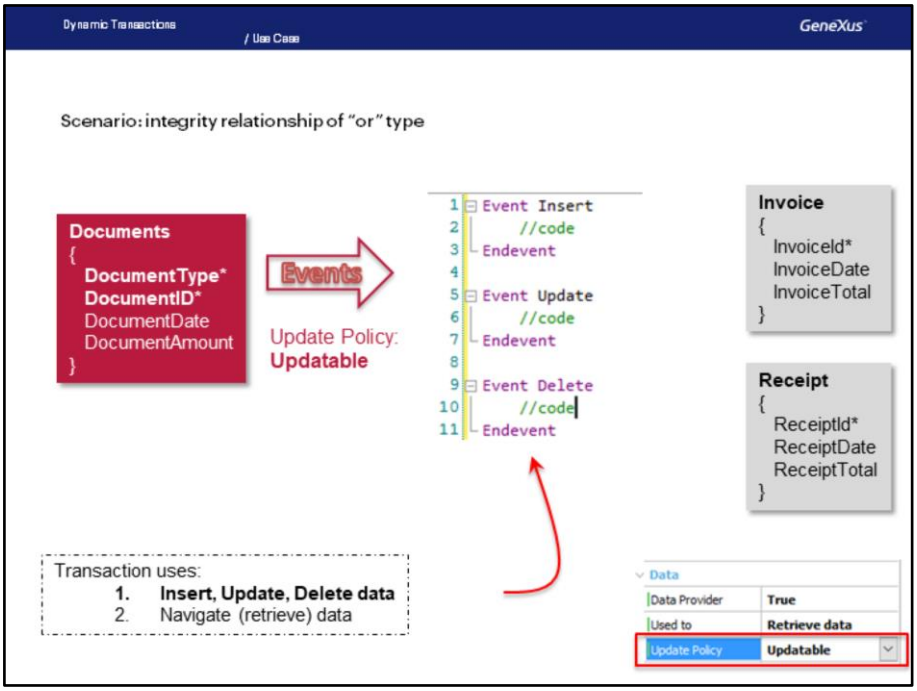


From here on, every time the transaction is run to navigate its data, this Data Provider will be run to load the corresponding information on the screen, in a transparent way for both the developer and the user, who will never notice that it is a transaction without a table.

Then, the dynamic transaction is used as any other transaction. For example, to print all the documents ordered by date in descending order, a procedure will be created with the For Each command that is displayed. In the printblock, the attributes DocumentDate, DocumentType, DocumentID, DocumentAmount are added as usual. GeneXus manages to obtain this data from the Data Provider that contains its logic.

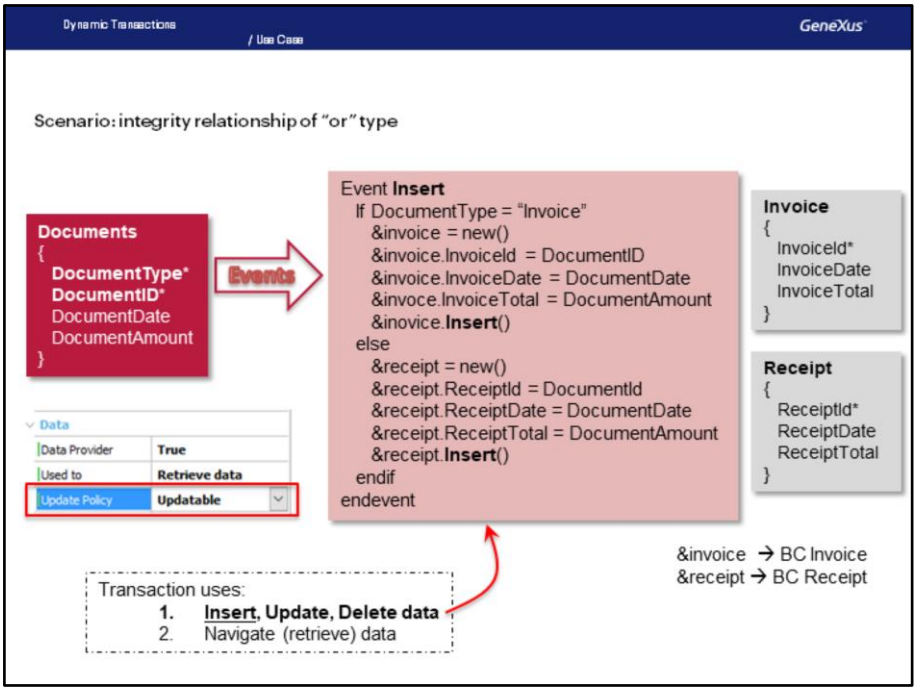


Transactions are not only used to retrieve data but also to update it. How do we go about it since we don't have a table associated with the transaction?



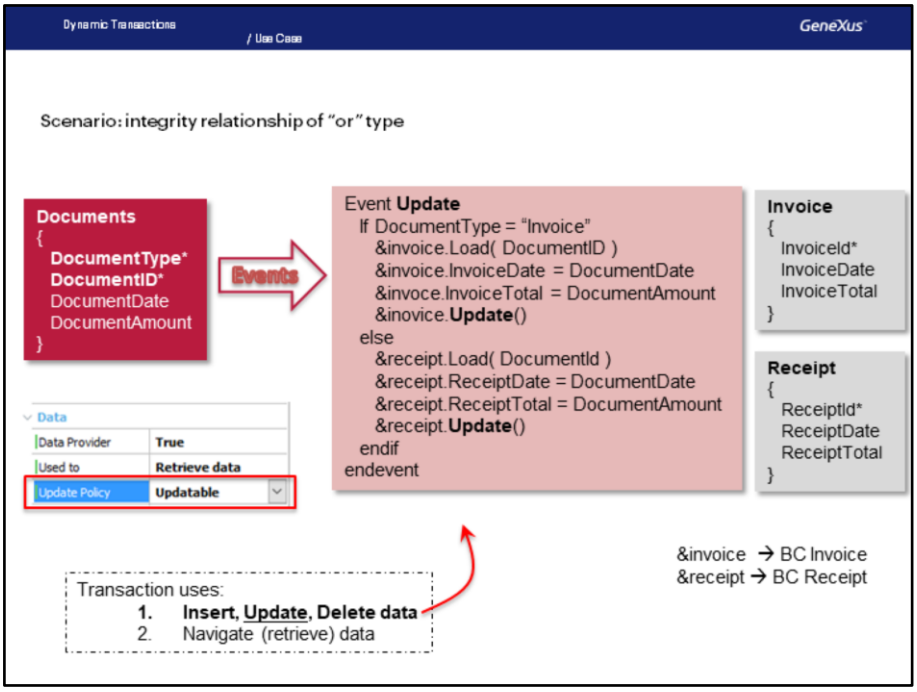
If the **Update Policy** property is set to "Updatable", the Insert, Update and Delete events will be offered to program the insertion, update and deletion of the data entered by the user on the screen. Only the developer will know what to do in each case with this information.

These actions may or may not be allowed depending on the reality. In our case, it seems that they shouldn't be allowed. However, let's suppose that they should be.

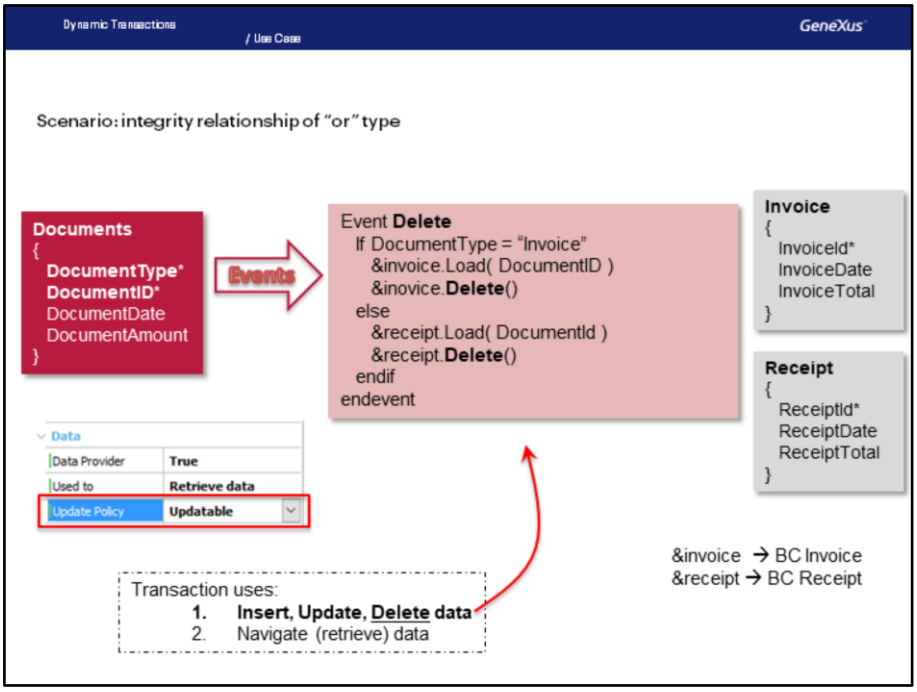


When the user has finished filling the screen fields to insert a new movement and has pressed Confirm, we will have to insert a new record in the Invoice or in the Receipt table, depending on the value given by the user to the DocumentType field. To do so, we used the &Invoice and &Receipt variables of the Business Component Invoice and Receipt data types, respectively (that must have been obtained from the transactions).

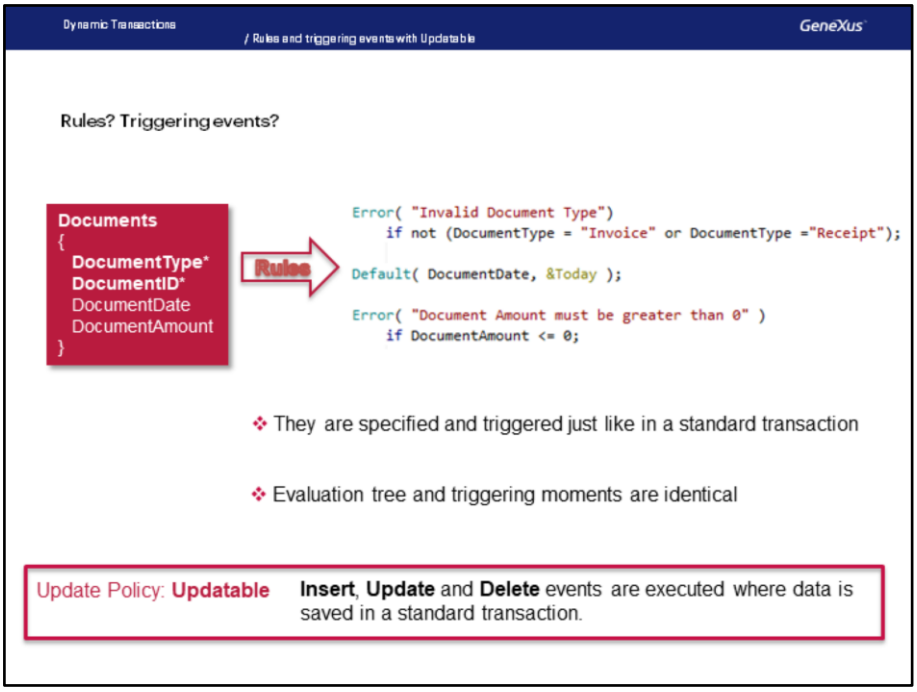
Instead of the Insert method of the Business Component, we could have used the Save method. Note that we don't need to write the Commit command because we're in the Documents transaction that still has the Commit on Exit property set to Yes by default. That is to say, it will implicitly run the Commit.



Here we see how we would code the Update. Since we don't know which fields have been changed by the user, we assign all the values.

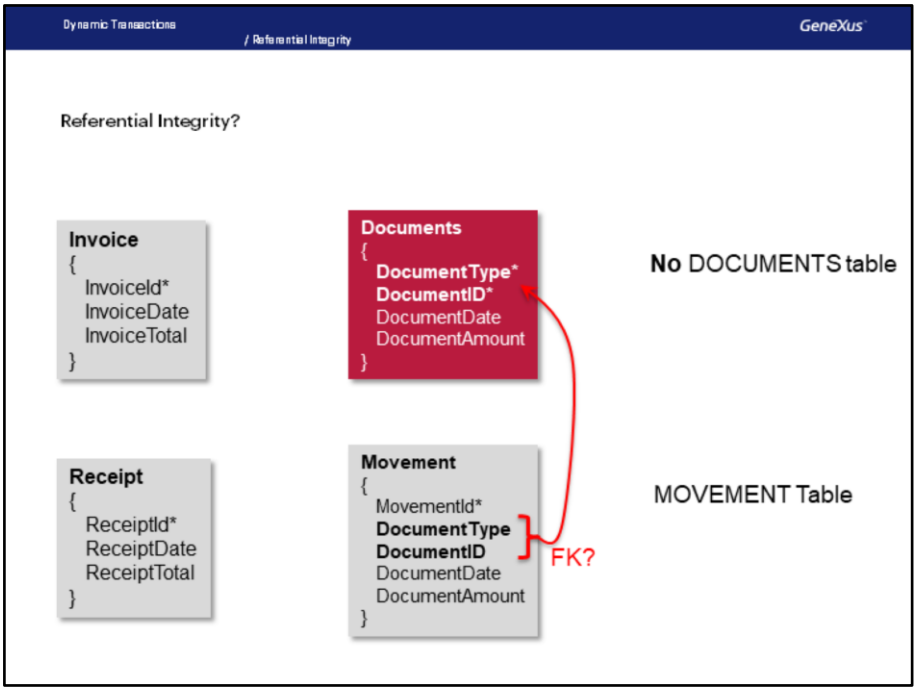


Lastly, the Delete operation.



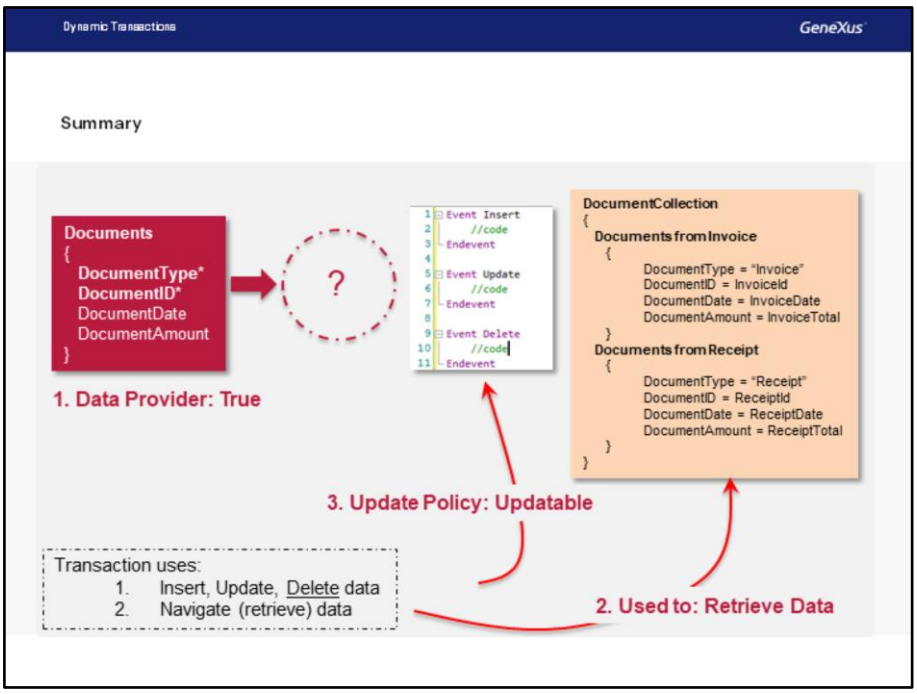
We haven't asked ourselves what happens if we have rules stated at the dynamic transaction level. When are they triggered? What happens with the evaluation tree?

What happens with the success or failure messages of Insert, Update, Delete operations? They are similar to “Data was successfully added”.



Since the DOCUMENTS table associated with the Document transaction will not be created, we may assume that in the MOVEMENT table associated with Movement standard transaction, the DocumentType and DocumentId attributes will not be able to make up the foreign key as they should. So, does it mean that GeneXus will not be able to check referential integrity?

Since referential integrity must be ensured, GeneXus generates SQL triggers to do so. Therefore, it could be said that {DocumentType, DocumentID} make up a “pseudo” foreign key in Movement. In sum, in Document it will not be possible to delete invoices or receipts that have an associated movement. In addition, it will not be possible to add a Movement that doesn't exist as a document.



Here is a summary of the properties and their effects.

Summary

1. Data Provider: True
2. Used to: Populate Data

DP (to initialize)

3. Update Policy: Updatable

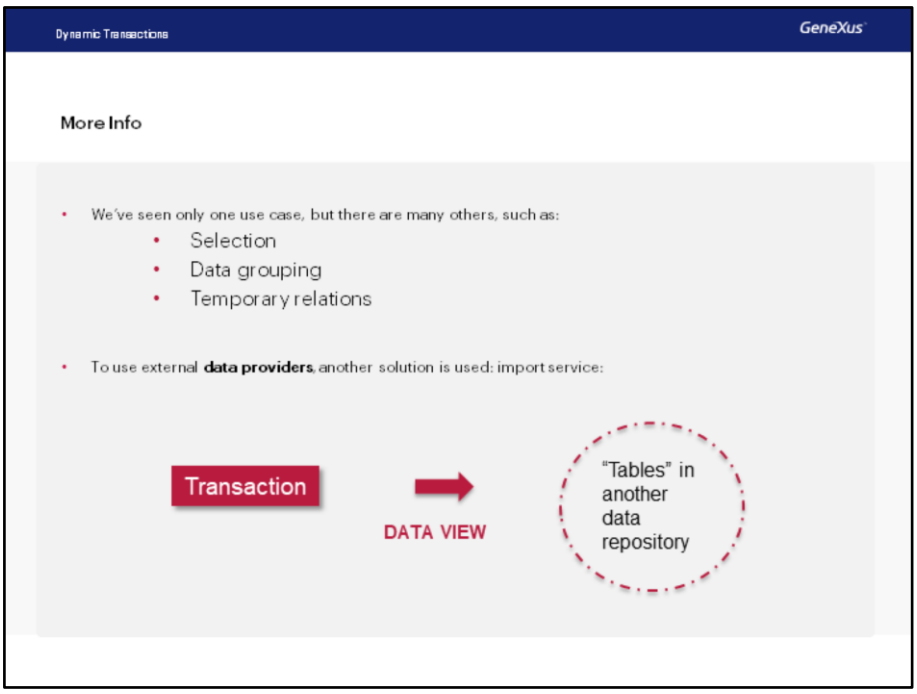
It allows (or not) making the usual updates (on the transaction tables)

1. Data Provider: True
2. Used to: Retrieve Data

DP (to retrieve)

3. Update Policy: Updatable

It allows (or not) making updates, but they have to be programmed in special events: **Insert, Update, Delete**



Here you will find more examples of use of dynamic transactions and an in-depth explanation of the topic. http://wiki.genexus.com/commwiki/servlet/wiki?28062,Dynamic%20Transactions.

For external data providers that handle data repositories with some relational algebra (they don't have to be databases in the SQL sense), it is solved differently by importing the service (for example: Odata, CouchDB, others that are not SQL). By doing it, GeneXus automatically generates the transaction and a **Data view**, which is an object created by GeneXus to provide the communication interface between the transaction and the external "table". This will be another case in which the transaction doesn't create a table in the proprietary database. It is used when Reverse Engineering is applied (with the Database Reverse Engineering Tool).

In this case, as with dynamic transactions, the developer will use BCs and For Each commands as usual, which will be internally translated into invocations to the external service.

GeneXus™

The power of doing.

Videos	training.genexus.com
Documentation	wiki.genexus.com
Certifications	training.genexus.com/certifications