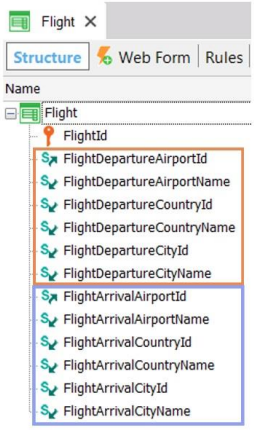
Subtypes

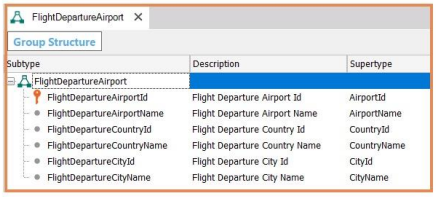
Multiple References and Specialization

GeneXus™

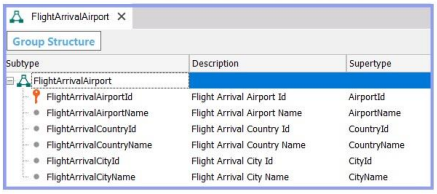
Review

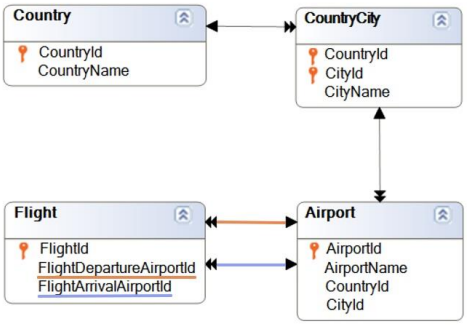


Subtype Group: FlightDepartureAirport



Subtype Group: FlightArrivalAirport





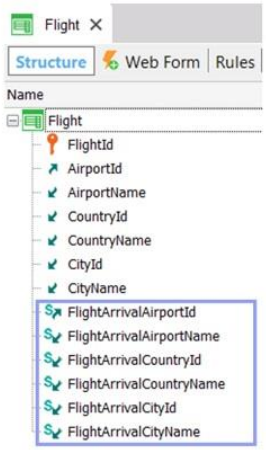
In the previous video, we saw the need to define groups of subtypes, since a transaction had a double reference to the same actor of reality.

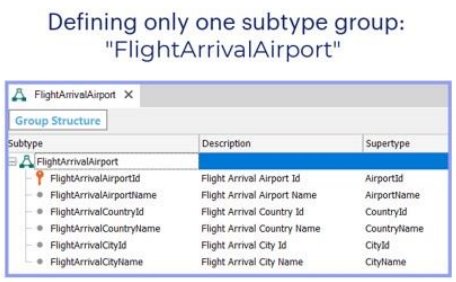
This was the case with the Flight transaction, which had a departure and an arrival airport for each flight. We could not include the same attribute, AirportId, in the transaction structure twice, as GeneXus showed an error for adding a duplicate attribute name. For that reason, we decided to define two groups of subtypes: “FlightDepartureAirport” to identify the departure airport, and “FlightArrivalAirport” to identify the arrival airport.

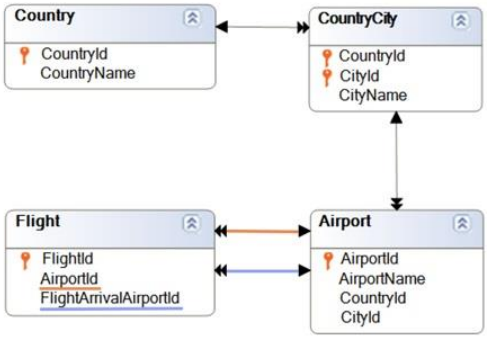
Since we wanted to infer the country and city of each airport, in each group we also defined subtypes of the attributes corresponding to country and city, as well as the name of the airport.

Therefore, when we name FlightDepartureCountryName in the Flight transaction, we know that it will be a CountryName inferred through the departure airport: FlightDepartureAirportId. These subtypes have been defined within the same group and therefore the association and relationship between them has been established.
And when we name FlightArrivalCountryName in the Flight transaction, we know that it will be inferred through the FlightArrivalAirporId attribute.
That is to say, there is no ambiguity. There are two completely different ways to get to Country from Flight.

Review

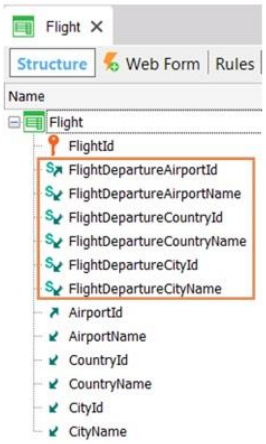


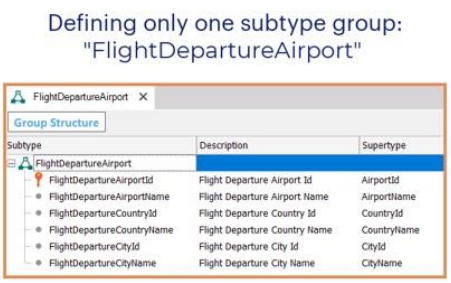


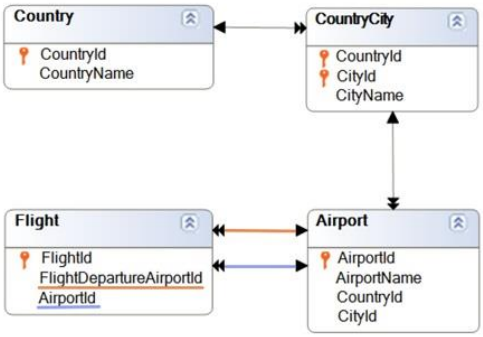


Another option was to leave the AirportId name attribute for the role of the departure airport and define a group of subtypes to identify the arrival airport;

Review



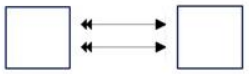




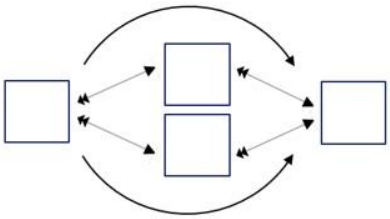
... or else leave the attribute named AirportId for the role of the arrival airport and define a group of subtypes to identify the departure airport.
In both options, the data model will reflect the same relationships as in the previous solution.

Multiple references

Direct



Indirect



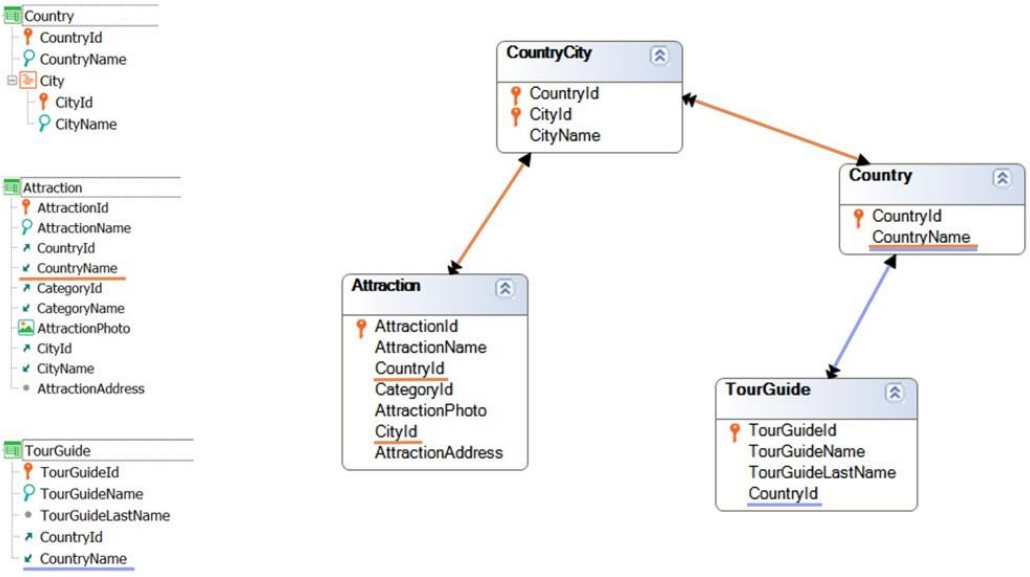
In the previous video we then saw the case of multiple references from one table to another directly related to it.
However, these references don't have to be direct.

From one table there can be two paths to another, as it is an indirect relationship, so subtypes will be needed to differentiate them.

New Transaction: TouristGuide

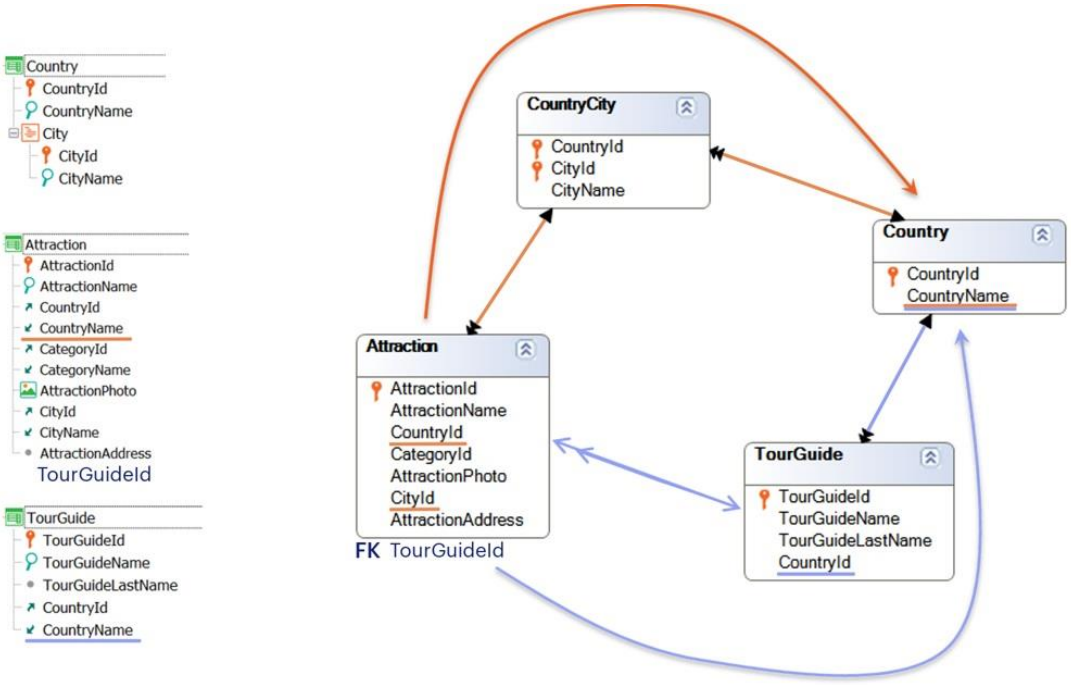
Suppose we have to add a transaction to record the information of the tour guides. Each guide has a specific nationality, and for that reason we have added the CountryId attribute to its transaction structure.

New Transaction: TouristGuide



If we look at the table diagram, from the Attraction transaction we can infer the CountryName corresponding to that attraction, since it is in its extended table. Likewise, from TourGuide we can also infer its own CountryName, that is, the guide's country.

TouristGuide: Multiple references



If we now link the Attraction and TourGuide entities by adding to the first one the attribute identifying the guide, TourGuideId, to indicate that a tourist attraction has only one guide assigned to it, how are the tables now related?

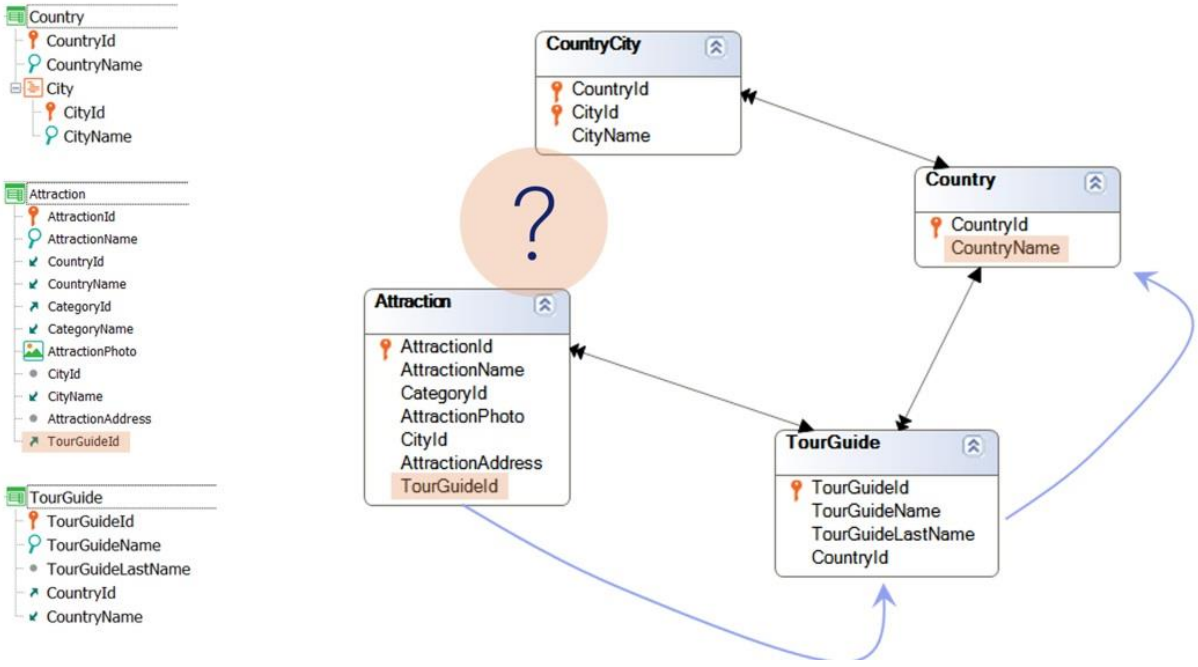
TourGuideId will be a foreign key in Attraction to the TourGuide table, so the relationship will be marked with the sky blue arrow. That may lead us to think that from Attraction there are now two ways to infer CountryName, which means that there is an ambiguity.

In other words, since GeneXus has the CountryName attribute in Attraction, where does it infer it from? From the attraction city or the attraction tour guide? If both values were the same, it wouldn't matter, but in this case they don't have to match. The country where the attraction is located doesn't have to match the tour guide's home country.

In order to differentiate both CountryName roles, we will need to use subtypes.

But in this case we will not only need them to differentiate the roles.

TouristGuide: Multiple references

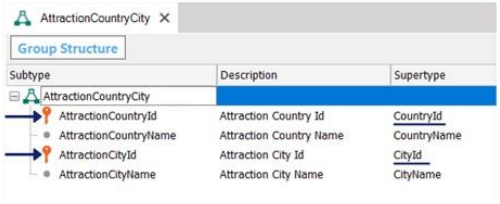


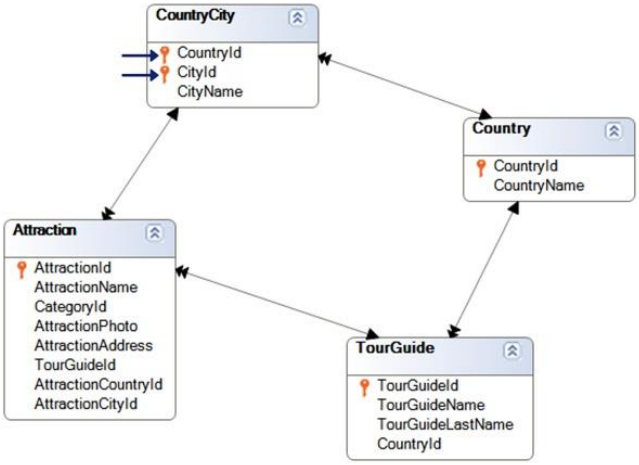
Let's look at the table diagram after adding the TourGuideId attribute in the Attraction transaction: the relationship between Attraction and CountryCity disappears, as CountryId is no longer a foreign key in Attraction. Note that it is no longer in the table; that is, it will be an inferred attribute. But, inferred from what? From TourGuideId!

Remember that GeneXus first normalizes the tables. That is, based on the attribute names, together with the identifiers, it determines which attribute is placed in each table and the relationships between them. As TourGuideId is shown in Attraction –that is the TourGuide identifier– and at the same time CountryId is shown in TourGuide –the Country identifier– it understands that given a TourGuideId in Attraction the CountryName is inferred from it, going through the TourGuide intermediate table.

Therefore with this transaction design we can't indicate which is the country of the attraction, because the CountryName will be that of the tour guide.
We have no choice but to use subtypes in order to make Attraction have its own country, independently from the country of the tour guide.

TouristGuide: Multiple references





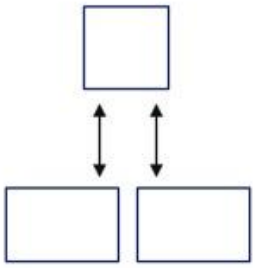
We will create a group of subtypes to represent the country and city of the attraction, since it is Attraction where the problem occurs.
Note that now GeneXus correctly represents the relationships in the table diagram; in addition, the CountryName attribute is now inferred from TourGuideId without ambiguity. The attribute that represents the attraction country and from which it is inferred will be the one called AttractionCountryName, a subtype of CountryName that belongs to the group AttractionCountryCity.

Also, note that this group has two primary attributes: AttractionCountryId and AttractionCityId, which correspond to the primary key of the CountryCity table, according to the supertypes indicated: {CountryId, CityId}.

This solution solves the problem, but we must take into account that if there are other objects that already use the original attributes CountryId and CountryName to search for the attraction country, we may have to modify them in those objects and use the subtype attributes AttractionCountrId and AttractionCountryName, respectively.

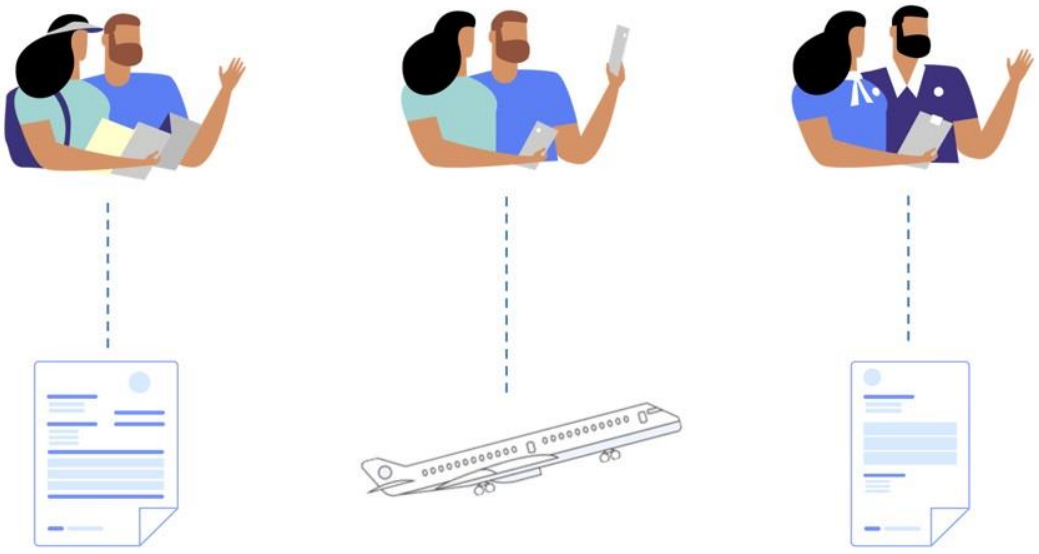
There are other possible solutions that will not be studied in this course.

Specialization



Now let's look at another case of use of subtypes, which we call specialization, in which there is a transaction that records general information, and then there are transactions with particular information, which are a specialization of that other one.

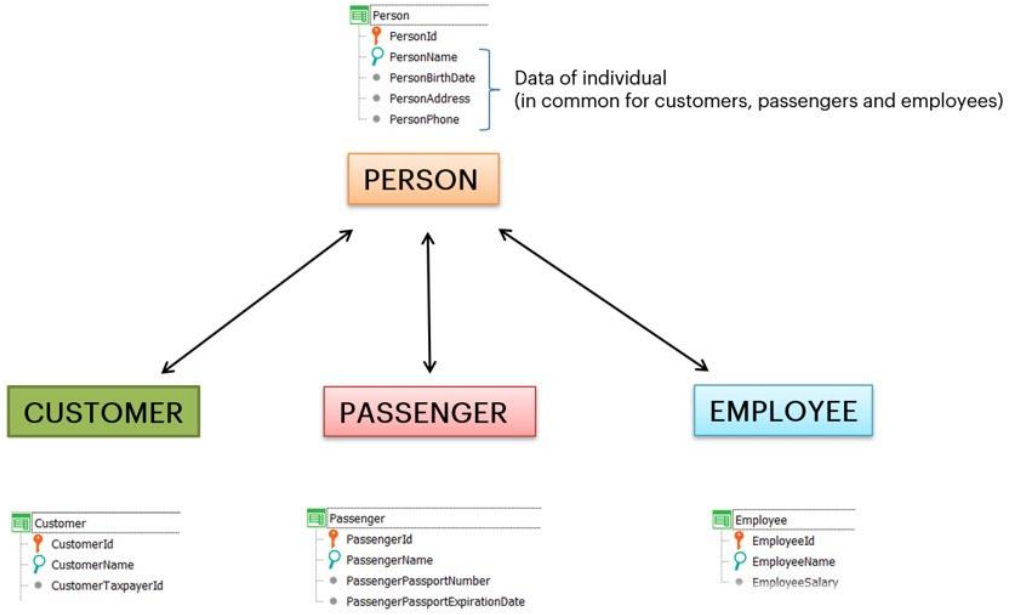
Specialization



Suppose that the travel agency needs to handle specific information about the customers to whom it sells tickets and tour packages (e.g. their taxpayer number at the state tax office, if any), specific information about the passengers (e.g. their passport number and validity) and also specific information about the agency's employees, for whom it must record, for example, their salary.

In other words, the travel agency will invoice customers, provide seat reservations for passengers, and issue pay slips to employees.

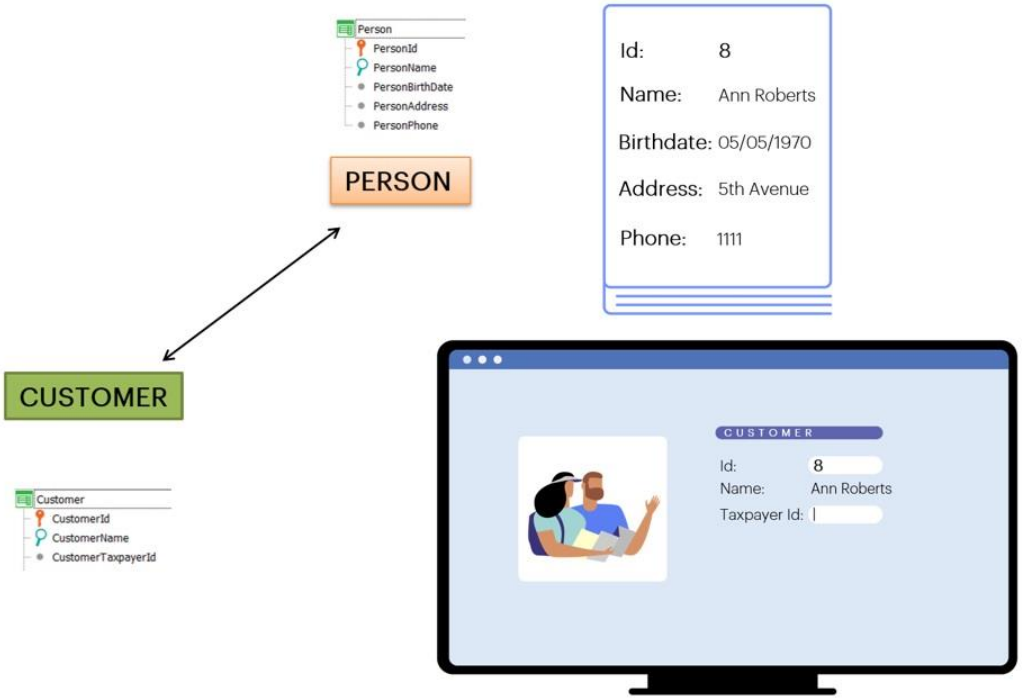
Specialization



So, instead of just a Customer transaction, we could define a transaction called Person that handles information that is common to all people (e.g., a name, date of birth, address, phone number, etc.), and transactions that are Person's specializations, because customers, passengers, and employees ARE people.

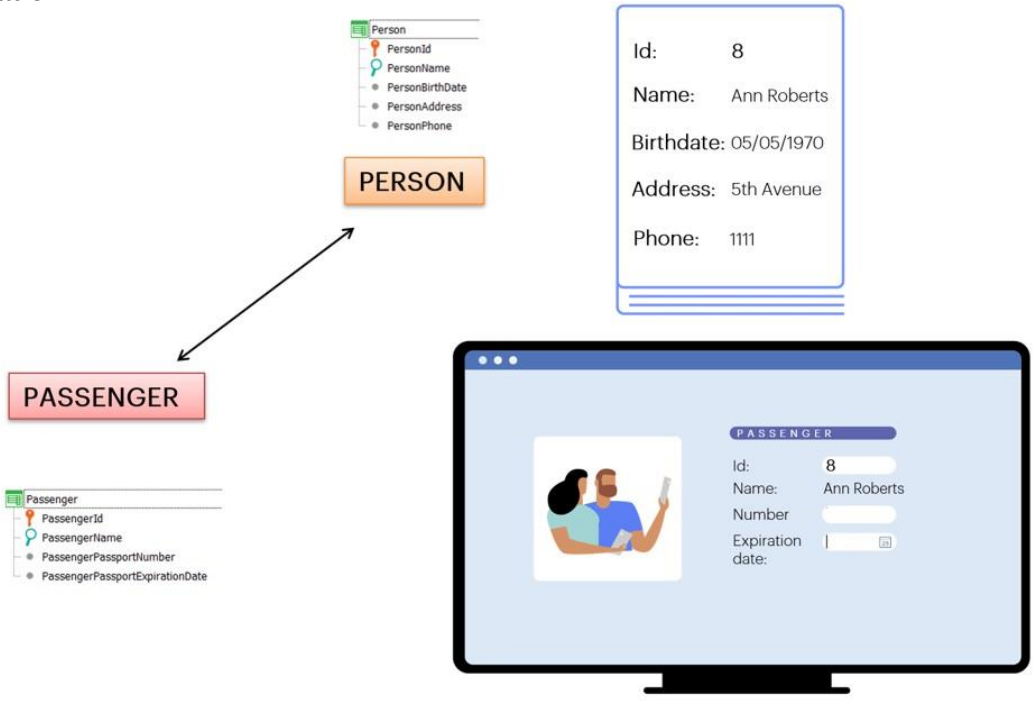
Each specialization will have its own specific data (Customer will have a taxpayer number, Passenger will have a passport number and expiry date, and Employee will have a salary).

Specialization



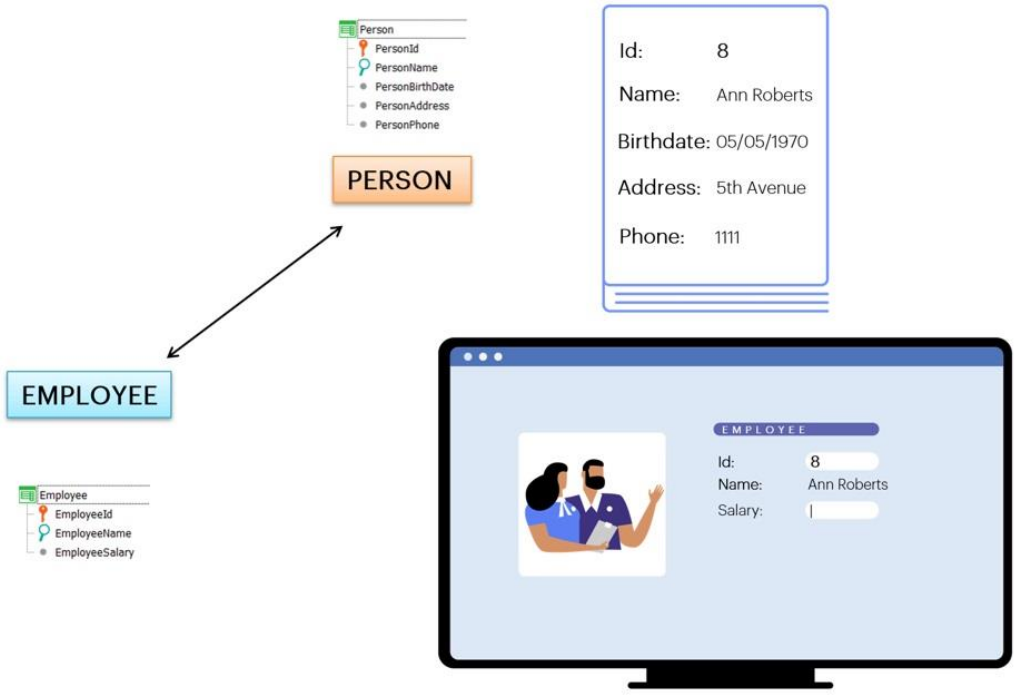
We want the customer ID to exactly match that of a person, to reflect that the customer is a person. That is, if the person with ID 8 is called Ann Roberts, and she was born on 05/05/1970, when entering her information as a customer, the user needs to be able to type the ID 8 in the Customer transaction. Also, when leaving the field the name Ann Roberts should be shown in order to enter the taxpayer number in the CustomerTaxpayerId attribute.

Specialization



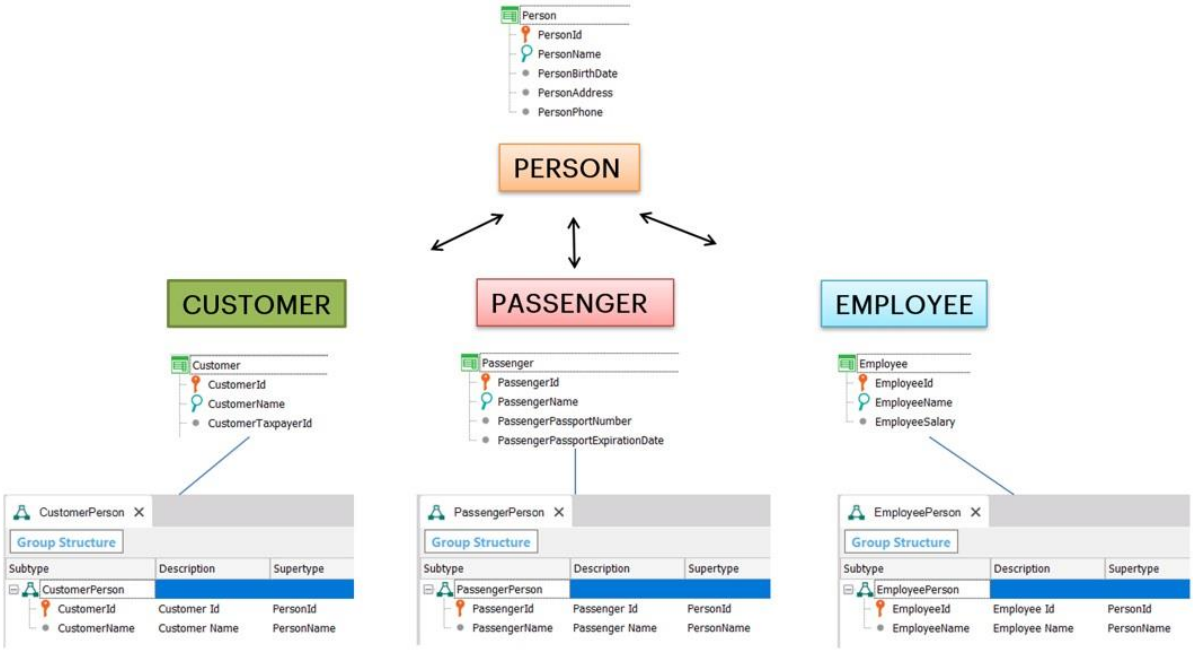
Likewise, if the Passenger transaction is executed, when the user types the value 8 in PassengerId, we want Ann Roberts to be inferred in PassengerName, and the user to be able to assign the passport number and expiry date in the specific attributes (PassengerPassportNumber and PassengerPassportExpirationDate).

Specialization



And for the employees as well.

Specialization



If we simply define as primary keys of Customer, Passenger and Employee, the attributes CustomerId, PassengerId and EmployeeId respectively, without relating them in any way to PersonId (such as CustomerName, PassengerName and EmployeeName without relating them to PersonName), we will not get what we are looking for. For GeneXus they will be completely independent transactions.

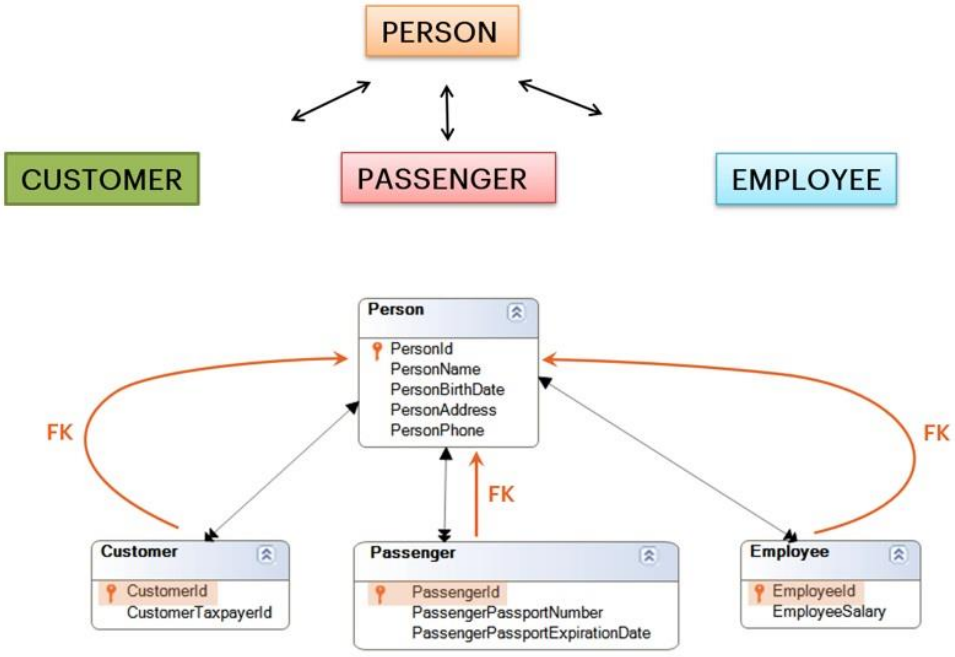
To relate them, we are going to define groups of subtypes, to indicate that customers, passengers and employees must be valid (i.e. previously registered) persons.

We create a CustomerPerson group, where we define CustomerId and CustomerName as subtypes of PersonId and PersonName, respectively.

Also, another one called PassengerPerson, where we define PassengerId and PassengerName as subtypes of PersonId and PersonName, respectively.

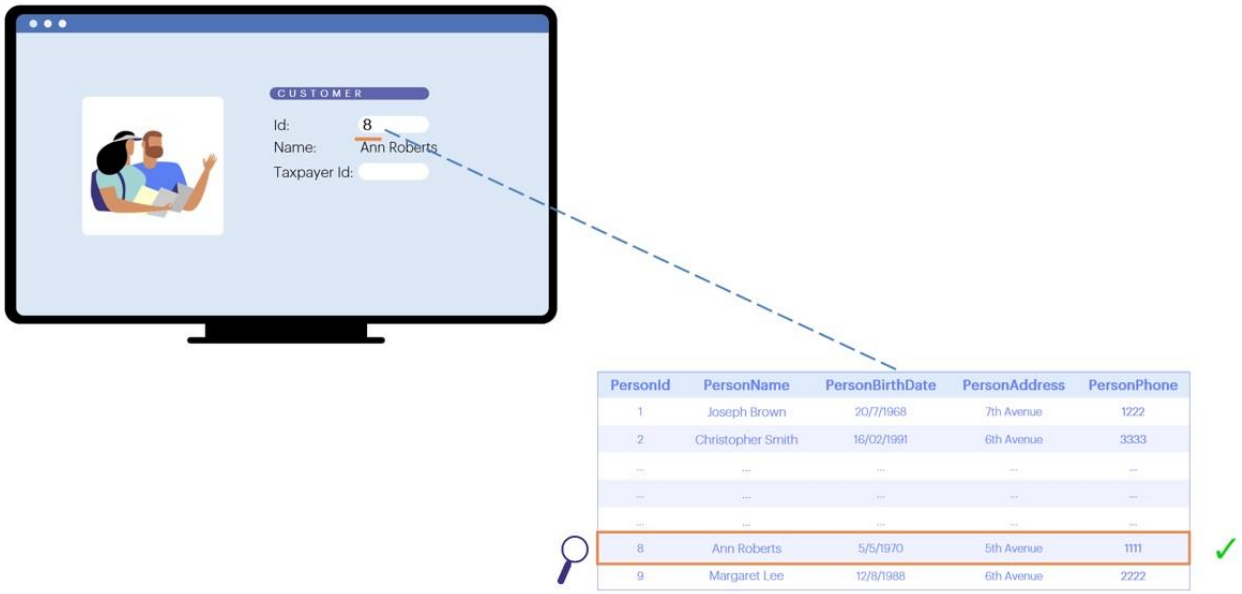
Lastly, a group called EmployeePerson, where we define EmployeeId and EmployeeName as subtypes of PersonId and PersonName, respectively.

Specialization



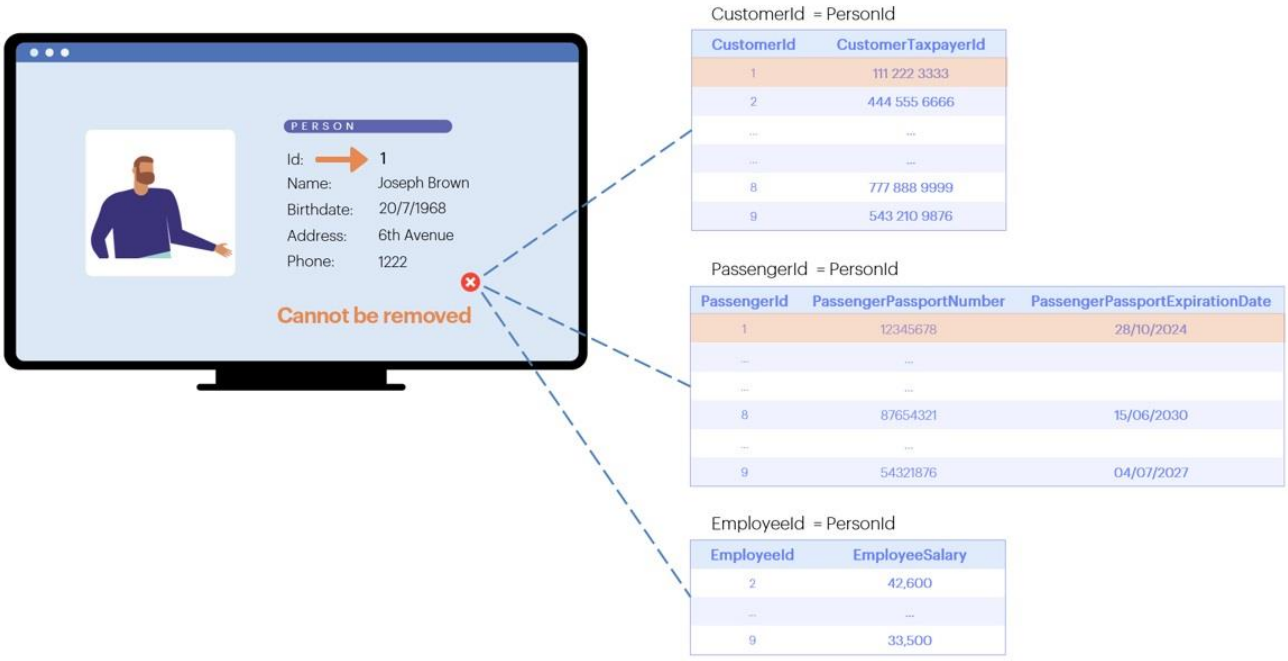
By doing this, the attributes CustomerId, PassengerId and EmployeeId, besides being the identifiers of the Customer, Passenger and Employee tables, respectively –and therefore, their primary keys– will also be foreign keys of the Person table. In this case, GeneXus will control the consistency of the data.

Specialization



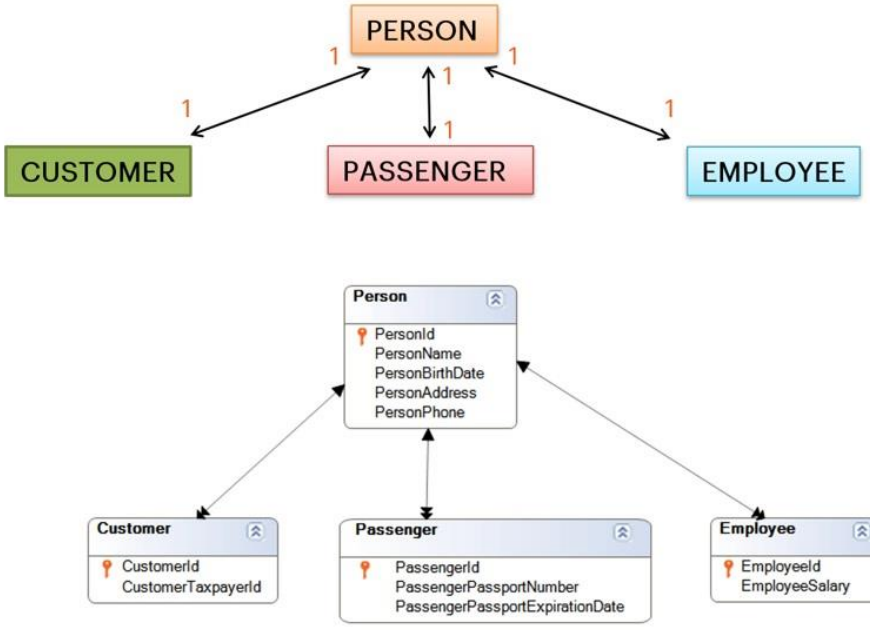
This means that when the user enters a value in the ID of any of the three transactions (Customer, Passenger or Employee), a record with the same ID value will be searched for in the Person table.

Specialization



Similarly, if you want to delete a person through the Person transaction, it will be checked that there is no record in Customer where CustomerId matches the PersonId you are trying to delete, or a record in Passenger where PassengerId matches the PersonId to be deleted, or a record in Employee where EmployeeId matches the PersonId to be deleted. If any of these three records exist, you will not be allowed to delete the person.

Specialization



This design represents 1 to 1 relationships between the general table and the one corresponding to each specialization, that is to say:
a person can only be registered once as a customer, because CustomerId is a valid PersonId, and is also a primary key. Likewise, a person can be registered only once as a passenger and only once as an employee.
A person can have all 3 roles, or be registered only as a person and have no additional data as a client of the agency, as an employee or as a passenger.

In the CUSTOMER, PASSENGER and EMPLOYEE transactions, CustomerName, PassengerName and EmployeeName will be inferred attributes, so they won't be physically stored in the tables CUSTOMER, PASSENGER and EMPLOYEE, respectively.

Remember that the diagrams created by GeneXus don't show 1 to 1 relationships, and the arrows only indicate foreign key relationships. That's why we see the double arrow next to the specialized tables.

There are other cases of use of subtypes that will not be studied in this course. If you are interested, you can look for information related to this topic in the next level course.

GeneXus™

training.genexus.com
wiki.genexus.com