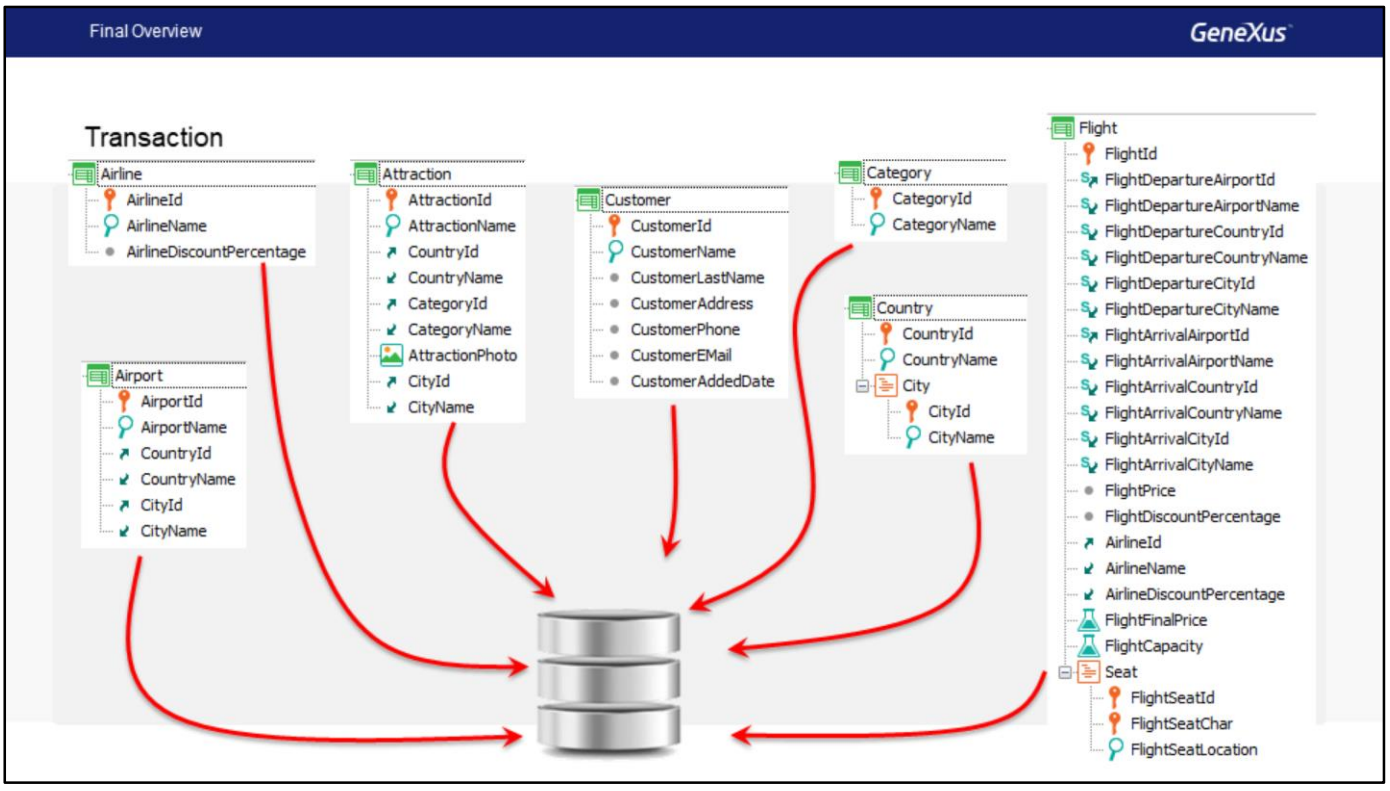
# Final Overview

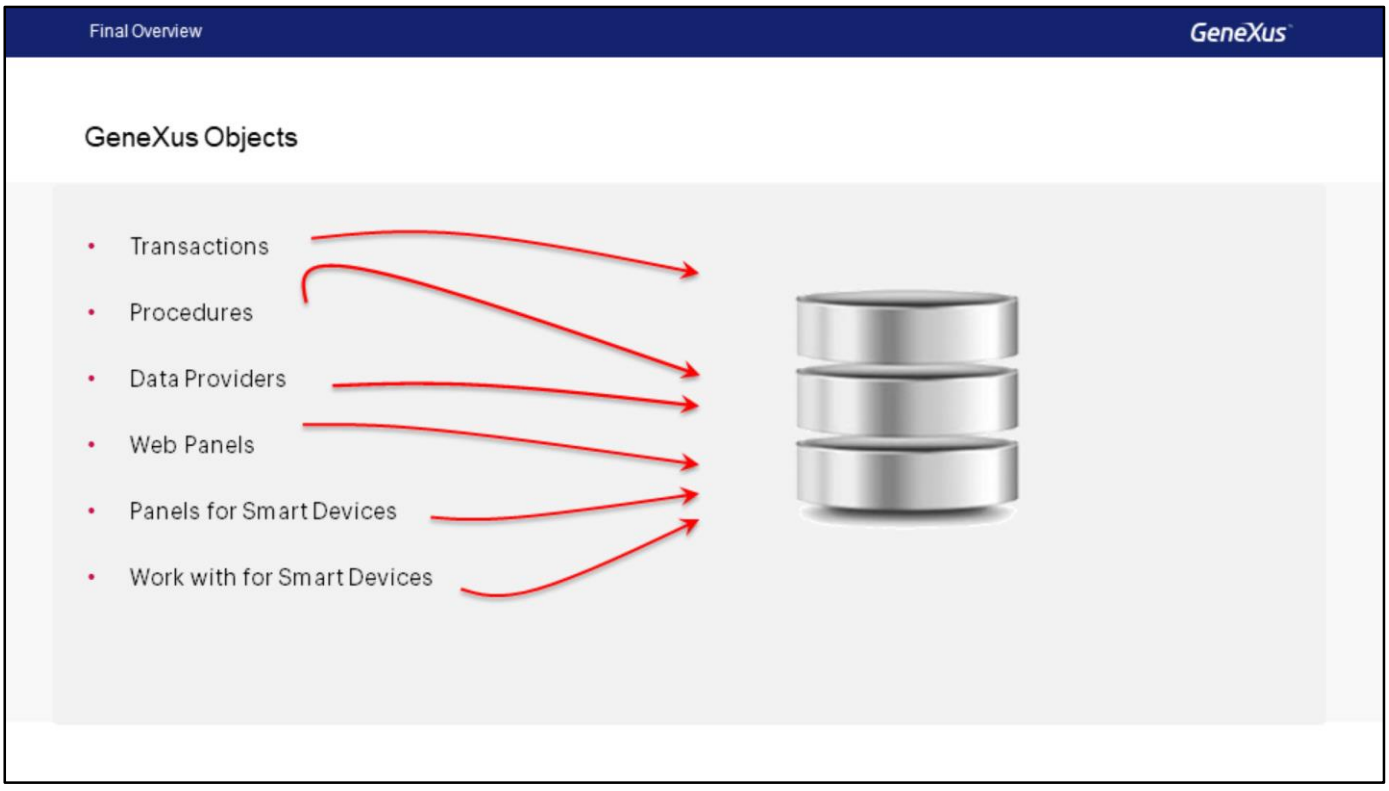
GeneXus 16


Throughout this course we have focused on the main GeneXus objects that enable the implementation of the most significant functionalities in a web application, besides having mentioned those that implement applications for smart devices.

We have seen that, based on the Transaction type objects defined in the knowledge base we could build the data model…




We could build the data model…that is to say that the entities from reality and their attributes, together with the way in which they relate to one another was later reflected on the creation of a database with its respective tables. Such creation, as well as its subsequent reorganizations were handled by GeneXus, as developers were freed from having to think at the physical level, that is, at the table level. So they could just focus on viewing the data at the highest level, more specifically, the level of transactions and their attributes.



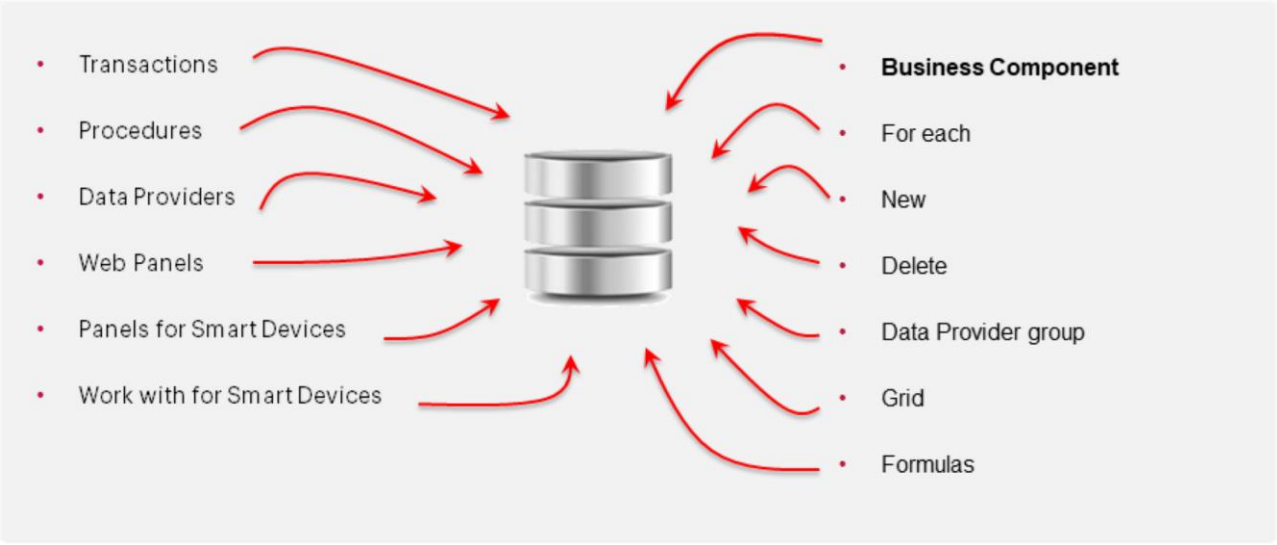

All the other objects we see here could then access the tables of the database to query or to also update information. But the details on such access are disguised in GeneXus, for it is in charge of relating attributes and transactions with database fields and tables. In this course, we have shown how this is done, but we need not know this at all.

But even when we have confirmed that the GeneXus philosophy implies that we can forget about tables to concentrate our thinking on the attribute and transaction levels, throughout the course we have gone to the database tables over and over again to explain each of the concepts were considering. In fact, this is opposite to the referred philosophy, which, again, implies the contrary, that is make developers independent from the storage details and the implementation so that they may think and express themselves at higher levels. This would be the case with individuals who are not fully proficient about computer programming, though capable of expressing a reality that is to be modelled. Such form of expression only calls for transactions and attributes, and it requires no tables. Why did we continue to go to the tables then? The reasons we had for it were basically founded on learning purposes.

But now the time has come to question all this.

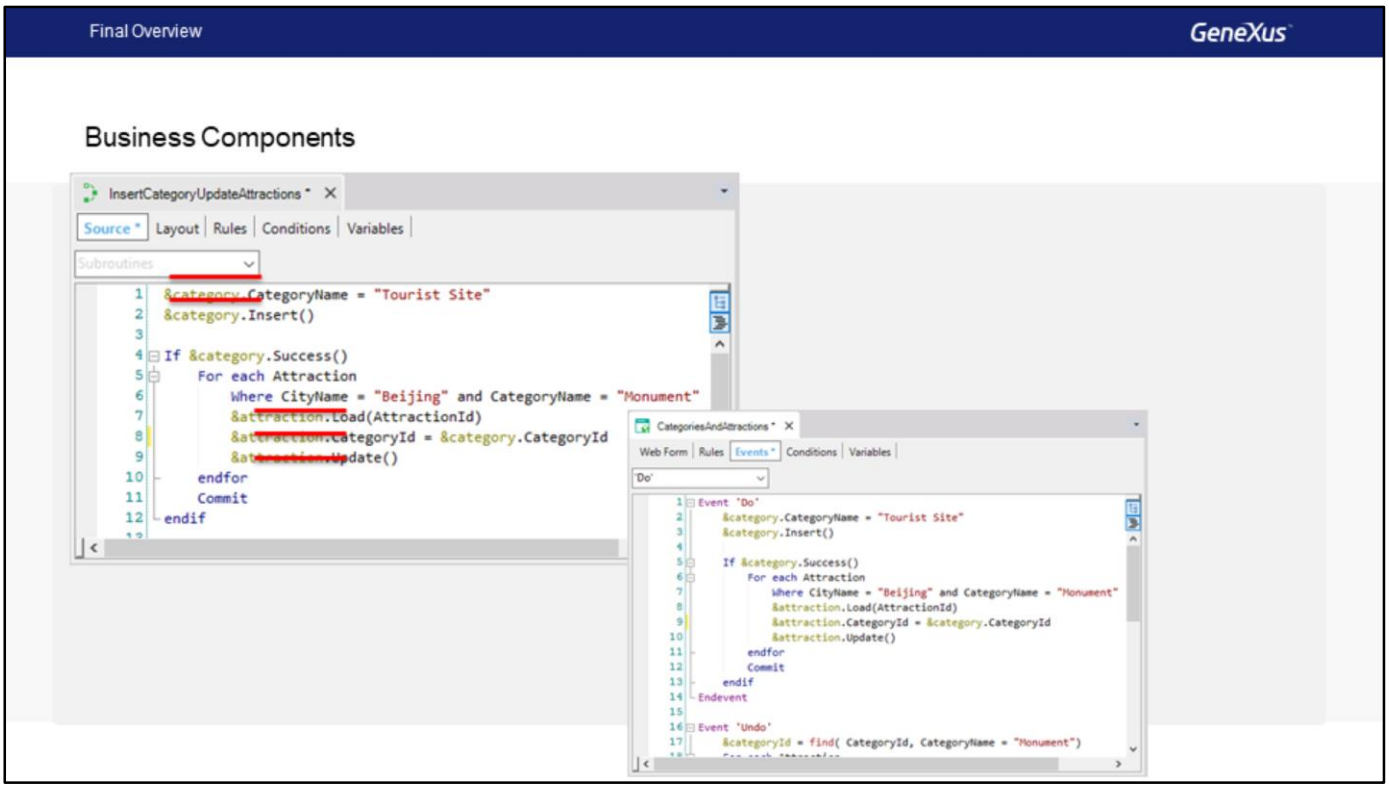
Despite the fact that all these objects could access the application's data, to do so, we never mentioned tables explicitly in any of them. What we did was name attributes and base transactions. And this the manner in which developers are expected to work.

GeneXus objects, types, commands, groups, grids, formulas




Here we added the data types, commands and other aspects that enabled, in the various objects, the access to data, both for just querying as well as for updating it.

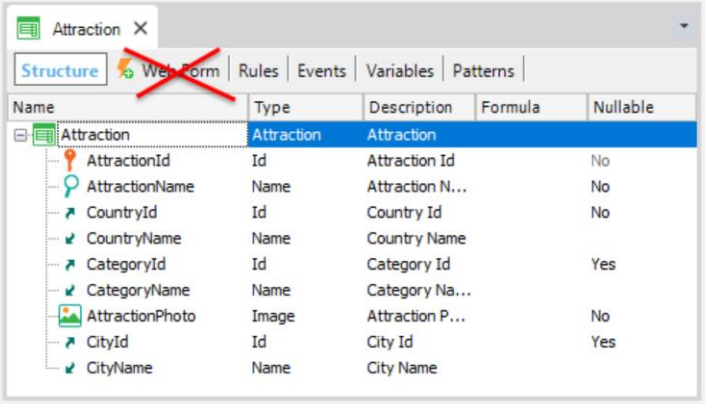
**Business Components** were a sort of structured data types built on transactions, from which they took the data structure and their rules and events, though with no screens or anything associated with them.




This was how Business Components were associated with variables, allowing –through methods– us to work with database without the need for thinking about physical tables. They were like a mirror for the transactions, though operating by code.

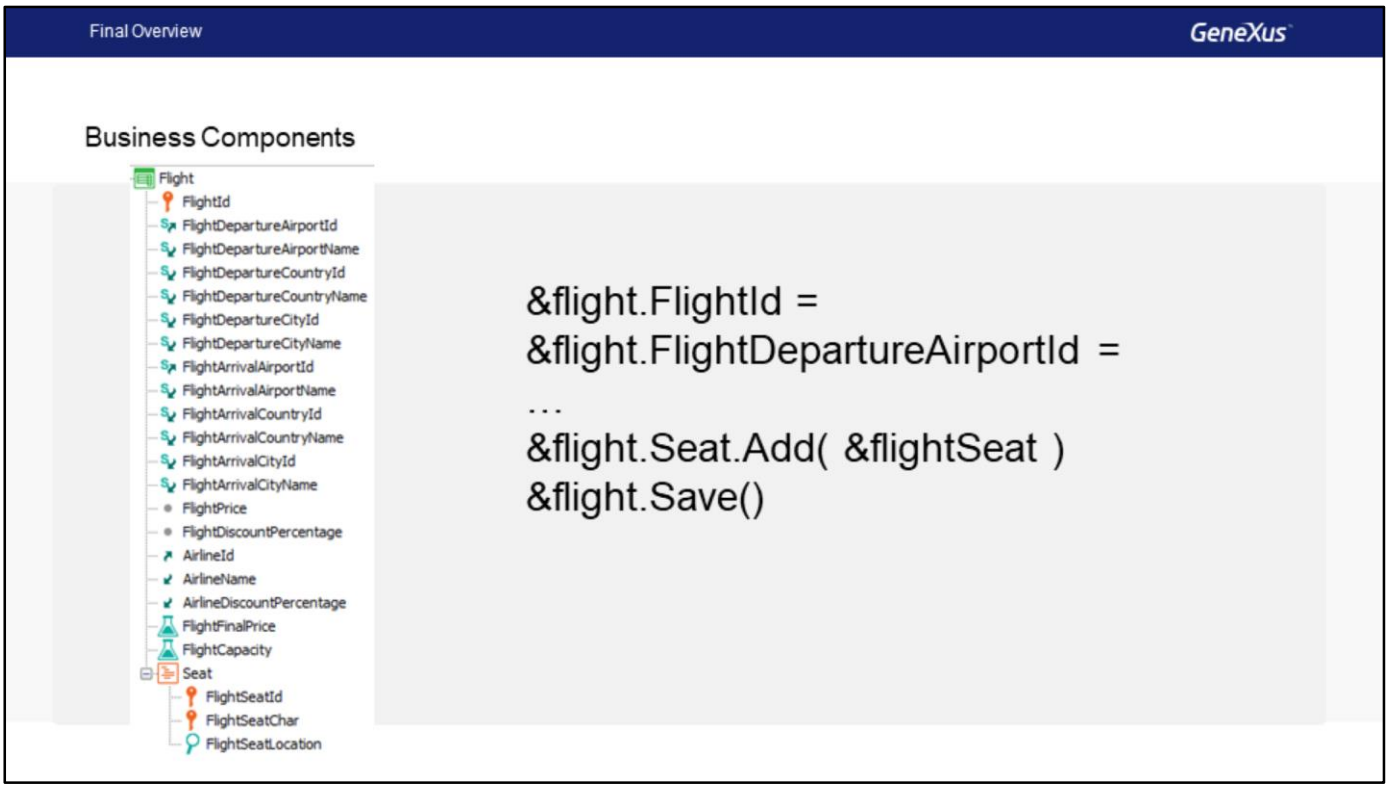
Since they were used through variables, they could be applied to several GeneXus objects (for example, in the Source of procedures or in events of web panels).

## Business Components

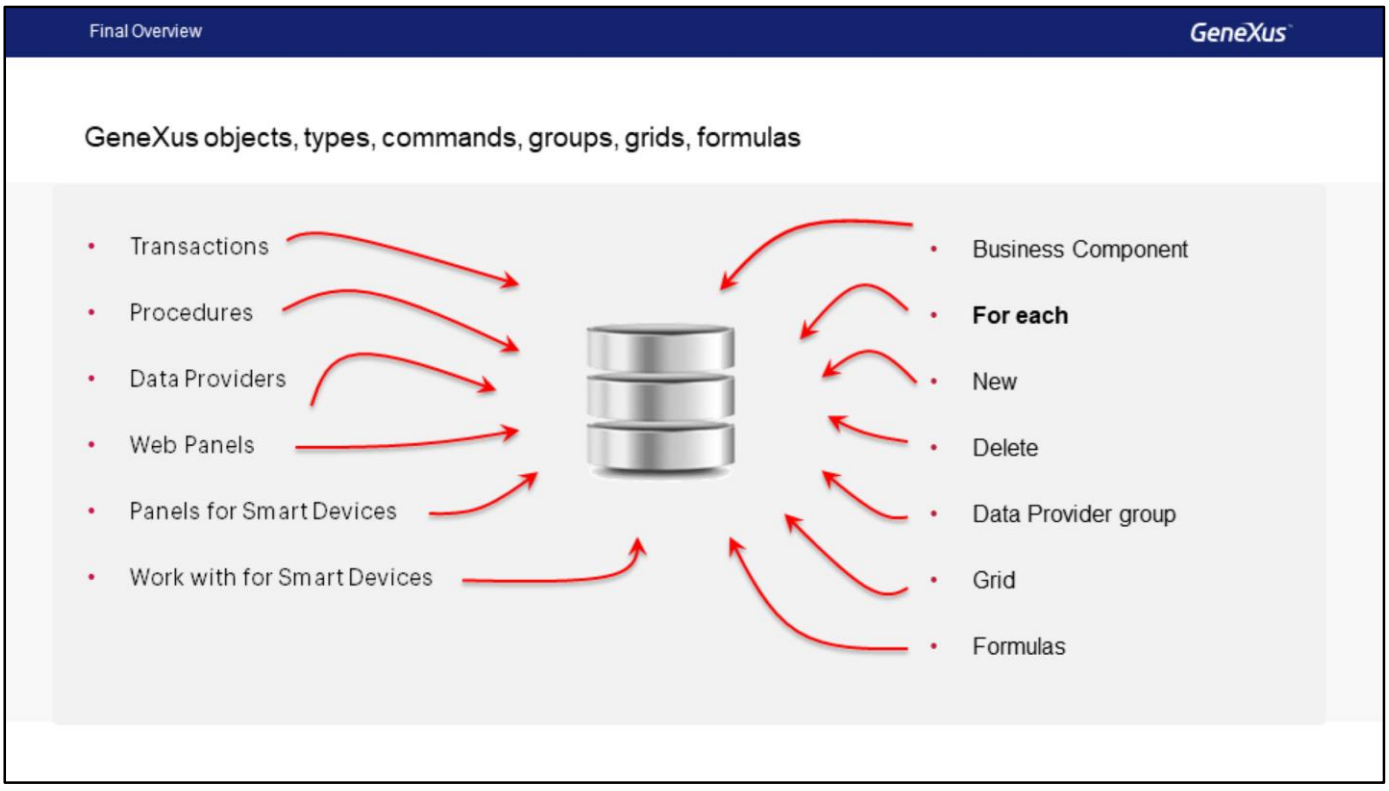



```
&attraction.AttractionId = …
&attraction.AttractionName = …
&attraction.CountryId = …
&attraction.CategoyId = …
&attraction.AttractionPhoto = …
&attraction.CityId = …
&attracion.Save()
```

Inserting information with the use of a business component is exactly the same as doing it through the associated transaction (though without its screen). We should recall that data integrity is controlled and the transaction rules are also executed.

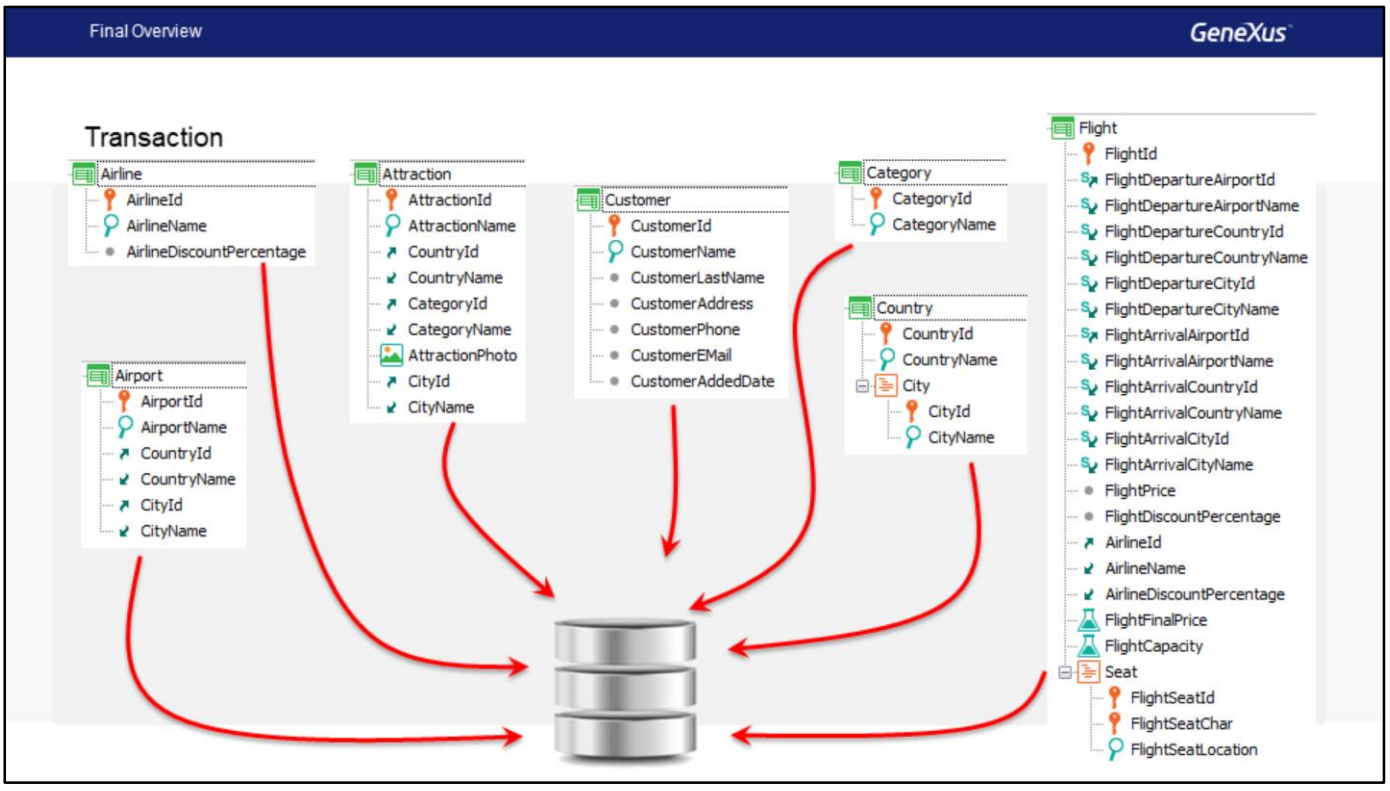



Even when we did not consider it in this course, a business component may have two levels when the transaction has two levels. As an example of this we have the Flight transaction.




The For each command is one of the main commands in GeneXus, since it may be used for almost any object.

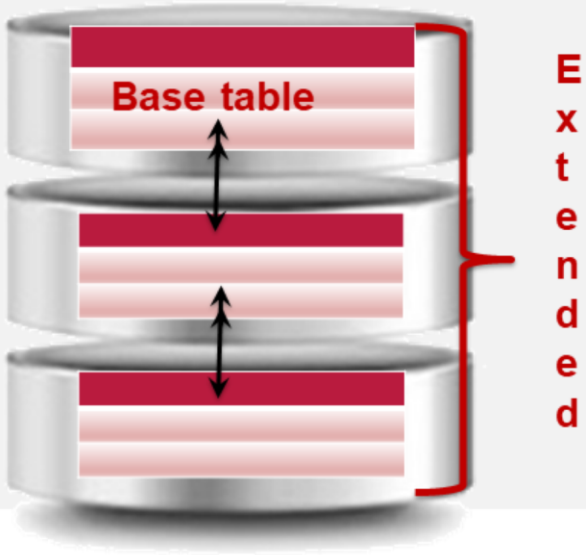
It is meant to go over each element in a transaction's level, meaning an actor from reality, if we consider each level in the transaction as an actor.



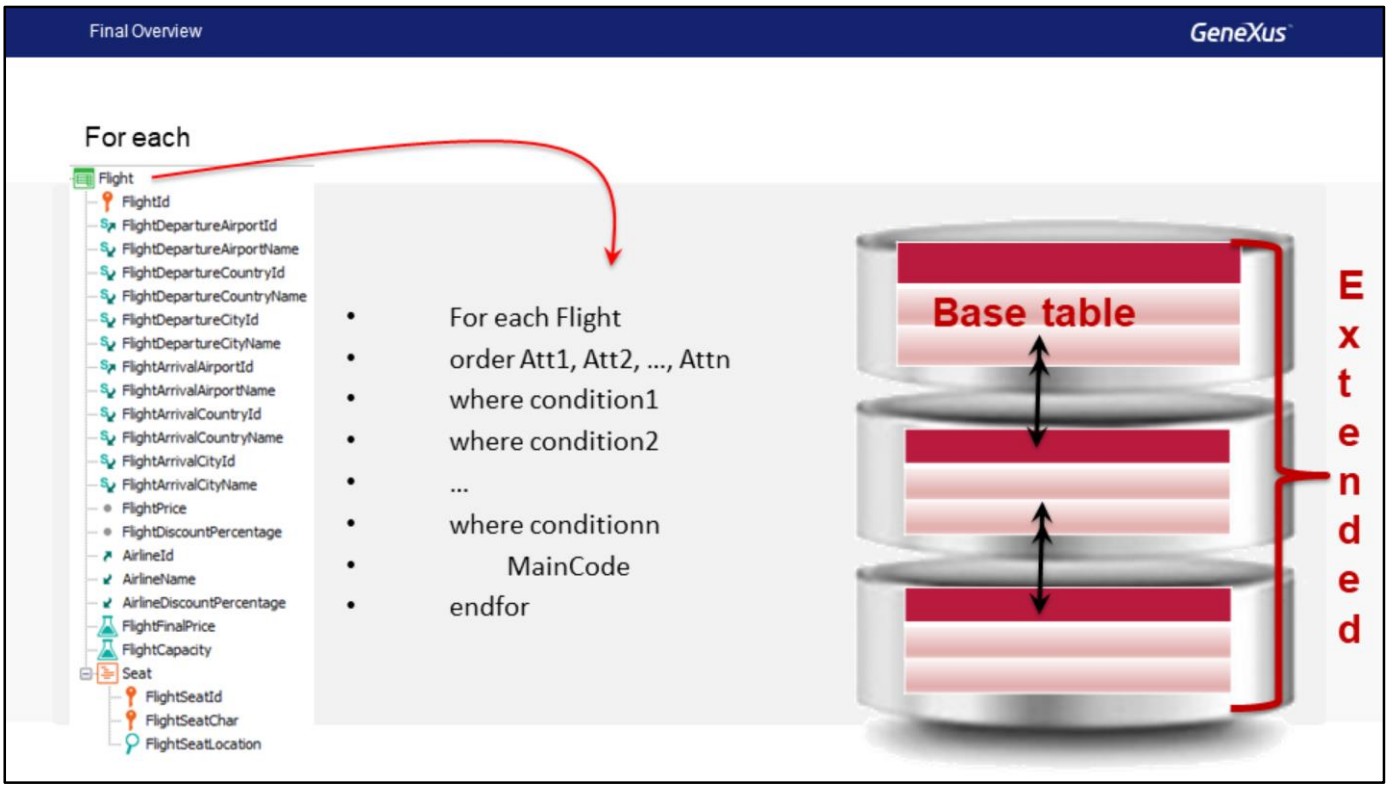

For instance, each tourist attraction, or each flight, or each seat on a flight, or each country, or each city, etc., for which something is to be done with its information.

## For each

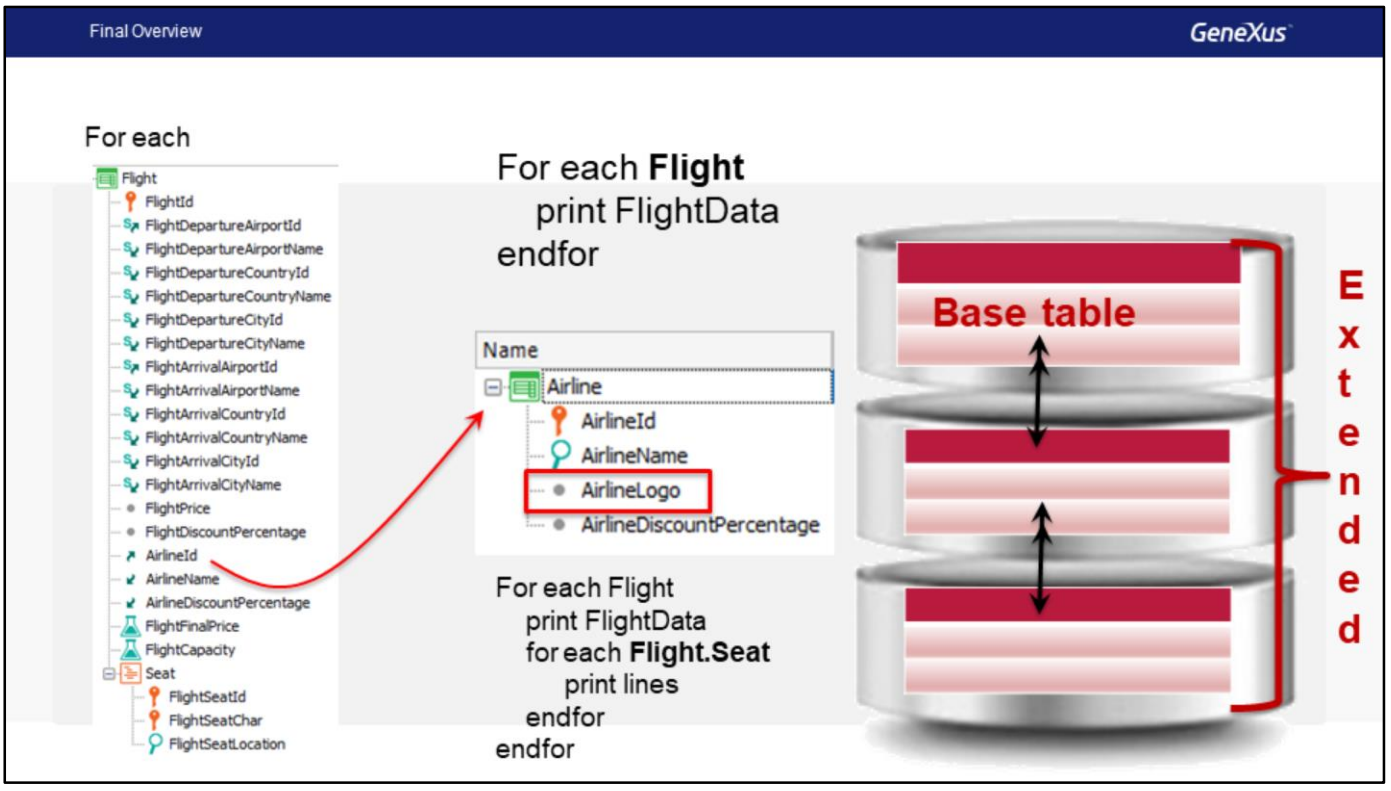
- For each BaseTransaction
- order Att1, Att2, ..., Attn
- where condition1
- where condition2
- ...
- where conditionn
- MainCode
- endfor




As we studied it, we saw that we indicated the base transaction but never the associated table. That associated base table was inferred by GeneXus, and it was important only to know all al the information we could have from it, that is: with the extended table, which is indeed a basic concept.

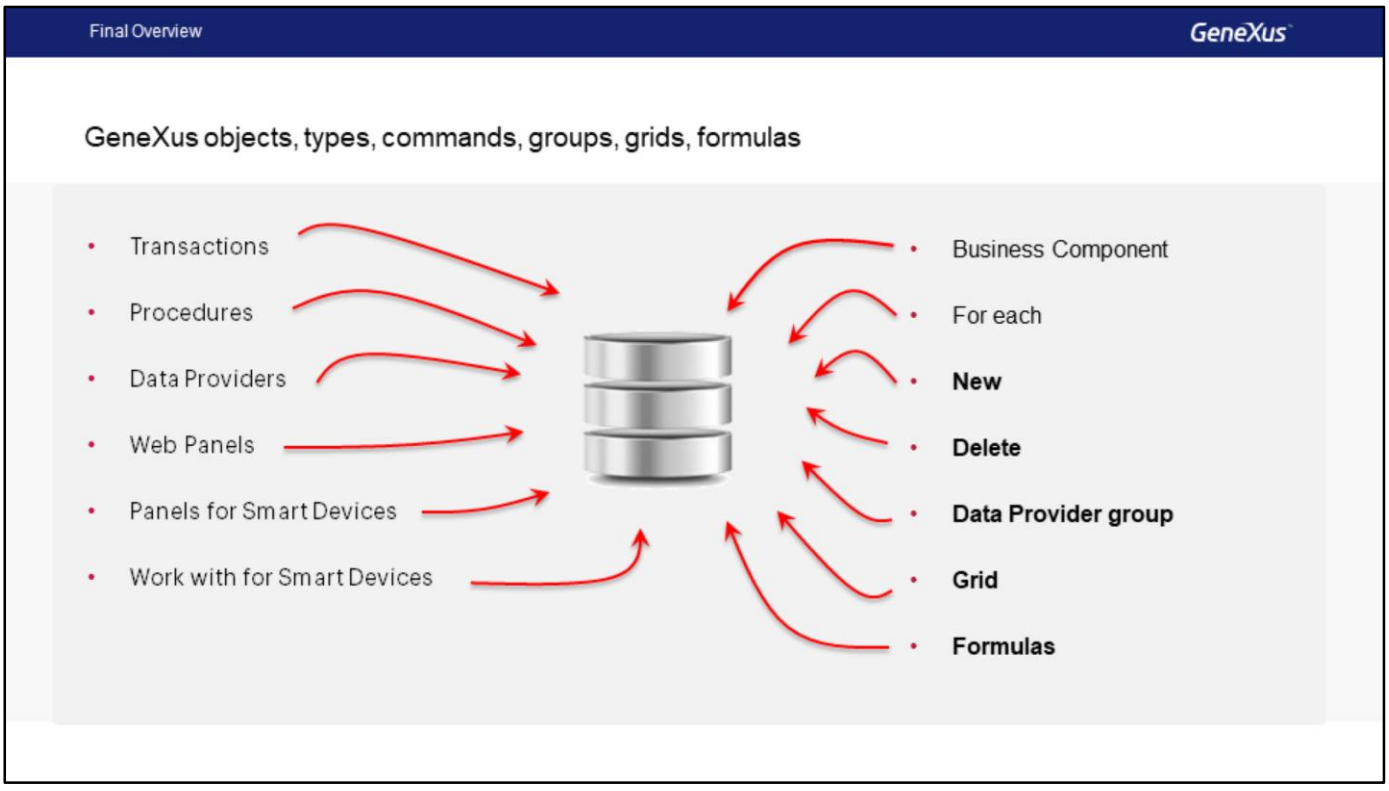



In other words, we indicate to the For each the level of the transaction with which we want to work, and it is from there that the extended information is to be deduced. This information might not be included in the structure of the transaction, but is may be “reached” based on the relations between entities. For example…




…if the airline had an attribute that saves the logo, even when we have not specified it in the structure of the Flight transaction because we did not need it there at all, it is absolutely possible to “reach” AirlineLogo by means of a For each with the Flight base transaction. In other words, for each flight, the logo of **its** airline is found. And, for example, we may print it along with the rest of the flight information. This is nothing but the significant concept known under the name of **extended table**.

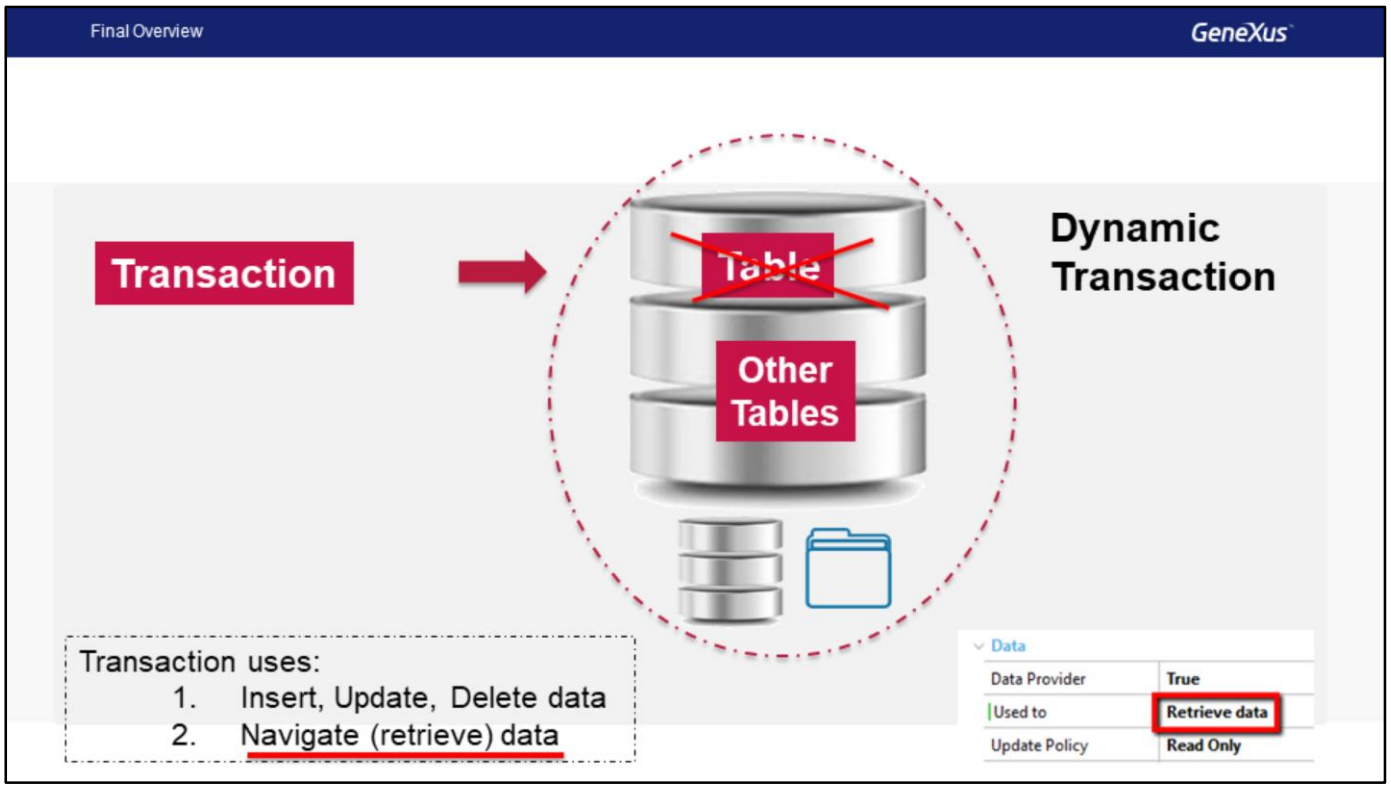
So, what is really important in a For each is not so much the base table, but rather the extended table that we obtain from it, which in turn is obtained from the only indication made by the developer, that is: the base transaction.
It is in fact important to point out that, with a For each like the one referred above, we will only navigate the headers of flights, and not its lines. Lines may not be reached by the For each. To navigate lines we must nest another For each, with Flight.Seat as the base transaction.




The New and Delete commands were briefly introduced, for they are at a lower level since they do not function at the extended table level but at the base table level instead. The New inserts a record in the table and the Delete eliminates records. This is why –due to their scarce degree of abstraction and expressiveness, and because they are not much integrated in relation to transaction rules- the use of business components is recommended instead of these commands.

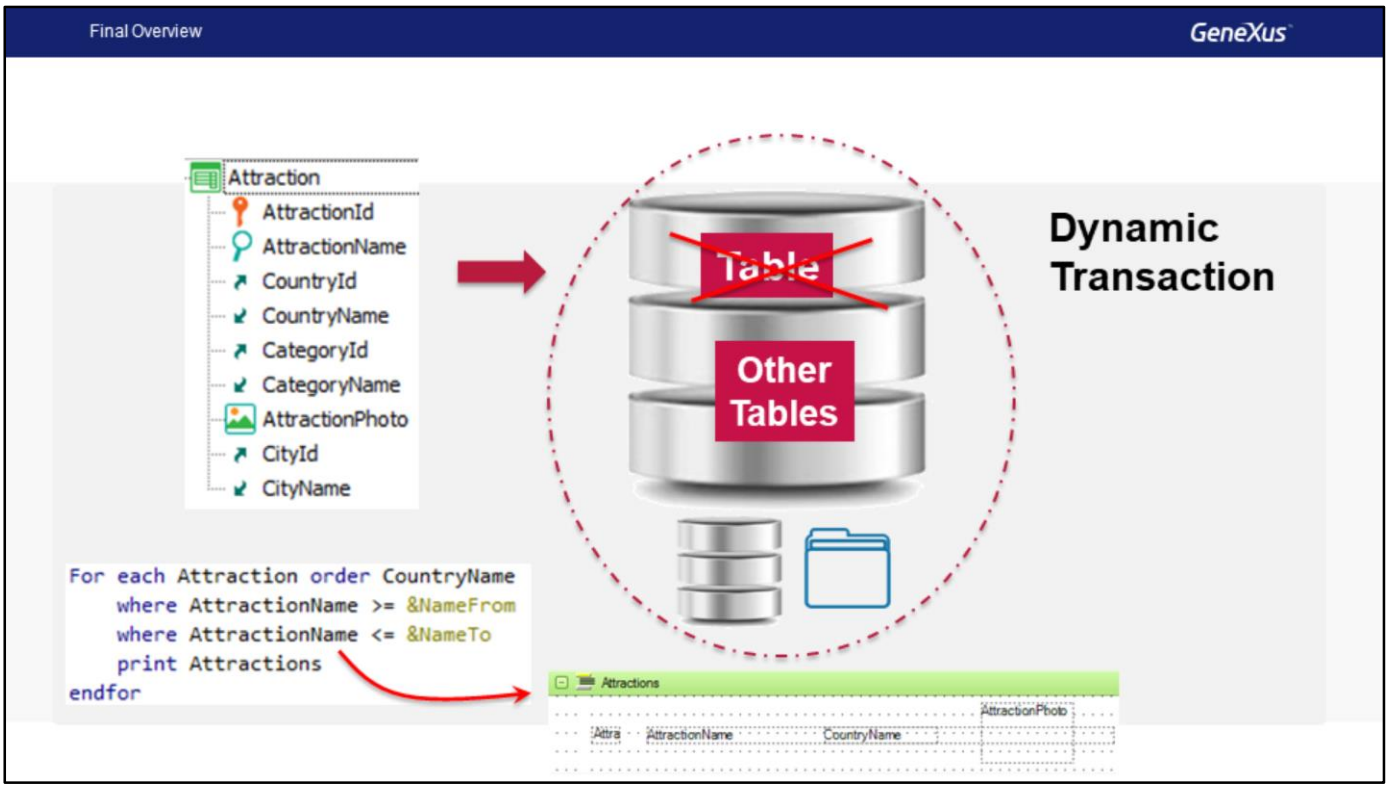
Then, there is the Data Provider group, analogous to a For each. And the same goes for a grid with attributes.

Regarding formulas, also in this case we saw that we never mentioned a table from the database. We always use attributes, and it is from them that GeneXus obtains the information that we want to navigate and the way to do it. Developers must continue to think about higher level aspects, according to our reality, which is created by the transactions and their attributes.




And there is more: we had seen that transactions may create physical tables, which is the regular behavior. Or, they may not create tables and take the data declared in their structure from other tables, form other external databases or from other sources. We just mentioned the case of dynamic transaction. These transactions will be totally equal to the usual transactions, except that the data we will see on screen, based on which we will be able to navigate in the usual manner, is to be loaded through a Data Provider, every time. However, this will not be visible to the user.

And, to a certain degree, it will also remain transparent for the developer ...




… if Attraction were a dynamic transaction whose data is taken from external services. For instance, nothing will stop us from using it as if it were a regular transaction with an associated physical table.

So, if we want a For each to list the attractions, we will continue implementing it just as we did before.

This enables us to better understand what we mean by saying that, in GeneXus, we break away from the physical implementation part. We work at a higher level, with information taken from a more conceptual viewpoint, regardless of where it is stored.

Therefore, every time that we mentioned the “physical table” in the course, the idea was to make understanding easier at the time. Now we know that this is not necessarily the case, and that tables may be a virtual that enable us to think, but their implementation may vary. At this level, it is not really important where the data is located.

In fact, most of the applications we will develop will create and manage their own database, though others will not. There are applications built to work with data provided by external databases, or even data provided by third parties, whose storage sources are unknown.

GeneXus is prepared to deal with all this because it works with a very high level language. And that is what we showed in this presentation.

GeneXus™

The power of doing.

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