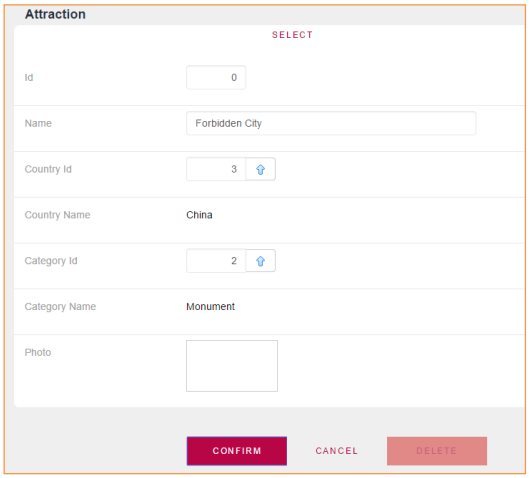
Database update with specific commands for procedures

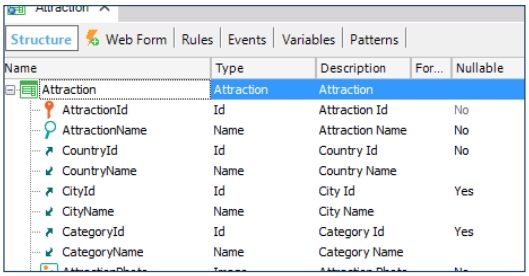
GeneXus™

Database update

Using a Transaction



Using a Business Component



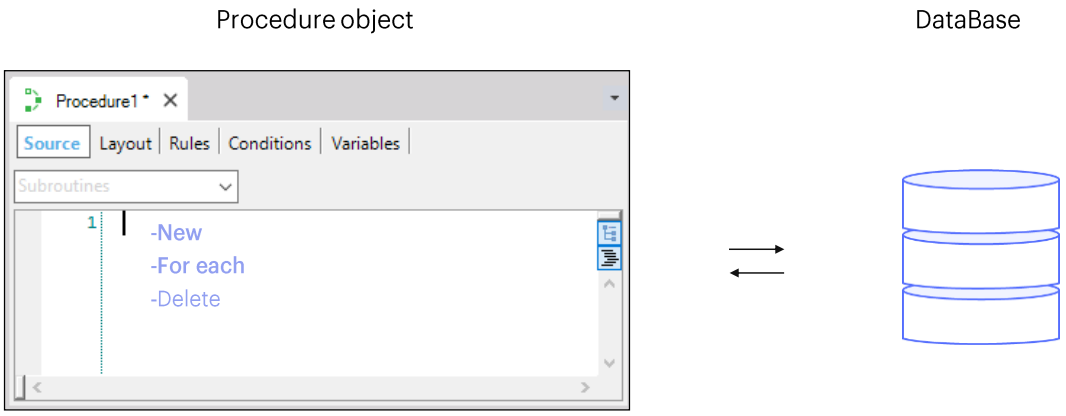

```
Event 'New Attraction'
    &Attraction.AttractionName = "Forbidden City"
    &Attraction.CountryId = find( CountryId, CountryName = "China")
    &Attraction.CityId = find( CityId, CityName = "Beijing")
    &Attraction.CategoryId = find( CategoryId, CategoryName = "Monument")
    &Attraction.Save()
    Commit
Endevent
```

So far, to update the database data we have used transactions in the 2 ways available to do so:

- Running their screens and entering data in an interactive manner
- Running them as Business Components, through a variable, without using their screens.

For example, to add a new attraction, in a web panel we could have placed a button and writing what we see above for its event.

Another way to update the database



Now we will learn about another option available to make insertions, modifications and deletions in the database.

We must take into account that what we'll see can only be used in procedure type objects, unlike the option that used a Business Component, which could be used from any object.

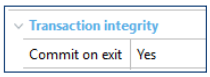
Only valid in procedures

INSERTION

New Table attributes

AttractionId	AttractionName	CountryId	CityId	CategoryId	AttractionPhoto
1	Louvre Museum	2	1	1	
2	The Great Wall	3	1	2	
3	Eiffel Tower	2	1	2	
4	Forbidden city	3	1	2	
5	Christ the Redemmer	1	2	2	

EndNew



Using this command, we can assign values to the attributes of one physical table to insert one record.

Procedures provide a command called New to insert records in a table.

Using this command we can assign values to the attributes of one physical table.

We're speaking of a table, not of a transaction, because not all the attributes in a transaction structure are included in the physical table.

For example, if we want to insert an attraction, in the Attraction transaction many attributes are declared in the structure but are not physically present in the ATTRACTION table.
Instead, they are in the extended table of the ATTRACTION table. We've included them in the structure to show them in the form or use them in the rules.

Attributes of the Attraction table and New

Transaction Attraction

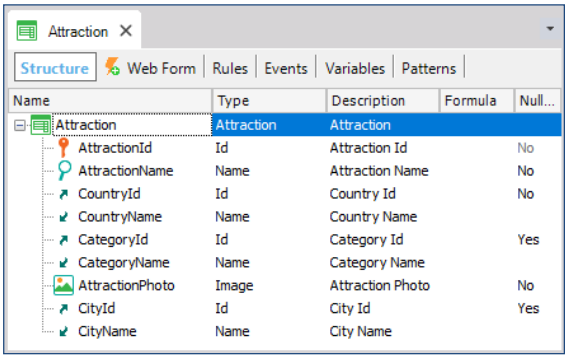
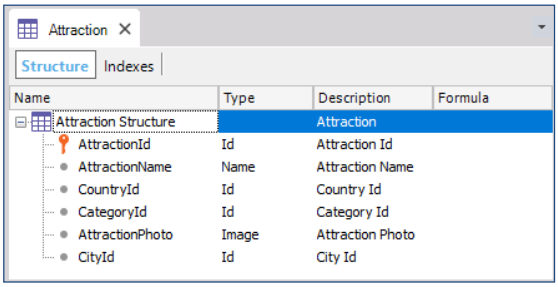


Table Attraction



The NEW command only allows assigning values to these attributes

If we select View/Tables and edit the ATTRACTION table, we'll see only the attributes that belong to the ATTRACTION table.

We can include these attributes in a New command and assign values to them. The New command will insert only one record in that table.

In procedures...

INSERTION

```
new
    AttractionName = "Forbidden City"
    CountryId = find( CountryId, CountryName = "China" )
    CityId = find( CityId, CityName = "Beijing" )
    CategoryId = find( CategoryId, CategoryName = "Monument" )
endnew
```

BASE TABLE

ATTRACTION

AttractionId: We don't assign a value to the AttractionId attribute because it is autonumbered

AttractionPhoto: The record will be inserted anyway, without the attraction photo

GeneXus will determine the physical table to insert the record, analyzing the attributes located to the left of the equal sign.

If they all belong to the same physical table, the record will be inserted in that table. Otherwise, we will be informed that it is not possible to select a table to make the insertion.

The table found by GeneXus is the base table of the New command. In this case: ATTRACTION.

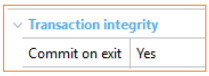
Note that we're not assigning an image to the AttractionPhoto attribute. In this case, the insertion will be made, even though the AttractionPhoto attribute will not be assigned and the attraction will be created without a photo.

Also, we could have left the attributes CountryId, CityId and CategoryId, which are foreign keys, with no values assigned. Or, rather, instead of using the Find formula to look for the corresponding IDs, we could type nonexistent IDs because the New command doesn't check for referential integrity. For this reason, we must be very careful when programming this command.

Insertion with New versus BC

Using the NEW clause in Procedures

```
new
    AttractionName = "Forbidden City"
    CountryId = find(CountryId, CountryName = "China")
    CityId = find(CityId, CityName = "Beijing")
    CategoryId = find(CategoryId, CategoryName="Monument")
endnew
```



Using Business Components

```
Event 'New Attraction'
    &Attraction.AttractionName = "Forbidden City"
    &Attraction.CountryId = find( CountryId, CountryName = "China")
    &Attraction.CityId = find( CityId, CityName = "Beijing")
    &Attraction.CategoryId = find( CategoryId, CategoryName = "Monument")
    &Attraction.Save()
    Commit
Endevent
```

When running the insertion, GeneXus will only check that no records are inserted with primary keys that already exist. Also, if there is a unique index defined over an attribute, it checks that no duplicate values are entered for this attribute. If this happened, the New command wouldn't do anything and wouldn't warn the user either. Next, we will see how to identify this situation and take the corresponding measures. The uniqueness control is the only control it makes, so NO referential integrity controls are made. Therefore, if we assign a nonexistent CountryId, there won't be a referential integrity failure, unlike what happens when the insertion is made through a Business Component. Remember that, in this case, uniqueness and referential integrity controls are made. In addition, the rules defined in the transaction are triggered.

When the insertion is made using a Business Component, just like it can be made from any object, we must explicitly run the Commit; that is to say, give the order to make the database keep the data changed since the last Commit operation.

Procedures have the property Commit on exit that is set to Yes by default. What effect does it have? If the procedure accesses the database, a Commit command will be added at the end of the code with no need for the developer to indicate it. In fact, it is not written in the object and is added to the generated program instead. That's why we don't need to explicitly add a Commit after the New command, unless we want to run the commit operation immediately.

GeneXus realizes that it has to access the database within a procedure when using New, For Each to update, delete (operations that we will see later). In these cases, it adds the implicit Commit; otherwise, it doesn't understand that the database needs to be accessed and doesn't add it. Therefore, database manipulation operations through business components will not result in implicit commit.

For this reason, when we type:

```
&BC.element1 = ...
&BC.element2 = ...
&BC.Save()
```

it doesn't add the Commit command because it doesn't realize that we want to make an Insert (not even if we used the Insert method directly). As a consequence, it handles the Business Component as if it were just any SDT. In this case, the Commit command has to be written explicitly.

If, within the procedure code, there was also a New, For Each that updates, or Delete, in any of those cases we wouldn't have to explicitly write the Commit, because it has already established a connection to the DB. If we only have the previous code in our procedure, then we would have to add the Commit command explicitly.

Only valid in procedures

INSERTION

```
New
    CategoryName = "Tourist site"
EndNew
```

BASE TABLE

CATEGORY

CategoryId (¿?)

In a New clause, to create a new record we assign values to the attributes that belong to a table record.

If we had the above New clause written in a procedure, where we only assign a value to the CategoryName attribute, obviously the base table will be CATEGORY, because it is the table where the CategoryName attribute is physically stored.

Once again, we don't assign a value to the CategoryId attribute because it is autonumbered.

Only valid in procedures

UPDATE

```
For each Attraction
    Where CityName = 'Beijing'
    Where CategoryName = 'Monument'
    CategoryId = find(CategoryId, CategoryName = 'Tourist site')
Endfor
```

BASE TABLE

ATTRACTION

The attributes of ATTRACTION's extended table can be updated in the For Each command (except the Primary Key)

Now we will see how we can update an existing value in the database.

To replace a value stored in an attribute -of a table record- with another value, we use a For Each command to access that record and give it the new value through an assignment.

In the example, the base table of the For Each command is ATTRACTION, because the base transaction is Attraction and we're running through all the attractions in Beijing that have the "Monument" category. We're changing the category of these ones to "Tourist Site".

Note that in this example we're updating many records, which are all those that meet the Where clause conditions. Also, we're updating the value of a single attribute, but they could be many. Even attributes of the extended table may be updated; for example, the name of the attraction category, the name of the attraction country, the name of the attraction city. We will see this in the following page.
Note that the New command only allows assigning values to attributes that are physically present in the table, because there we want to insert only one new record. In the For Each command, by contrast, we're always positioned in an existing record and from there we can edit all the records associated through an extended table.

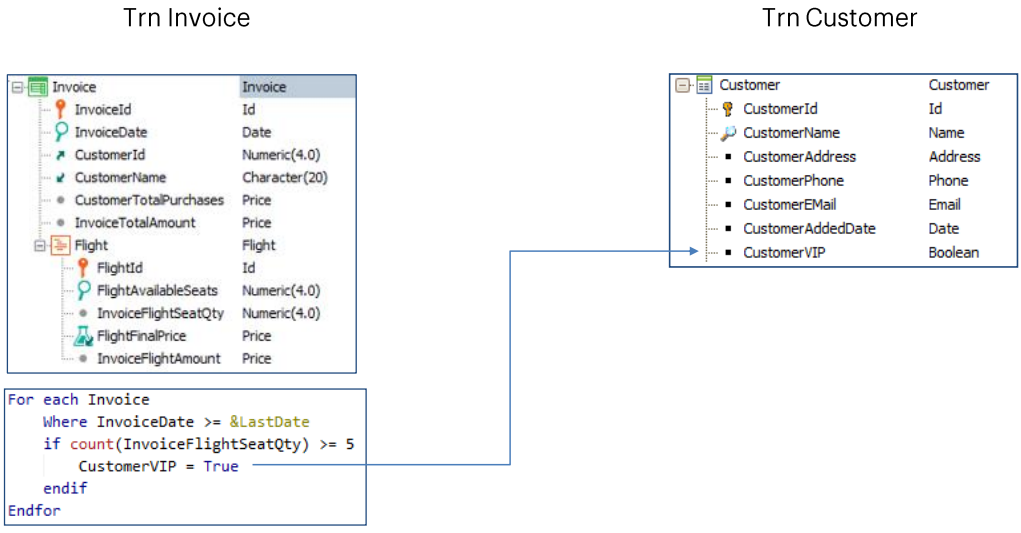
If we had a candidate key (that is to say, a unique index defined over one of the attributes) and inside the For Each command we were replacing its value with an existing one for another record, the update would not be

made. So, as we've said for the New command, here we will not see any warning either. Next, we will see how to capture this case and do something about it.

There's a restriction that doesn't allow changing, inside a For Each command, the primary key of the table being run through.

Only valid in procedures

UPDATE



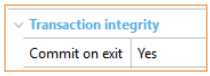
The travel agency wants to identify those customers that, starting from a certain date, in the same invoice have purchased more than five flights, and mark those customers as VIP. To do so, we add the CustomerVIP attribute to the Customer transaction; this attribute will be stored in the associated table.
To identify and mark those customers, we will have to run through the Invoice table and count the flights in each one of them. If that number exceeds five flights, we must update the value of the CustomerVIP attribute by setting it to True.

Here the base table of the For Each command is INVOICE, so the Count formula will only count the flights that belong to this invoice. If they are more than 5, note that we're updating the value of an attribute of the CUSTOMER table, which is in the extended table of the table we're running through.

Update using a For Each command vs BC

Using a For Each command in a Procedure

```
For each Attraction
    where CityName = "Beijing" and CategoryName = "Monument"
    CategoryId = find( CategoryId, CategoryName = "Torist Site")
endfor
```



✓ Uniqueness control of PK and CK

Using Business Components

```
For each Attraction
    Where CityName = "Beijing" and CategoryName = "Monument"
    &Attraction.Load(AttractionId)
    &Attraction.CategoryId = find( CategoryId, CategoryName = "Tourist site")
    &Attraction.Save()
Endfor
Commit
```

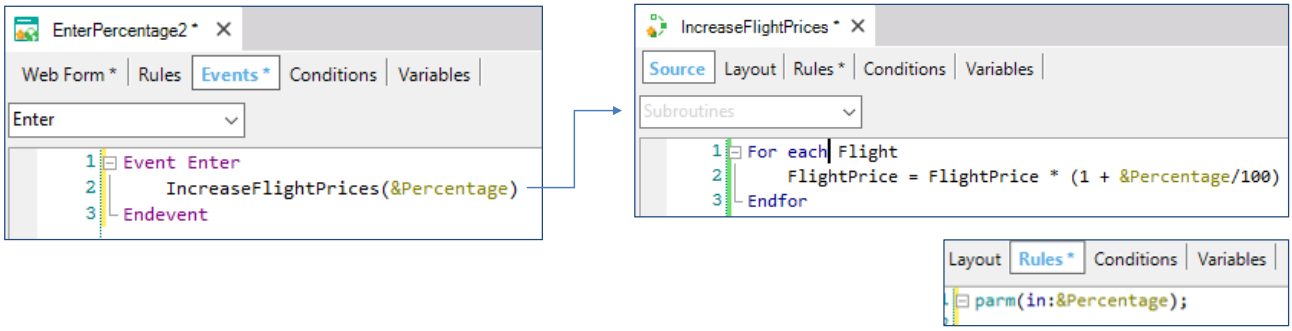
✓ Uniqueness control of PK and CK
✓ RI Control
✓ Rules

As we've said before, when a For Each command is used inside the procedure, the only control made by GeneXus will be a uniqueness control for the primary key and candidate keys.

By contrast, the solution that uses Business Components will also check for referential integrity. Therefore, if we assigned a nonexistent value to CategoryId, it would not allow making this update. In addition, it will run the business rules.

Remember what we've said about the New command for Commit on exit.

Procedure to increase prices



We want to prompt the user to enter a percentage and recalculate the price of all flights by applying this increase percentage.

To do so, we create a web panel to ask this data from the user, and from there we invoke a procedure that will effectively update the flights table, to change the price value according to the percentage received in a parameter.

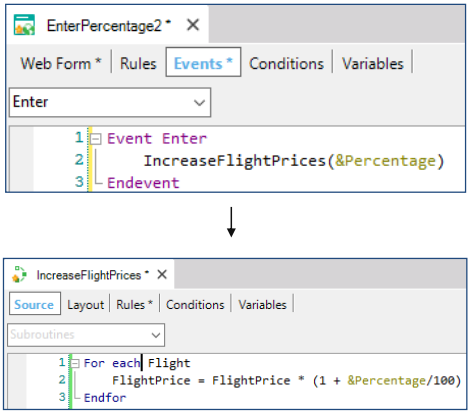
To make this update, we use the For Each command to run through the FLIGHT table and replace the FlightPrice attribute value with the result of the expression displayed.

Comparing alternatives

Using Business Components

```
Event Enter
    For each Flight
        &BCFlight.Load(FlightId)
        &BCFlight.FlightPrice = &BCFlight.FlightPrice * (1+ &Percentage/100)
        &BCFlight.Save()
        If &BCFlight.Success()
            Commit
        Else
            Rollback
        Endif
    Endfor
Endevent
```

Using a Procedure



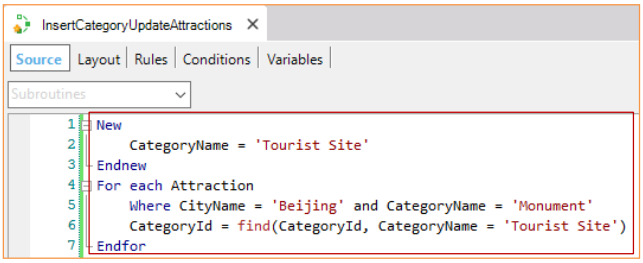
Let's compare the two alternatives we use to update the database.

In the first alternative we implemented, in the Enter event of the EnterPercentage web panel we type the For Each command and update the database using the Flight transaction as a Business Component.

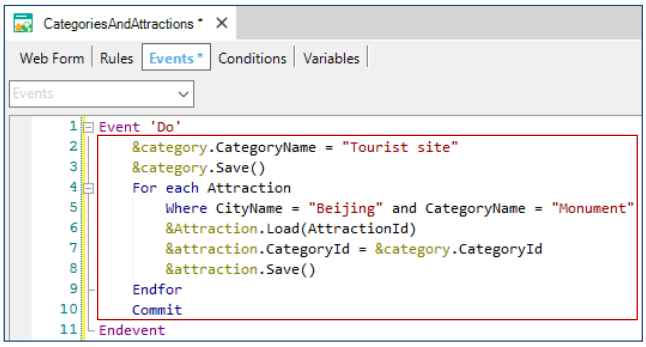
In the second solution, the Enter event of the EnterPercentage2 web panel only contains a call to a procedure, which receives the percentage value, navigates all the flights with a For Each command and, for each one of them, calculates and assigns the new price.

What are the differences, advantages and disadvantages of solving this in one way or the other?

Example Insert Category + change in Attractions



Exclusivamente en Procedure



Puede estar en cualquier objeto

Here we can see both solutions to insert a new category and replace the category value of certain attractions with this category.

Note that the code in the web panel event is run in any GeneXus object, and the code in the procedure is only valid inside a procedure object.

Let's examine the procedure:
Since CategoryId is autonumbered, the New command will never fail. We're using a Find formula to find the ID of the category called "Tourist Site" that we've just inserted with the New command. If we had made a mistake when typing the category name, the Find formula would not have found the record and would have returned an empty value, which is 0 for a numeric value. In this case, we would be changing the category ID of the attractions in Beijing that were monuments to value 0. If the CategoryId attribute didn't accept null values, an inconsistency would be created in the database. Remember that no referential integrity checks are made when the For Each command is used to make an update, unlike what happens when a Business Component is used, because in that case it would check for referential integrity. In our case, CategoryId accepts null values, so we won't have any consistency problems when using this option through a procedure.

Updating values through Business Components is always the safer option, because they will make the same controls as the New and For Each commands, and even more: referential integrity checks in addition to the rules stated in the associated transaction. Consider that, even if you were

careful when programming the procedure, its associated reality may change at any time. For example, the need to add error rules to the transaction that you didn't think of when programming the procedure. The use of a Business Component guarantees that these rules will be included in all checks.

Only valid in procedures

DELETION:

```
For each Category
    where CategoryName = "Tourist Site"
    Delete
Endfor
```

```
For each Attraction
    Delete
Endfor

For each Category
    Delete
Endfor
```

We've seen how procedures allow us to insert and update records in the database. Now, let's see how to delete records.

To delete records we have the Delete command that is used inside a For Each command.

Basically, since the Delete command removes the record from the table we're positioned in, the For Each command is used to access that record.

In the first example we see that we're navigating the Category table, filtering by the "Tourist Site" category; therefore, we're only deleting one record with the Delete command.

But we may have also deleted all the attractions and all the categories (if we wanted to empty those tables).

If we delete the categories first and the attractions later, since the database checks for referential integrity by default, when we try to delete the first category, if it has a related attraction it will fail and the program will fail too, as we will see next.

Using special commands (NEW, FOR EACH, DELETE)

Data validations: only uniqueness, not referential integrity.

Only valid in procedures.

Usando business components:

Data validations: both uniqueness and referential integrity.

Transaction rules are triggered.

Valid in any object.

When we use Business Components, the data that is going to be updated in the database is checked for consistency, and the rules stated in the transaction being run as a Business Component are run.

Those rules that generate messages, such as msg and error, are also triggered and the corresponding messages are saved in a collection that can be run through and printed.

In a procedure, none of these actions are performed.

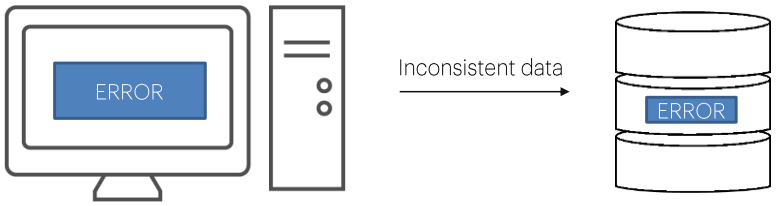
Something that we've talked about and it's worth mentioning again, is that even if the For Each command can be used in a web panel, it cannot be used to change the database directly by assigning values to attributes, and that it can only be made from a procedure object. It's not possible to code a New command in a web panel either, or include a Delete command inside the For Each command. This can only be done in procedures.

However, in any object, it is possible to change the database using Business Components.

We must take into account that procedures don't check the data being assigned for consistency. They only check primary and candidate keys for uniqueness.

Using special commands (NEW, FOR EACH, DELETE)

We will be responsible for assigning valid and well related data.



When using these commands to make direct updates in procedures, it is our responsibility to assign or insert data that is consistent with the rest of the stored data.

In the previous example of the New command, we can assign any value to CategoryName because this data is not related to any other tables.

In the example where we updated the attractions category, by contrast, we were assigning a new value to an attribute that was a foreign key: CategoryId. If the value assigned didn't exist as a category identifier in the CATEGORY table, the procedure wouldn't validate it, and that's why we may be entering inconsistent data.

Since databases check for the consistency of interrelated data, when the user runs the application and tries to assign an inconsistent value, the database will reject the operation and the inconsistent data will not be saved.

However, the program will stop running and this is not user-friendly at all.

Therefore, if we use procedures to update the database, we will be responsible for assigning valid and well related data.

Using special commands (NEW, FOR EACH, DELETE)

If data validations only check for uniqueness: where do we program the actions to run if attempts are made to duplicate the primary key or candidate key?

```
new
    AttractionName = "Forbidden City"
    CountryId = find( CountryId, CountryName = "China")
    CityId = find( CityId, CityName = "Beijing")
    CategoryId = find( CategoryId, CategoryName = "Tourist Site")
when duplicate

endnew
```

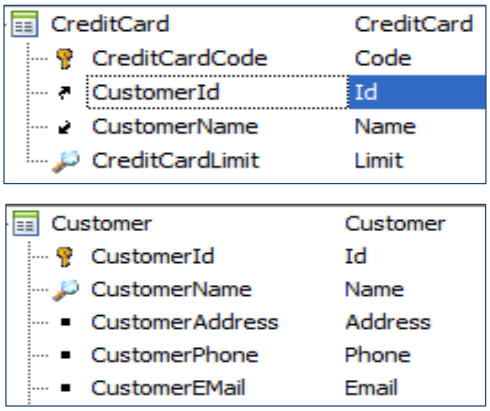
If AttractionName is a candidate key, that is to say, it can't be repeated, and we run a New command in order to insert a new record (with autonumbered ID) by assigning the value “Forbidden City” to AttractionName and there already exists a record with that value, we can do something about it, which is to program it inside the When duplicate clause of the New command.

Using special commands (NEW, FOR EACH, DELETE)

Duplication of the primary key or candidate key

```
for each CreditCard
    where CreditCardCode = &creditCardcode
    CustomerId = &newCustomerId
when duplicate

endfor
```



Likewise, suppose that we have two transactions called Customer and CreditCard that have a 1 to 1 relationship: that is to say, CustomerId has been set as a candidate key (through a unique index), and in a procedure that receives two variables through a parameter -the identifier of a new customer and the identifier of an existing credit card- a For Each command is programmed to change that card's customer for the new one. If the database already has another card for the same customer, the For Each command will not make the update. If we want to perform an action in that case, we can use the When duplicate clause.

To learn more about this clause, please visit our wiki: http://wiki.genexus.com/commwiki/servlet/wiki?24843,When+duplicate+clause.

The attributes included in the When duplicate clause will not be taken into account when determining the base table of the For Each command in which they are located.

For each sintax

```
For each   BaseTransaction
      skip expression1 count expression2
      order  att1, att2, ... , attn [when condition]
      order  att1, att2, ... , attn [when condition]
      using DataSelector( parm1, parm2, ... , parmn)
      unique att1, att2, ... , attn.
      where  condition [when condition]
      where  condition [when condition]
      where  att  IN DataSelector( parm1, parm2, ... , parmn)
      blocking N
        main code
When duplicate
      ...
When none
      ...
endfor
```

The blocking clause allows updating the database in groups of N records, to reduce the number of database accesses and improve performance. We won't talk about it in this course. For more information, read the article here:

http://wiki.genexus.com/commwiki/servlet/wiki?4837,Blocking+clause+in+a+%27For+each%27+command.

For more information about the When duplicate clause, which is valid for both the For Each and New command, click on this link: http://wiki.genexus.com/commwiki/servlet/wiki?24843,When+duplicate+clause.

For the full syntax of the New command, click on this link: http://wiki.genexus.com/commwiki/servlet/wiki?6714,New+command.

Remember that the attributes included in the code written inside the When duplicate clause will not be taken into account when determining the base table of the For Each command in which they are located.

training.genexus.com
wiki.genexus.com
training.genexus.com/certifications