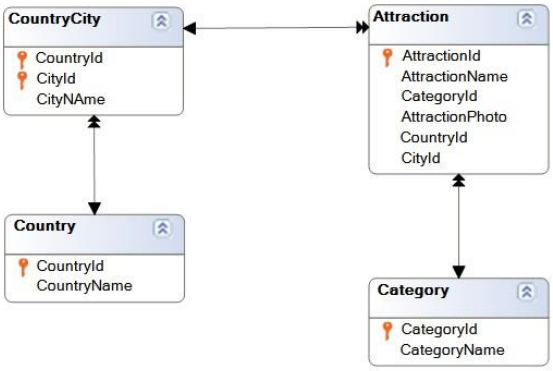
Más sobre el comando For each

GeneXus™

Repaso: Transacción base



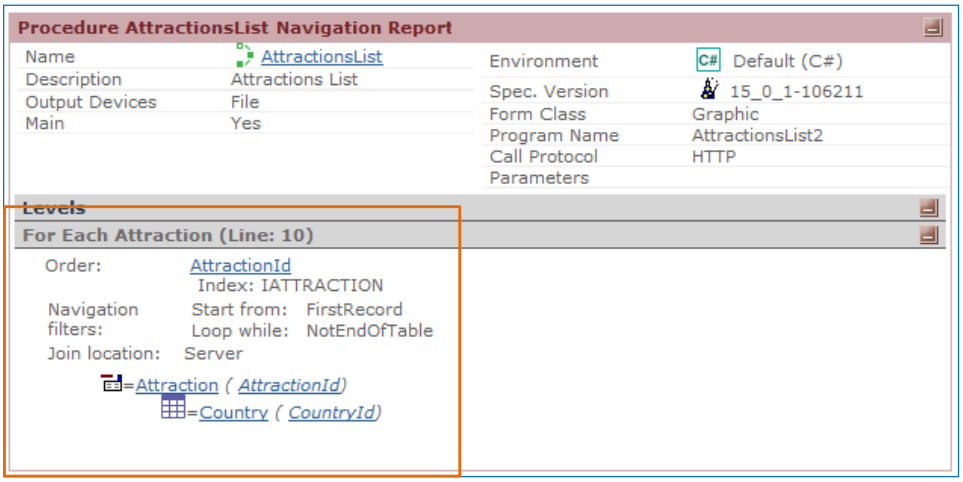

```
print Title

for each Attraction
     print Attractions
endfor
```

Recordemos que GeneXus determina la tabla base del for each teniendo en cuenta el nombre de la transacción que declaramos al lado del for each, que corresponde al nombre de la transacción base, o sea aquella transacción cuya tabla física asociada queremos recorrer.

Pero además, los atributos declarados dentro del for each, ya sea en **printblocks**, cláusulas **where**, **order**, etc, deben pertenecer a la tabla extendida de la tabla base del for each.

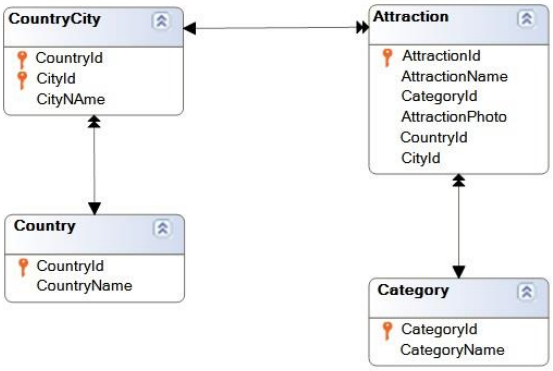
En este ejemplo que estamos viendo, la tabla base del for each será ATTRACTION, o sea la tabla que se va a recorrer y se accederá a su tabla extendida para poder recuperar los datos requeridos.

Repaso: Transacción base



Si observamos el listado de navegación, vemos que nos informa claramente que la tabla base es ATTRACTION, que la recorrida será ordenada por la clave primaria de dicha tabla, o sea por AttractionId, y que se recorrerá toda la tabla, accediendo también a la tabla COUNTRY para recuperar el valor de CountryName, que corresponde al país de la atracción.

Repaso: Transacción base



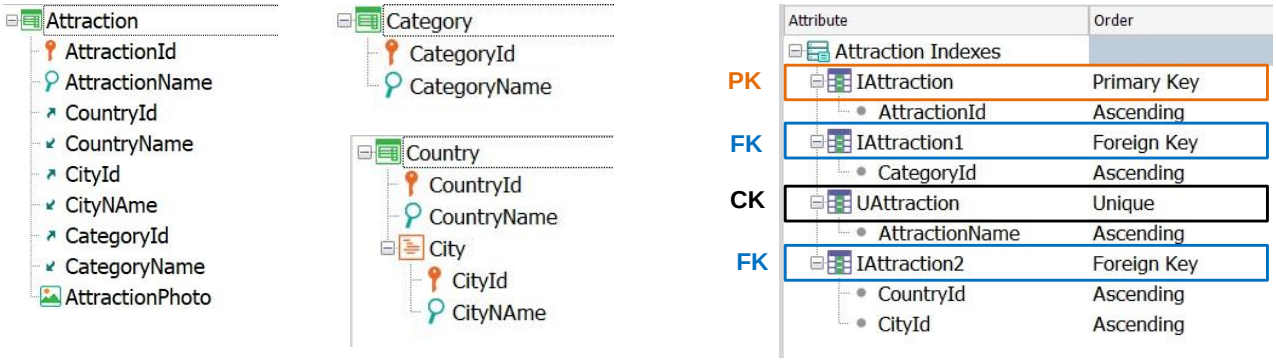

```
print Title

for each  ?
      print Attractions
endfor
```

No es obligatorio especificar la transacción base de un For each.

Ahora bien, ¿Es obligatorio especificar transacción base para un for each?

La respuesta es No. GeneXus podrá calcular la tabla base del for each a partir de los atributos que participen del comando. La forma en que encuentra la tabla base no será vista en este curso.

Indices y su relación con las consultas a la base de datos



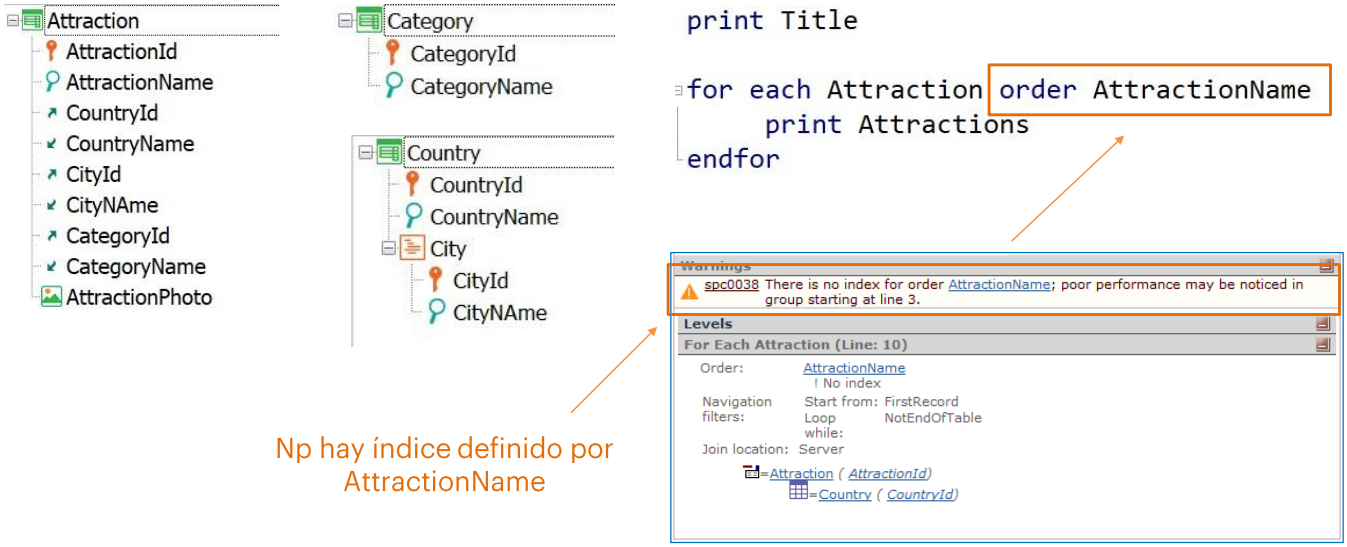
Pasemos ahora a los índices y su relación con las consultas a la base de datos.

Ya sabemos que los **índices** son vías de acceso eficiente a los datos.

Si recordamos lo ya visto, en cada tabla, GeneXus crea un índice por el atributo primario (ya sea una clave simple o compuesta) y un índice por cada clave foránea. Esto lo hace para que sean más eficientes los controles de consistencia de los datos entre las tablas.

Hemos visto también que es posible definir índices, indicando si aceptan valores duplicados o no. Si definimos un índice que no acepta valores duplicados, o sea, un índice Unique, le estamos diciendo a GeneXus que debe controlar automáticamente la unicidad de su valor, y ese atributo, o conjunto de atributos sobre los cuales se define el índice, pasa a ser una clave candidata.

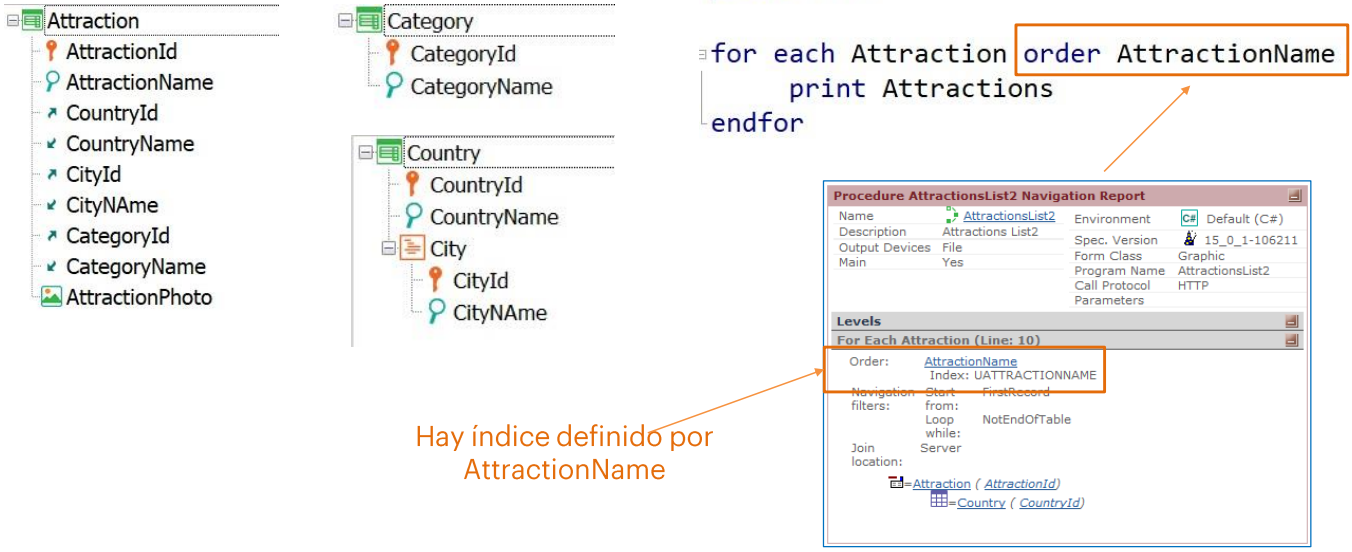
Indices y su relación con las consultas a la base de datos



También hemos visto que si agregamos una cláusula order, por ejemplo para ordenar por el nombre de la atracción, el listado de navegación nos da un aviso, informándonos de que en la base de datos no existe un índice por el atributo por el que necesitamos ordenar la información, por lo que podríamos tener una baja performance para esta consulta.

Es que al indicarle un atributo por el que ordenar, GeneXus intenta que la ordenación sea eficiente y por lo tanto busca si existe un índice por ese atributo. Pero como no lo encuentra, nos lo hace saber.

Indices y su relación con las consultas a la base de datos



Si necesitamos entonces obtener los registros de ATTRACTION ordenados por el atributo AttractionName, entonces tendrán que reordenarse estos registros ya que por defecto están ordenados por el valor del atributo que es clave primaria.

Cuando se define una consulta, si hay un índice físico creado en la tabla por el atributo a ordenar, GeneXus lo usará. Pero en este caso la consulta se necesita ordenar por un atributo secundario: AttractionName. Y GeneXus nos advierte en el listado de navegación como hemos visto, que no hay un índice definido.

La existencia del índice optimizaría la consulta. Pero la desventaja de crear un índice es que, a partir de allí, debe ser mantenido. Es decir, si los usuarios van agregando, modificando y eliminando atracciones en la tabla ATTRACTION, debe reacomodarse este índice.

Una vez hecho esto, al presionar F5, deberá reorganizarse la base de datos, para crear este nuevo índice. Entonces, en el listado de navegación veremos que GeneXus utilizará ese índice que se acaba de crear.

Vale mencionar que así como lo creamos, en cualquier momento podemos eliminar un índice, y al hacer F5 y reorganizar, volveremos a la situación de la que habíamos partido antes de crearlo.

Indices y su relación con las consultas a la base de datos: Ejemplo


```
Parm (in:&NameFrom, in:&NameTo);

print Title

for each Attraction order AttractionName
    Where AttractionName >= &NameFrom
    Where AttractionName <= &NameTo

    print Attractions
endfor
```

Where AttractionName >= &NameFrom and AttractionName <= &NameTo

Veamos este ejemplo:

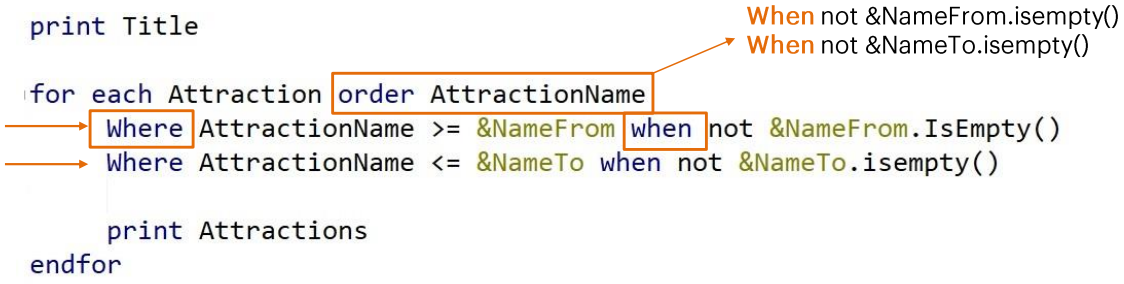
Supongamos que nos interesa obtener un listado de las atracciones cuyos nombres estén alfabéticamente entre un par de valores recibidos por parámetro. Por ejemplo, entre las letras "B" y "N".
Para eso especificamos las cláusulas where que estamos viendo.

Tener varias cláusulas where es equivalente a tener una sola, donde las condiciones se conjugan con el operador lógico "and". Es decir, que se van a considerar solamente los registros que cumplan con todas las condiciones a la vez.

Si vamos a filtrar por AttractionName, y tenemos un índice creado por ese atributo, nos convendrá siempre ordenar por AttractionName para optimizar la consulta.

Observemos que si no especificamos cláusula order, GeneXus ordenará por clave primaria, y deberá recorrerse toda la tabla para saber si una atracción está dentro del rango especificado o no.

Cláusula When



Veamos ahora la cláusula When que nos permitirá condicionar la aplicación de los órdenes y los filtros.

¿Qué resultado se obtendrá para el for each que estamos viendo, si las variables &NameFrom y &NameTo están vacías?

Si existiera una atracción con nombre vacío, sería entonces la única devuelta, pues sería la única que va a cumplir con las condiciones. En caso contrario, ninguna atracción será listada.

¿Es posible entonces considerar los ordenamientos y los filtros, para que sólo se apliquen ante determinadas circunstancias? Por ejemplo, para que sólo se aplique el primer where **cuando** la variable &NameFrom no está vacía?. Y que sólo se aplique el segundo where **cuando** la variable &NameTo no está vacía?.

La respuesta es sí. Lo conseguimos condicionando las cláusulas where con **when**. Sólo se aplicará cada where cuando la condición del when se satisfaga.
De esta forma, en ejecución, cuando dejemos ambas variables vacías, no se aplicará ninguno de los where, por lo que saldrán listadas todas las atracciones de la tabla. Si la variable &NameFrom está vacía pero &NameTo no lo está, no se aplicará el primer where pero sí el segundo, por lo que se listarán todas las atracciones cuyo nombre sea menor o igual a &NameTo.

De la misma manera puede condicionarse la aplicación o no de un order. De hecho puede especificarse una sucesión de órdenes condicionados, de manera que el primero cuya condición se satisfaga sea el elegido.

Cláusula When none

```
print Title

for each Attraction order AttractionName
     Where AttractionName >= &NameFrom when not &NameFrom.IsEmpty()
     Where AttractionName <= &NameTo when not &NameTo.isempty()

     print Attractions
When none
     Print NoAttractions
endfor
```

Pasemos ahora a la cláusula **When none**.

¿Qué sucede cuando ninguno de los registros de la tabla base cumple con las condiciones indicadas?
Supongamos que en ese caso queremos imprimir en la salida un mensaje que lo advierta... y que diga que no hay registros asociados.

Para eso varmos a programar la cláusula **when none**.

Todos los comandos que se escriban entre el when none y el endfor se ejecutarán secuencialmente y **en el único caso en que no se hayan encontrado registros de la tabla base del for each que cumplan con las condiciones indicadas.**

En este ejemplo que estamos viendo hemos decidido imprimir un mensaje, pero se podrían escribir una serie de comandos, como por ejemplo otro for each.

Como la ejecución de lo que sigue al **when none** implicará que no se encontró lo que se buscaba, si allí se escribe un for each, entonces no quedará anidado al del when none. Será como un for each independiente.

Resumen...

For each *BaseTransaction*

order att_1, att_2, … , att_n [when *condition*]

order att_1, att_2, … , att_n [when *condition*]

where *condition* [when *condition*]

where *condition* [when *condition*]

main code

When none

.....

endfor

Resumiendo esto que hemos visto...

Como ya hemos visto, la tabla base de un For each se determina a partir de la transacción base especificada; el resto de los atributos mencionados, tanto en el cuerpo del For each (main code) como en las cláusulas Order y Where, deberán pertenecer a la tabla extendida de esa tabla base.

Los atributos mencionados en el bloque When none no serán considerados.

Si observamos, dejamos en gris todo lo que ya habíamos visto antes. Aquí hemos agregado las cláusulas when y when none.

Más adelante veremos que es posible agregar más cláusulas a este fundamental comando de acceso a la base de datos.

Caso de estudio

```
Customer
{
   CustomerId*
   CustomerName
}

Trip
{
   TripId*
   TripDescription
   TouristGuideId
   TouristGuideName
}
```

```
TouristGuide
{
   TouristGuideId*
   TouristGuideName
   Phone
   {
      TouristGuidePhoneId*
      TouristGuidePhoneNumber
   }
}
```

```
Reservation
{
   ReservationId*
   ReservationDate
   CustomerId
   CustomerName
   Trip
   {
      TripId*
      TripDescription
   }
}
```

Para finalizar, analicemos un caso de estudio:

Consideremos el siguiente diseño de transacciones:

La transacción Customer, la transacción Trip que corresponde a las excursiones con un guía de turismo a cargo, la transacción Tourist Guide con su conjunto de teléfonos, y la transacción Reservation para registrar, para cada cliente, el conjunto de excursiones que tiene reservada.

Necesitamos obtener un listado que muestre, para un cliente dado, y a partir de una fecha dada, todas las excursiones que tiene reservadas, y para cada una de ellas los teléfonos de contacto del guía de turismo a cargo.

Caso de estudio

```
Trip
{
   TripId*
   TripDescription
   TouristGuideI
   TouristGuideName
}
```

```
TouristGuide
{
   TouristGuideId*
   TouristGuideName
   Phone
   {
      TouristGuidePhoneId*
      TouristGuidePhoneNumber
   }
}
```

```
Reservation
{
   ReservationId*
   ReservationDate
   CustomerId
   CustomerName
   Trip
   {
      TripId*
      TripDescription
   }
}
```

/out | **Rules *** | Conditions | Variables

```
Parm(in:&ReservationDate, in: CustomerId);


For each Reservation.Trip
     Where ReservationDate >= &ReservationDate
     Print Trips
     For each TouristGuide.Phone
          Print TouristGuidesPhones
     Endfor
Endfor
```

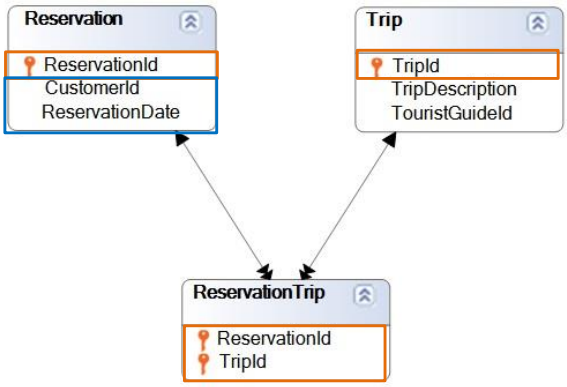
Para resolverlo, proponemos el siguiente source...

Analicemos si la información listada es la que se nos pide. Tenemos un par de for each anidados. En el primero decimos explícitamente que la tabla base será la correspondiente al nivel Trip de la transacción Reservation, es decir, la de nombre ReservationTrip.

Verificamos que dentro de este for each externo no hay ningún atributo que no pertenezca a la tabla extendida de RESERVATIONTRIP, puesto que de ser así, el listado de navegación arrojará una advertencia indicando que ese atributo no es accesible.

Los atributos entonces que debemos verificar son los que se encuentran en la cláusula Where y dentro del printblock de nombre Trips, que en este caso son los atributos TripDescription, presente en TRIP, y ReservationDate presente en RESERVATION.

Caso de estudio




```
/out | Rules * | Conditions | Variables

Parm(in:&ReservationDate, in: CustomerId);


For each Reservation.Trip
     Where ReservationDate >= &ReservationDate
     Print Trips
     For each TouristGuide.Phone
          Print TouristGuidesPhones
     Endfor
Endfor
```

Si observamos el diagrama de tablas vemos claramente que desde RESERVATIONTRIP accedemos a un único registro deTRIP y a un único registro de RESERVATION. GeneXus deberá, entonces, acceder a esas dos tablas cada vez que itere en el for each.

Entonces ¿Con qué registros de la tabla base va a trabajar el for each? Con aquellos que cumplan que, al ir a la tabla RESERVATION para evaluar el valor de ReservationDate, éste sea mayor o igual al valor de la variable &ReservationDate recibida por parámetro.

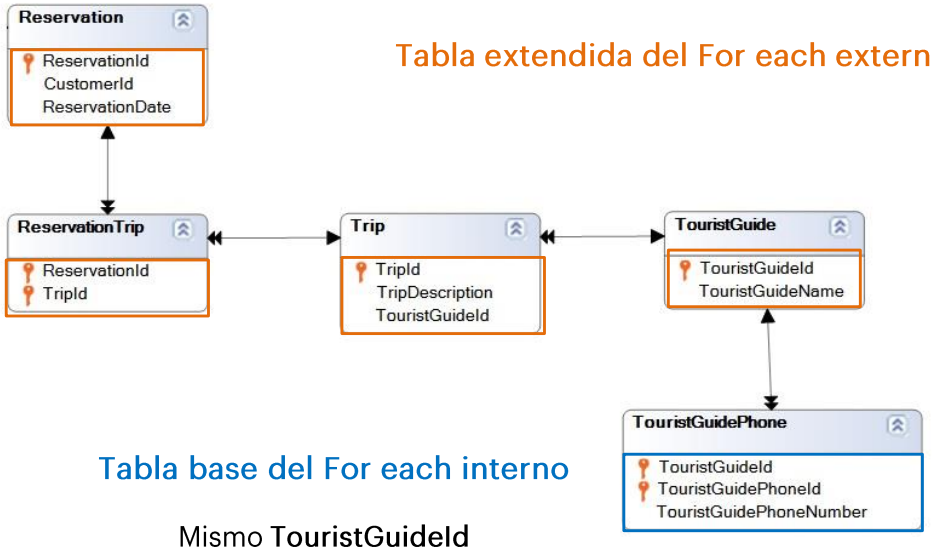
Pero también deberá cumplir que el valor de CustomerId coincida con el valor recibido por parámetro directamente en ese atributo. Recordemos que "recibir en atributo" es establecer que ese atributo estará instanciado. O sea, que toda vez que ese atributo participe en algún lado, lo hará con el valor recibido cuando se invocó al objeto.

Como en el for each que estamos analizando ya se accede a la tabla RESERVATION que lo contiene, entonces se aplicará un filtro automático por ese valor de CustomerId.

Es importante hacer la siguiente aclaración: El hecho de que el atributo recibido en la regla parm pertenezca a la tabla extendida del for each NO HACE que éste aplique automáticamente como filtro. Para que eso suceda, el for each tiene que estar accediendo particularmente a esa tabla de la extendida para realizar alguna acción.

Caso de estudio

Relación 1-N indirecta



Ahora pensemos qué sucede con el for each anidado. Su tabla base será claramente la asociada al nivel Phone de la transacción TouristGuide. La siguiente pregunta es: ¿establece filtros implícitos para la información que utilizará? Sabemos que sí, que mostrará los teléfonos del guía de cada excursión

.¿Por qué? GeneXus busca si hay relación entre la tabla extendida del for each externo, y la tabla base del for each anidado:

Es una manera de buscar una relación 1 a N, aunque en este caso indirecta. Si cada RESERVATIONTRIP tiene un TouristGuideId, y en la tabla a ser navegada también hay un TouristGuideId, entonces GeneXus entiende que por la relación entre la información, se tratará del mismo. Y por eso realiza el join.

En los próximos videos seguiremos profundizando sobre el comando For each.

GeneXus™

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