

Nombres de atributos diferentes para el  
mismo concepto

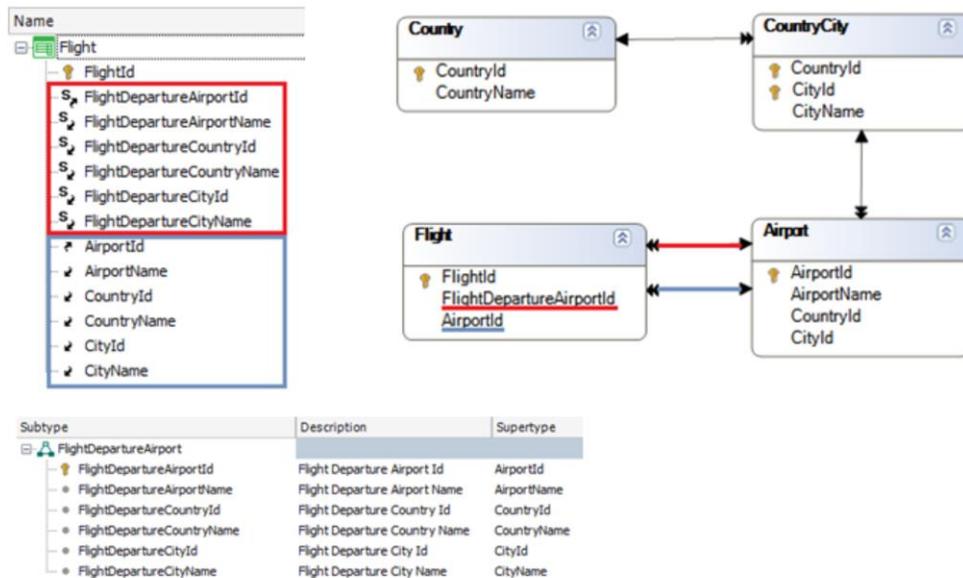
Más casos de uso de subtipos

*GeneXus*™ 15

## Referencias múltiples

## Referencias múltiples

Directa



Habíamos visto un caso donde debimos definir un grupo de subtipos porque teníamos en una transacción una **doble referencia** a un mismo actor de la realidad. Era el caso de la transacción Flight, en la cual teníamos un aeropuerto de partida del vuelo y un aeropuerto de llegada. No podíamos incluir en la estructura de la transacción dos veces el mismo atributo, AirportId. Por esa razón, decidimos dejar ese atributo para el rol de aeropuerto de llegada, y definimos un subtipo de AirportId, al que llamamos FlightDepartureAirportId, para identificar al aeropuerto de partida (lo definimos dentro de un grupo al que llamamos “FlightDepartureAirport”).

Como queríamos inferir el país y ciudad de ese aeropuerto, fue que en el grupo “FlightDepartureAirport” definimos subtipos también de los atributos correspondientes a país y ciudad, así como también un subtipo del nombre del aeropuerto.

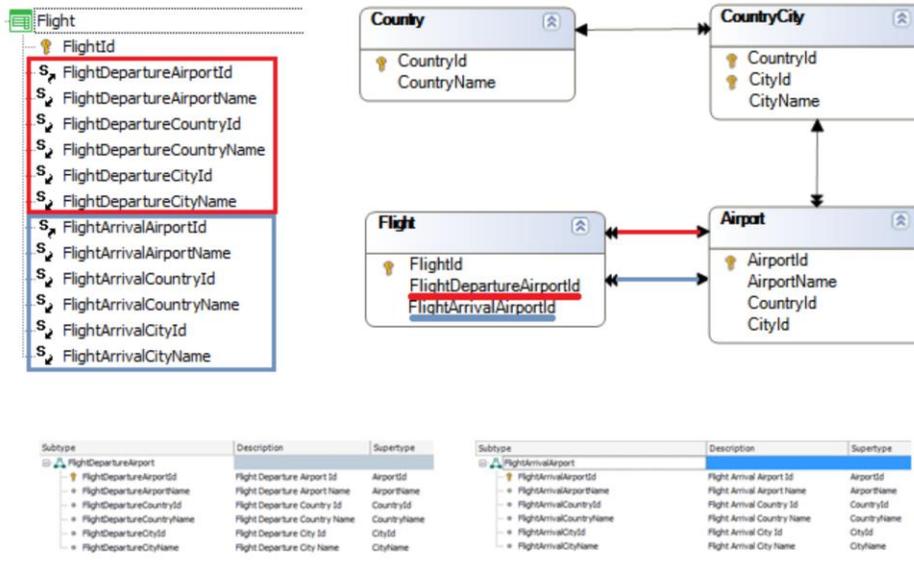
Por lo tanto, cuando en la transacción Flight nombramos a FlightDepartureCountryName, sabemos que será un CountryName inferido a través del aeropuerto de partida: FlightDepartureAirportId. Estos subtipos se han definido dentro del mismo grupo y por lo tanto se ha establecido la asociación y relación entre ellos.

Y cuando en la transacción Flight nombramos a CountryName, sabemos que será inferido a través del atributo AirportId.

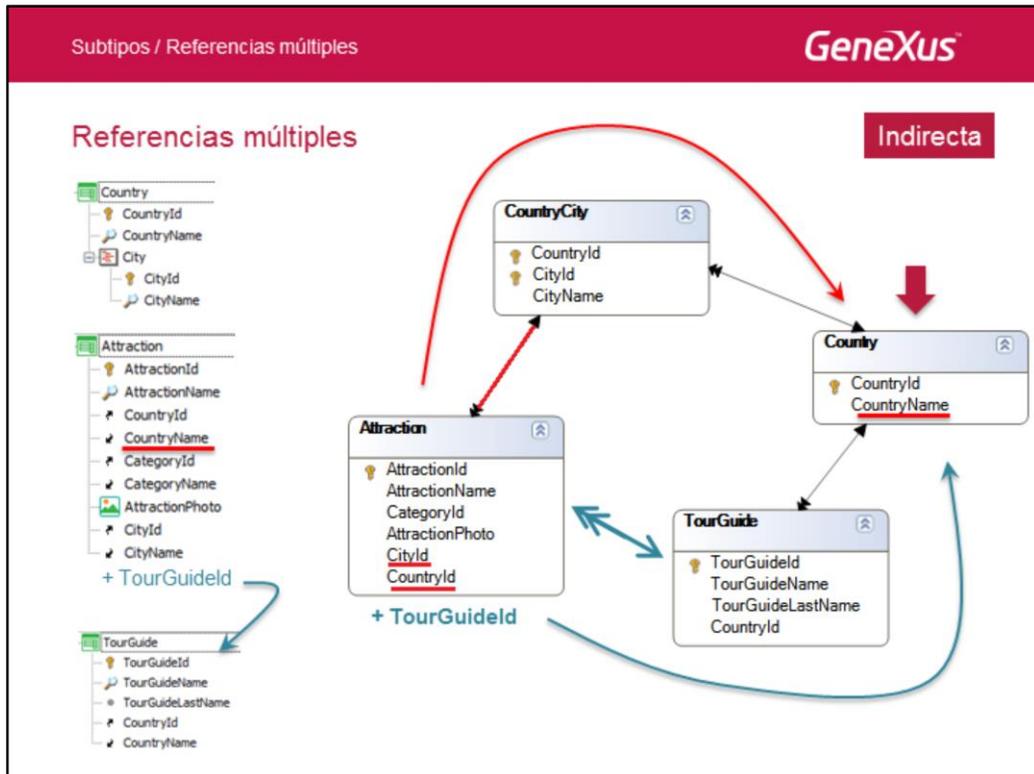
O sea que no hay ninguna ambigüedad. Tenemos dos caminos perfectamente diferenciados para llegar a Country desde Flight.

## Referencias múltiples

Directa



Otra opción era definir también para el rol aeropuerto de llegada un grupo de subtipos. El modelo de datos reflejará las mismas relaciones que en la solución anterior.



En el caso anterior teníamos una doble referencia desde una tabla a otra relacionada directamente con ella. Ahora veamos el caso de una relación indirecta.

Imaginemos que agregamos una transacción para registrar la información de los guías turísticos. Cada guía tiene una nacionalidad determinada, y por ello hemos agregado a la estructura de su transacción el atributo CountryId.

Si observamos el diagrama de tablas, desde la transacción Attraction podemos inferir el CountryName correspondiente a esa atracción, pues se encuentra en su tabla extendida. Análogamente, desde TourGuide también inferimos su propio CountryName, es decir, el país del guía.

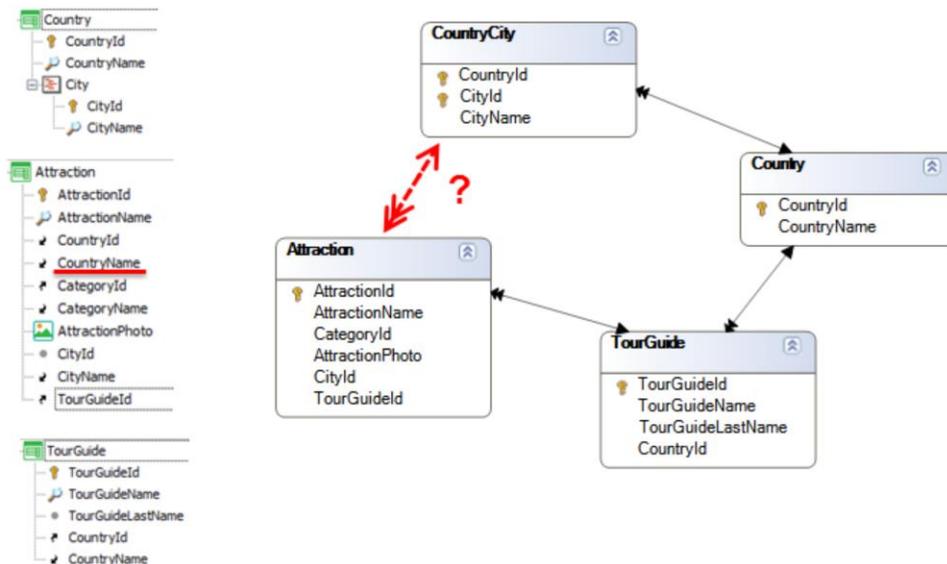
Si ahora vinculamos las entidades Attraction y TourGuide agregando a la primera el atributo TourGuideId, identificador del guía, representando que una atracción turística tiene un guía asignado y sólo uno, ¿cómo se relacionan ahora las tablas?

TourGuideId será una llave foránea en Attraction a la tabla TourGuide, por lo que aparecerá la relación marcada con la flecha celeste. Eso nos puede conducir a pensar que entonces desde Attraction ahora existen dos caminos para inferir CountryName, y eso significa que hay una ambigüedad. O, dicho de otro modo, dado que GeneXus tiene el atributo CountryName en Attraction, ¿de dónde lo infiere?: ¿de la ciudad de la atracción o del guía turístico de la atracción? Si ambos valores coincidieran, no importaría, pero en este caso no tienen por qué coincidir. El país donde se encuentra la atracción no tiene por qué coincidir con el país natal del guía turístico.

Necesitaremos subtipos para poder diferenciar ambos roles de CountryName.

## Referencias múltiples

Indirecta



Pero en este caso no sólo los necesitaremos para diferenciar los roles. Si observamos el diagrama de tablas después de agregar el atributo TourGuideId a Attraction, vemos que desaparece la relación entre Attraction y CountryCity. Dicho de otro modo, CountryId ya no es una llave foránea en Attraction. Observar que ya no está en la tabla, es decir, será un atributo inferido. ¿Inferido a partir de quién? ¡De TourGuideId!

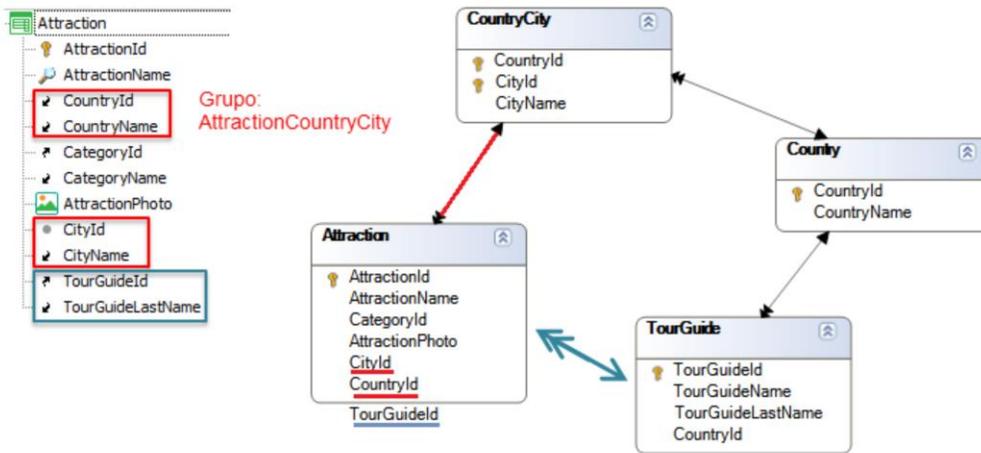
No olvidemos que antes que nada GeneXus normaliza las tablas. Es decir, en base a los nombres de los atributos, en conjunto con los identificadores, determina qué atributo se coloca en cada tabla y las relaciones entre las tablas. Como en Attraction aparece TourGuideId, que es identificador de TourGuide, y a la vez en TourGuide aparece CountryId, que es identificador de Country, está entendiendo que dado un TourGuideId en Attraction el CountryName lo infiere a partir de él, pasando por la tabla intermedia TourGuide.

Por tanto con este diseño de transacciones **hemos perdido** la posibilidad de indicar cuál es **el país de la atracción**. El CountryName será el del guía turístico.

No tenemos otra alternativa que utilizar subtipos para poder lograr que Attraction tenga un país propio, independiente del país del guía turístico.

## Referencias múltiples

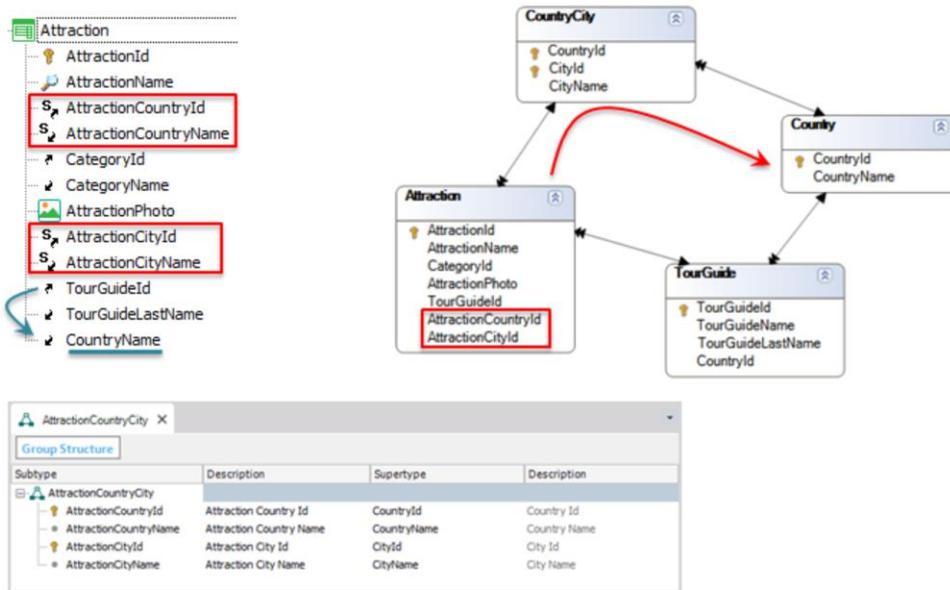
Indirecta



Otra vez, tenemos varias alternativas. La más evidente: armar un grupo de subtipos para país/ciudad de la atracción.

## Referencias múltiples

Indirecta



Aquí definimos un grupo de subtipos para representar el país y ciudad de la atracción. Observemos que ahora sí GeneXus representa correctamente las relaciones, y además el atributo CountryName ahora se infiere a partir de TourGuideId sin ambigüedad. El atributo que representa y en el que se infiere el país de la atracción será el de nombre AttractionCountryName, subtipo de CountryName, perteneciente al grupo AttractionCountryCity.

También obsérvese que este grupo tiene dos atributos primarios: AttractionCountryId y AttractionCityId, que corresponden a la llave primaria de la tabla CountryCity, por los supertipos que indicamos: {CountryId, CityId}.

## Subtipos recursivos

## Subtipos recursivos

- Cuando desde una entidad debe auto-referenciarse:

Employee	Employee	Employee	
EmployeeId	Id	Employee Id	No
EmployeeName	Name	Employee Name	No
EmployeeLastName	Name	Employee Last Name	No
EmployeeIsManager	Boolean	Employee Is Manager	No
EmployeeManagerId	Id	Employee Manager Id	Yes
EmployeeManagerName	Name	Employee Manager Name	
EmployeeManagerLastName	Name	Employee Manager Last Name	

Subtype	Description	Supertype
EmployeeManager		
EmployeeManagerId	Employee Manager Id	EmployeeId
EmployeeManagerName	Employee Manager Name	EmployeeName
EmployeeManagerLastName	Employee Manager Last Name	EmployeeLastName

En el ejemplo estamos representando la información de los empleados de la agencia de viajes. Cada empleado puede ser, a su vez, gerente de otro u otros empleados. De todos los empleados que tienen un jefe, se necesita indicar quién es ese jefe.

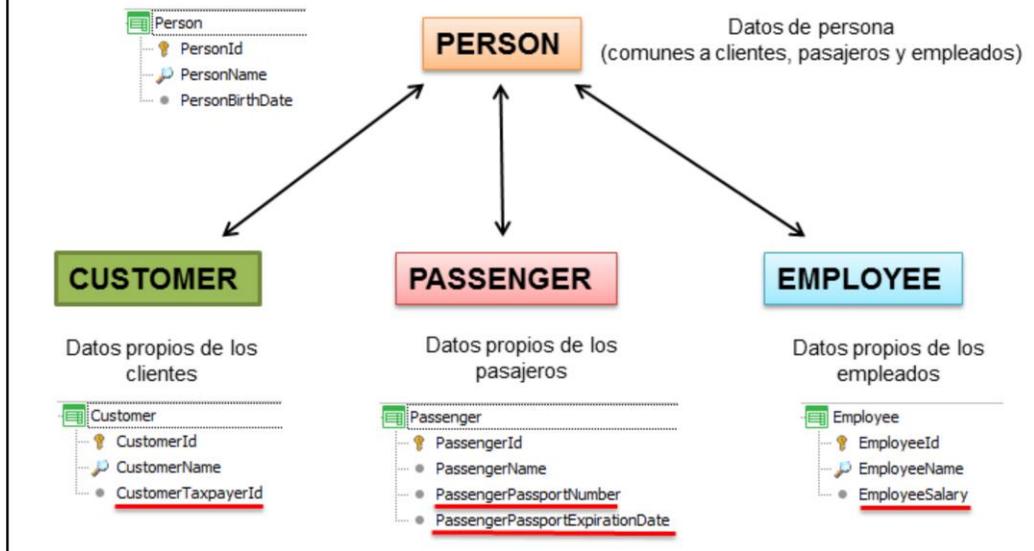
El jefe es, en particular, un empleado. Por tanto se establece una relación de la tabla de empleados con ella misma.

Para ello, debemos crear un grupo de subtipos para representar la información del jefe del empleado.

El atributo EmployeeManagerId será, a todos los efectos, tomado como un EmployeeId. Por tanto, conformará una llave foránea a la propia tabla Employee. Por lo que, cuando se ingrese la información de un empleado a través de la transacción, cuando el usuario elija un valor para el campo EmployeeManagerId, GeneXus controlará la integridad referencial. Esto es: controlará que exista un registro en la tabla de empleados con ese valor para el atributo EmployeeId.

# Especialización

## Especialización



¿A qué nos referimos con **especialización**?

Supongamos que la agencia de viajes necesita manejar información específica de los clientes a los que les vende pasajes y paquetes turísticos (por ejemplo su número de contribuyente en la oficina estatal de impuestos, si lo tiene), información específica de los pasajeros (por ejemplo, necesitará registrar su número de pasaporte) y también información específica de los empleados de la agencia, para los que deberá registrar, por ejemplo, su salario.

Es decir, la agencia de viajes hará facturas a los clientes, registrará en los asientos de los vuelos a pasajeros y realizará recibos de sueldo a empleados.

Podríamos definir entonces, en vez de lo que antes era solamente una transacción, Customer, la definición propuesta **arriba**. Tanto los customers, como los passengers, como los employees, son personas, por lo que tienen cierta información en común (por ejemplo, un nombre, una fecha de nacimiento, etc.).

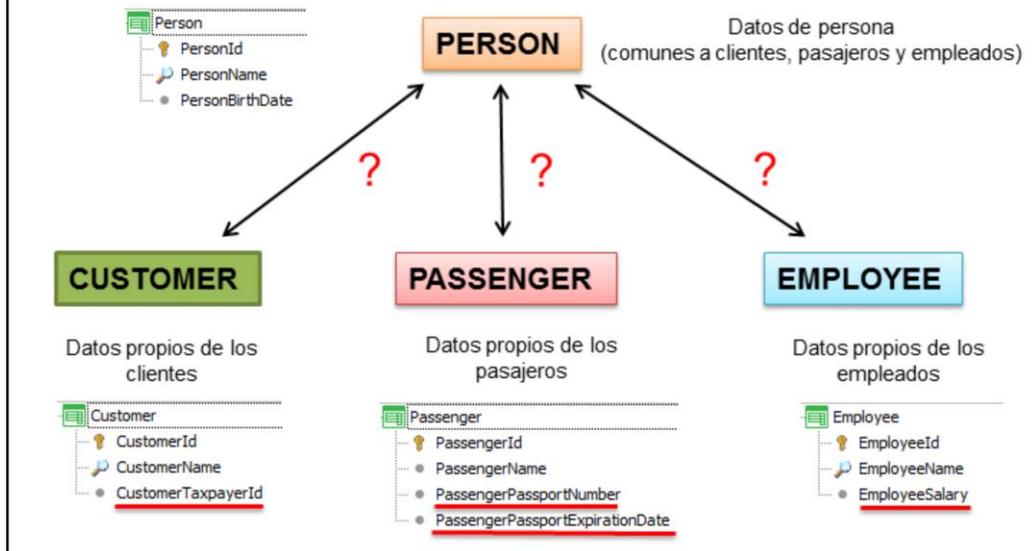
Ahora Customer es una especialización de Person, así como Passenger y Employee.

Podemos leer la relación:

- Cada Customer **es** Person
- Cada Passenger **es** Person
- Cada Employee **es** Person

Observemos que la propuesta tiene 4 transacciones. Person, con los datos comunes a todas las personas. Y luego las especializaciones: Customer, Passenger y Employee, cada una con sus datos específicos (Customer tendrá el nro. de contribuyente, Passenger el número de pasaporte y Employee el salario).

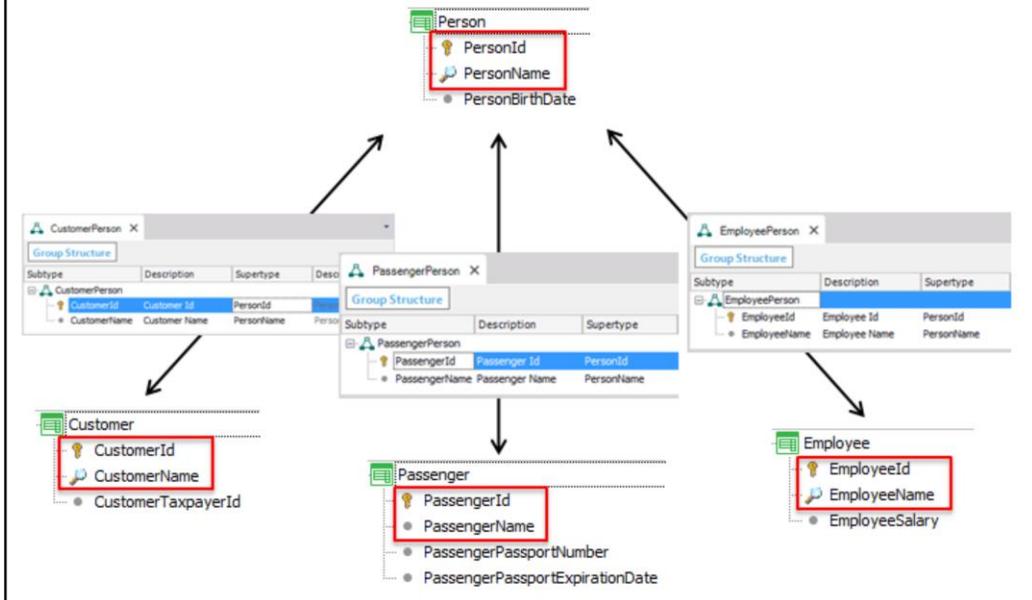
## Especialización



Queremos que el identificador de cliente **coincida exactamente** con el de una persona, para reflejar que el cliente es una persona. Es decir, si la persona de id 8 se llama Ann Roberts, y nació el 05/05/1970, cuando vamos a ingresar su información como cliente, necesitamos que el usuario pueda digitar en la transacción Customer el id 8, y que al salir del campo se le muestre el nombre Ann Roberts y pueda ingresar el número de contribuyente (CustomerTaxpayerId). De la misma manera, si se ejecuta la transacción Passenger, deseamos que cuando el usuario digite en PassengerId el valor 8, en PassengerName se infiera Ann Roberts, y el usuario pueda asignar el número de pasaporte y la fecha de expiración del mismo en los atributos propios (PassengerPassportNumber y PassengerPassportExpirationDate). Análogamente con los empleados.

Si simplemente definimos como claves primarias de Customer, Passenger y Employee, los atributos CustomerId, PassengerId y EmployeeId respectivamente, sin relacionarlos de ningún modo con PersonId (lo mismo que CustomerName, PassengerName y EmployeeName sin relacionarlos con PersonName), no conseguiremos lo que buscamos. Para GeneXus serán transacciones por completo independientes.

## Especialización



Pero vamos a definir grupos de subtipos para representar que un cliente debe ser una persona válida (o sea previamente registrada), que un pasajero debe ser una persona y que un empleado también.

Creamos un grupo CustomerPerson, donde definimos a CustomerId y a CustomerName como subtipos de PersonId y PersonName, respectivamente.

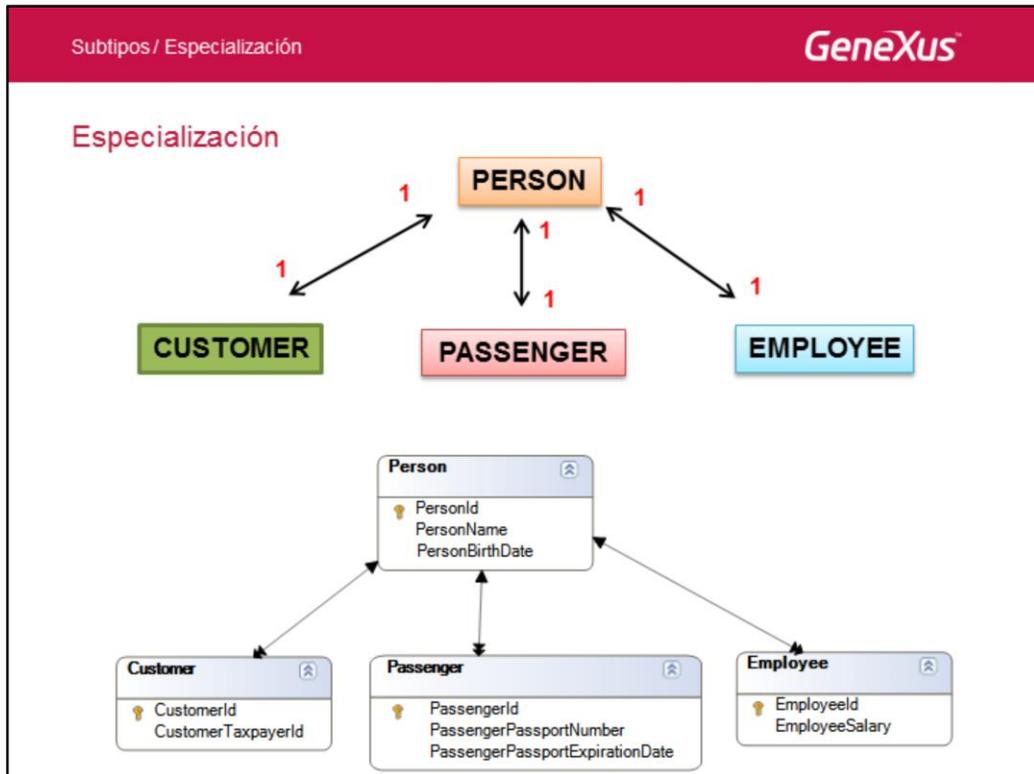
Otro de nombre PassengerPerson, donde definimos a PassengerId y a PassengerName como subtipos de PersonId y PersonName, respectivamente.

Y por último un grupo EmployeePerson, donde definimos a EmployeeId y EmployeeName como subtipos de PersonId y PersonName, respectivamente.

Al hacer esto, los atributos CustomerId, PassengerId y EmployeeId, además de ser los identificadores de las tablas Customer, Passenger y Employee respectivamente, y por tanto, sus claves primarias, serán, a la vez, claves foráneas a la tabla Person.

Por tanto, cuando el usuario ingrese un valor en el id de cualquiera de las tres transacciones (Customer, Passenger o Employee), se irá a buscar a la tabla Person, un registro que tenga como id ese mismo valor.

Del mismo modo, si se quiere eliminar una persona, a través de la transacción Person, se controlará que no exista registro en Customer donde CustomerId = PersonId que se está queriendo eliminar, ni registro en Passenger donde PassengerId = PersonId, ni registro en Employee donde EmployeeId = PersonId. Si existiera alguno de esos tres registros, no se permitirá la eliminación de la persona.



Observemos que este diseño representa relaciones 1 a 1 entre la tabla general y la correspondiente a cada especialización.

Es decir: una persona puede estar registrada una sola vez como cliente, porque CustomerId es un PersonId válido, y además es llave primaria. De igual manera una persona puede estar registrada una sola vez como pasajero y una sola vez como empleado.

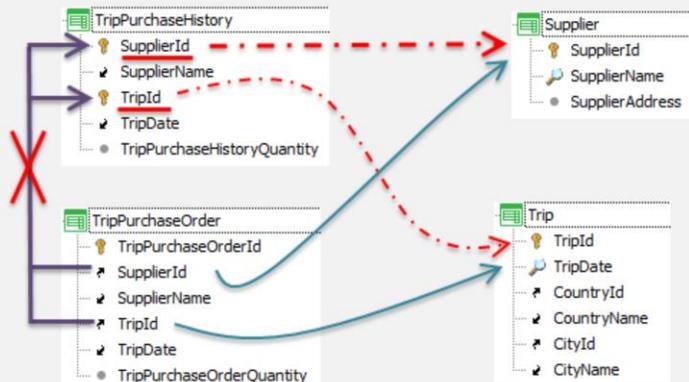
Una persona puede tener los 3 roles, o estar registrado solamente como persona y no tener datos extra como cliente de la agencia, ni como empleado ni como pasajero.

Piense cómo será la estructura de las tablas CUSTOMER, PASSENGER y EMPLOYEE. Evidentemente, CustomerName, PassengerName y EmployeeName serán atributos inferidos (que no estarán físicamente en las tablas físicas CUSTOMER, PASSENGER y EMPLOYEE respectivamente).

Nota: Los diagramas que realiza GeneXus no muestra las relaciones 1 a 1. Es por ello que vemos la flecha doble del lado de las tablas especializadas. Con las flechas GeneXus sólo indica las relaciones de claves foráneas.

## Más casos de uso

- Eliminación de relaciones no deseadas por claves foráneas compuestas.



Hay más casos de uso. Por ejemplo, si tenemos una transacción para registrar las órdenes de compra de excursiones a proveedores (suppliers) de la agencia de viajes, y luego una para registrar el histórico de compras por proveedor/excursión, vemos que entre las claves foráneas, se determinará una que relaciona a la orden de compras con el histórico. Pero no queremos que se genere, puesto que cuando se crea la primera orden de compra de un supplier/trip no existirá registro en el histórico.

Una forma de lograr que esa relación no se controle, será modificándole el nombre a alguno de los atributos de la clave primaria de TripPurchaseHistory. Es decir, definiendo subtipos.

No estudiaremos en detalle este caso en este curso.

# GeneXus™

Videos

[training.genexus.com](http://training.genexus.com)

Documentación

[wiki.genexus.com](http://wiki.genexus.com)

Certificaciones

[training.genexus.com/certificaciones](http://training.genexus.com/certificaciones)