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

- Running their screens and interactively entering data
- And executing them as Business Components, through a variable, without using the screen.
Now we will learn another alternative to make insertions, modifications and deletions in the database.

We must take into account that what we'll see next can only be used in objects of procedure type.
unlike the alternative we saw, which involved the use of BC and could be used from any object.

In procedures
we have 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 mean a table, not a transaction
because not all the attributes of a transaction structure are included in the physical table.

For example, if we want to insert a flight, the Flight transaction has many attributes stated in its structure and are not physically included in the FLIGHT table. Instead, they are in the extended table of the FLIGHT table. We have included them in the structure to show them in the form or use them in the
rules.

If we go to the Tables node and locate the FLIGHT table

we see only the attributes **belonging to the FLIGHT table**.

We can include these attributes within a New command and assign values to them.

GeneXus will determine the physical table in which the record will be inserted, examining the attributes **to the left of the equal sign**.
If they all belong to the same physical table, the record will be inserted in said table.

Otherwise, we will be informed that the table in which to perform the insertion cannot be determined.

The table found by GeneXus is called **base table of the New command**.
Let’s go back to assigning values to attributes.

Note that we haven’t assigned a value to the primary key attribute FlightId.

That’s because FlightId has its Autonumber property set to True, and therefore the database will assign its value automatically and consecutively.
Since the New command inserts one record in one table,

we can only assign values to the attributes belonging to that single physical table.
That is, we can't assign values to attributes that belong to different physical tables.

We can skip assigning a value to an attribute of the table in which we're performing insertions, whether because it's not necessary (such as FlightId because it's autonumbered), or because we want to leave an attribute unspecified.
For example, if we don’t assign a value to FlightPrice,

the inserted record will be left without a price –in other words-, with an empty price or not specified.

We must remember that procedures don’t check the consistency of the data we assign.
For example, in these assignments, we can assign any value to the flight price, because this detail is not related to other tables.

However, the rest of the assignments involve country and city identifiers... so we must be careful and assign values that exist in the tables storing the countries and cities, respectively.

In this example, we have assigned value 1 as country identifier in both cases.
knowing that in our country table Brazil is saved with identifier: 1

The cities 1 and 2 that we’re assigning
also have been recorded: they are Rio de Janeiro and Sao Paulo, respectively.

However, if we had assigned a country or city identifier value that hadn't been recorded in the corresponding table
the procedure wouldn’t validate it, so we may be entering inconsistent data.

Since databases check 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 working and this isn’t very user friendly.

Therefore, if we use procedures to update the database, it will be our responsibility to assign valid and well related data.
Here we explained it for an insertion using the New command,

```
New
  FlightPrice - 300
  FlightDepartureCountryId - 1
  FlightDepartureCityId - 1
  FlightArrivalCountryId - 7
  FlightArrivalCityId - 4
Endnew
```

but the same should be taken into account when updating data or deleting records.
Let’s see how we can update an existing value in the database.

To replace a value stored in an attribute with another value, we navigate its base table with a For Each command

```
For each
    FlightPrice = FlightPrice * 1.10
End for
```

and give it a new value through an assignment.
We can make assignments to attributes of the base table we’re navigating and to those of the extended table.

In this example, since the only attribute included in the For Each command is FlightPrice,

the base table navigated by GeneXus is: FLIGHT
Since no filters have been defined, all the table records will be navigated. For each flight, we update its price, and in this case we increase it by 10%.

Now we look at GeneXus to put this into practice.

We will solve the same feature that we had implemented, using the Business Component concept and we will be able to compare both solutions.

Remember that we had a web panel called “EnterPercentage”
in which the travel agency user could enter a percentage \(<\text{highlight the variable}>\) and press the confirm button,

\[
\text{to execute this code}
\]
to navigate all flights and increase the price of each flight.

We right-click on the tab with the object name

and select “Save As” to obtain a copy of the object with another name.

We call it “EnterPercentage2”
Here we have the new web panel, and so far it is identical to the previous one.

We open the event associated with the Confirm button.

Even though this event has a For Each command, the possibility of using it to update attributes by assigning values to them is only allowed in procedure objects.

So, we delete this code
and in the Enter event we only include a call to a procedure, which is the one that will update the database.

We create the procedure object, call it “IncreaseFlightPrices”

and in the source we write the same code we proposed in this example.

We type For Each... Flight... and press Control—Enter to autocomplete and select FlightPrice.
To the FlightPrice attribute, we assign the previous value multiplied by 1.10.

As we have explained, the base table of this For Each is FLIGHT.

All flights are navigated and for each one of them we’re increasing its price by 10%.
Now we call this procedure from the web panel.

```
# Event Enter
IncreaseFlightPrices.call()
# EndEvent
```

We find the procedure in the Folder View and drag it to the Web Panel. We type a period and select the call suggested in this context menu.

This Enter event will be executed when the user presses the button associated with it.
But before running what we’ve done, look at a small detail: in the web panel, the user can type in this variable

a certain increase percentage, and by pressing the button
we call the procedure
that increases prices a fixed percentage of 10%.

Initially, we set the fixed percentage of 10% as an example... but now we want to consider the percentage entered by the user in the web panel.
How do we make the **procedure** know the value entered in the **web panel**?

In the web panel we know the value of the &Percentage variable, but not in the procedure.

What we’ll do is send the value saved in the variable when the procedure is called.

To achieve this, inside the call parenthesis we include the &Percentage variable.
We should receive it in the procedure. We go to the procedure and in the rules section we type parm followed by parenthesis.

We have to receive the variable inside the parenthesis, but the variable is not defined here. We can define it using the same name with which it is defined in the web panel or with another name. The data type must be identical to the data type of the variable sent from the web panel to the procedure.

Note that the data type of the &percentage variable in the web panel is: Numeric(3)
so, in the procedure, we will define it in this way.

Now in the rules section, inside the parenthesis of the Parm rule, we include the variable we received.

All rules must end with a semicolon in every object, so we type it and save.

Let's open the source to include the &Percentage variable in the calculation.
We press F5...

look at the current price of flights...
run the web panel EnterPercentage2,
type a 50% increase,

and confirm.

Now we display the flight prices and see that they have increased by 50%.
We must bear in mind

that in a single For Each command we can update several physical tables.

Specifically, a For Each command always has a base table that it navigates and can change

but it can also change the entire extended table of said base table.
In this example, the base table of the For Each command is Flight, since within the For Each only the FlightPrice attribute is included.

As we know, each flight has 1 country of origin, 1 country of destination, 1 city of origin and 1 city of destination. Therefore, in this For Each command we could change the details of those countries and cities, if necessary. Or, for example, if each flight had an airline,
within this For Each we could change the details of the airline associated with each flight.

We’ve seen how procedures allow us to insert and update records in the database. Now, we’ll see how to delete records.

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

Basically, we need to navigate the table from which we want to delete one or more records, and include the Delete command within the For Each.

In the example we can see that we’re navigating the Country table, filtering by the value of a certain country, and therefore we’re deleting 1 record with the Delete command. But we could also have deleted all the countries entered in the Country table.
Since we didn’t need to filter certain records, nor to order or print them, the For Each command offers the **Defined by** clause to make reference to an attribute that helps determine the base table.
As we’ve said before, procedures don’t take into account the data related in other tables.

This For Each command that deletes all countries upon execution, could leave tourist attractions that make reference to countries that have been deleted.

Because the database controls the consistency of interrelated data, it will reject the operation and the program will stop working.
Therefore, when using procedures it is our responsibility to delete, assign or insert data that is consistent with the rest of the stored data.

Lastly, let's compare the 2 alternatives we used to update the database.

In the first alternative,
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 2nd solution, the enter event of the EnterPercentage2 web panel only contains a call to a procedure.

The procedure receives the percentage value, navigates all flights with a For Each command
and calculates and assigns the new price to each one of them.

What are the differences, advantages and drawbacks of either way?
Something that we’ve seen and should go over again is that even though the For Each command can be used in a web panel, it can’t be used to change the database directly by assigning values to attributes, as it can only be done from a procedure object. Nor is it possible to code a New command in a web panel, or include a Delete command within the For Each. **This is only valid in procedures.**

**However, in any object we can** change the database with business components.

In addition, when we use business components

**USING BUSINESS COMPONENTS:**

√ Data validations are performed

the consistency of the data to be updated in the database is validated
triggering the rules stated in the transaction executed as a business component.

The 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.

None of this can be performed in a procedure.

Thus we’ve seen several alternatives to update the database in a GeneXus application.