Para modelar la información correspondiente al recorrido de cada una de las rutas, se puede utilizar un tipo de entidad que representa las paradas que se efectúan en cada una de las rutas, débil con respecto a ruta. Para identificar cada una de estas paradas y además representar la relación de orden de paradas de la ruta, se puede utilizar como discriminador un número natural. El lugar asociado a la parada (ciudad, punto turístico, restaurante, etc.) se puede representar con un simple atributo de parada, puesto que no se dispone de información adicional del lugar (si fuese así, se tendría que modelar lugar como un tipo de entidad y unirlo a través de una interrelación con parada). Esta solución permite representar que haya varias paradas en un mismo lugar de la ruta (p.e. una ciudad, paradas de descanso en una autopista, etc.)

Si se quieren representar rutas para las que todavía no se ha definido el recorrido concreto, la cardinalidad mínima de parada con respecto a ruta en el tipo de interrelación constar será 0, tal y como se muestra en la fig.1. En caso contrario, la cardinalidad de parada con respecto a ruta en el tipo de interrelación constar sería (1,N).

El lugar de salida de una ruta se corresponde con la parada de la ruta cuyo valor de *num_parada* es menor; también se podría imponer que su número de parada fuese siempre 0.

El tiempo empleado en llegar a una parada desde la anterior, así como la distancia, se pueden representar como atributos de la parada. Evidentemente, para la primera parada de una ruta cualquiera, estos valores son cero (véase fig.1)

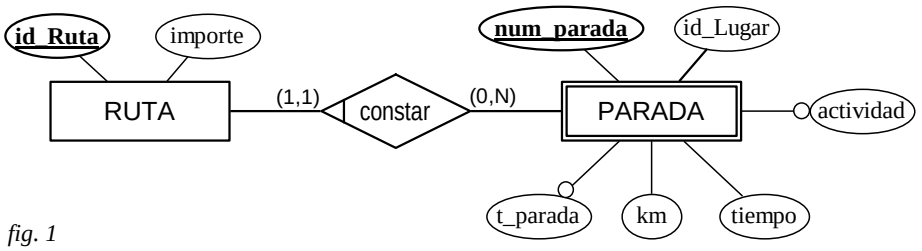

fig. 1

RESTRICCIONES:

- *Verificar que para toda ruta, la parada con mínimo valor de* num_parada *tiene km=0 y tiempo=0*

Si se desea representar explícitamente el lugar de salida de una ruta, se puede poner un atributo derivado (véase fig.2). Obsérvese que se ha modelado como opcional, dado que se ha considerado que una ruta puede no tener paradas.

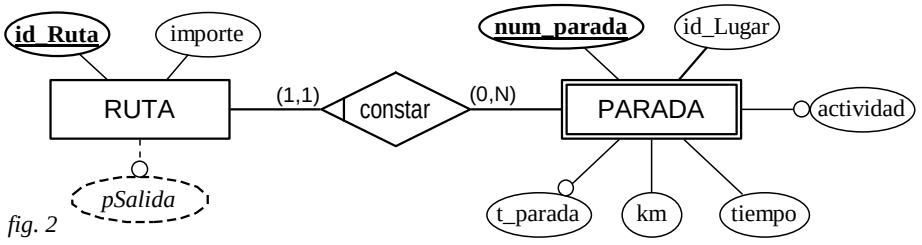

fig. 2

- *Donde* lug_salida *es el menor valor de num_parada correspondiente a la ruta*

RESTRICCIONES:

- *Verificar que para toda ruta, la parada con mínimo valor de* num_parada *tiene km=0 y tiempo=0*

También se podría haber explicitado el lugar de salida mediante una interrelación que, al ser redundante, necesitaría la correspondiente restricción (véase fig.3)

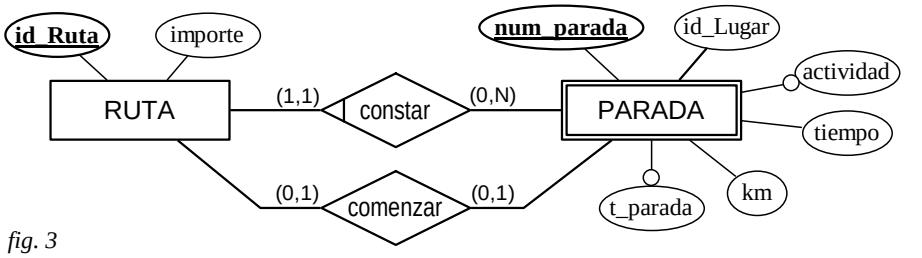

fig. 3

RESTRICCIONES:

- *Verificar que toda ruta que tiene al menos una parada, comienza en la parada de dicha ruta con menor valor de* num_parada.
- *Verificar que para toda parada correspondiente al lugar de salida de una ruta, km=0 y tiempo=0*

Otro modo de modelar la lista de paradas de una ruta consiste en representar de forma explícita el trayecto entre dos paradas consecutivas, mediante una interrelación. Esto significa que se puede utilizar como discriminador el *id_Lugar* (no es necesario *num_parada*), aunque con esta solución, a lo sumo puede haber una parada por lugar en cada una de las rutas; es decir, no se pueden representar ciclos. Para resolver esta limitación basta con utilizar un *id_Parada* como discriminador, en vez de *id_lugar* (véanse las soluciones anteriores).

La información de la distancia y el tiempo de cada trayecto serán atributos de esta interrelación. Cada parada se puede modelar con un tipo de entidad débil como en el caso anterior (véase fig.4)

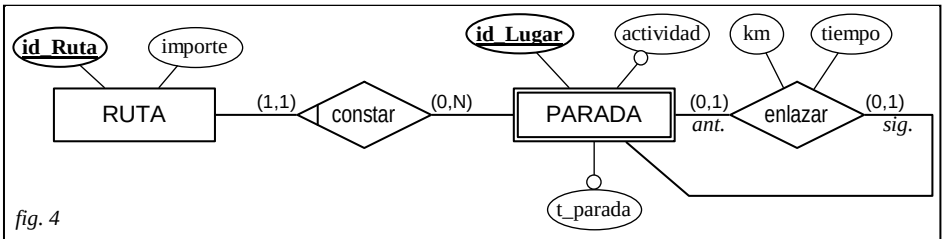

fig. 4

RESTRICCIONES:

- *Verificar que la siguiente parada de una ruta (si hay) pertenece a la misma ruta.*
- *Verificar que todas las paradas de una ruta excepto una (la primera) tienen una parada anterior, y todas excepto una (la última) tienen una parada siguiente; es decir, que no hay ciclos y que los trayectos forman un conjunto conexo.*

Para representar explícitamente el lugar de salida se pueden emplear las mismas soluciones que en el caso anterior (véase por ejemplo las fig.5 y fig.6)

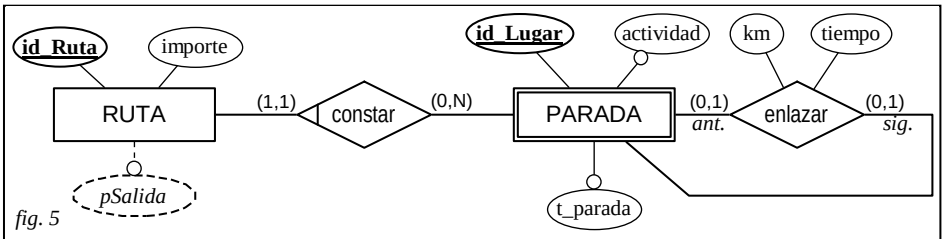

fig. 5

- *donde* pSalida *representa la parada de la ruta que no tiene una parada anterior*

RESTRICCIONES:

- *Verificar que la siguiente parada de una ruta (si hay) pertenece a la misma ruta.*
- *Verificar que todas las paradas de una ruta excepto una (la primera) tienen una parada anterior, y todas excepto una (la última) tienen una parada siguiente; es decir, que no hay ciclos y que los trayectos forman un conjunto conexo.*

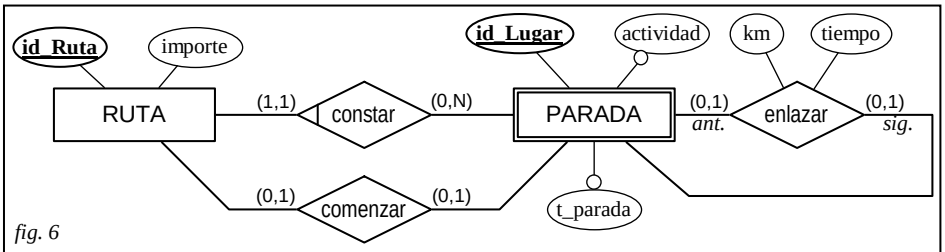

fig. 6

RESTRICCIONES:

- *Verificar que toda ruta comienza en la parada de la ruta que no tiene una parada anterior.*
- *Verificar que la siguiente parada de una ruta pertenece a la misma ruta.*
- *Verificar que todas las paradas de una ruta excepto una (la primera) tienen una parada anterior, y todas excepto una (la última) tienen una parada siguiente; es decir, que no hay ciclos y que los trayectos forman un conjunto conexo*

Otra solución interesante consiste en identificar las paradas con un único atributo identificador (por ejemplo un número), de modo que las paradas de una ruta son todas las que son "sucesoras" de la primera parada de la ruta. Esto permite simplificar algo el esquema E/R (véase fig.7)

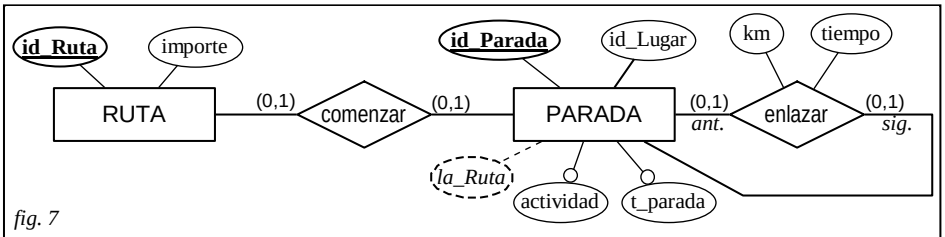

fig. 7

- *donde* la_Ruta *es, si la parada no tiene parada anterior, el valor de* id_Ruta *de la ruta que comienza en dicha parada. En caso contrario, el valor de* id_Ruta *de la parada anterior*

RESTRICCIONES:

- *Verificar que toda parada que no tiene una parada anterior es la parada de comienzo de una ruta.*
- *Verificar que ninguna parada es sucesora de sí misma (no hay ciclos).*

o también:

- *Verificar que toda parada de comienzo de una ruta no tiene una parada anterior.*
- *Verificar que toda parada es, o bien de comienzo de una ruta, o "sucesora" de la parada de comienzo de una ruta (no hay ciclos).*

ESQUEMAS RELACIONALES

ESQUEMAS DE RELACION *corresp. al esquema E/R de la* ***fig.1***

```
Ruta (
  Id_Ruta : tpNombre;
  importe : entero, NO NULO);

Parada (
  id_Ruta : tpNombre, clave ajena de Ruta;
  num_parada: entero;
  id_Lugar : tpNombre, NO NULO;
  actividad : tpActividad;
  t_parada : entero;
  Km : entero, NO NULO;
  tiempo : entero, NO NULO);
```

☛ *verificar que* km=0 *y* tiempo=0 *para toda parada de una ruta con mínimo valor de num_parada.*

ESQUEMAS DE RELACION *corresp. al esquema E/R de la* ***fig.2*** *y al E/R de la* ***fig.3***

```
Ruta (
  Id_Ruta : tpNombre;
  importe : entero, NO NULO;
  pSalida : entero;
  (Id_Ruta, pSalida), clave ajena de Parada);
```

☛ *verificar que* pSalida *es el menor valor de* num_parada *en* Parada *para todas las paradas de esa ruta*

```
Parada (
  id_Ruta : tpNombre, clave ajena de Ruta;
  num_parada: entero;
  id_Lugar : tpNombre, NO NULO;
  actividad : tpActividad;
  t_parada : entero;
  Km : entero, NO NULO;
  tiempo : entero, NO NULO);
```

☛ *verificar que para la parada de salida de cada una de las rutas,* km=0 *y* tiempo=0

ESQUEMAS DE RELACION *corresp. al esquema E/R de la* ***fig.4*** *(propagación de clave hacia parada anterior)*

```
Ruta (
  Id_Ruta : tpNombre;
  importe : entero, NO NULO);

Parada (
  id_Ruta : tpNombre, clave ajena de Ruta;
  id_Lugar : tpNombre;
  sigParada : tpNombre;
  actividad : tpActividad;
  t_parada : entero;
  Km : entero;
  tiempo : entero;
  (id_Ruta, sigParada), UNICO, clave ajena de Parada);
```

☛ *verificar que sii* sigParada *es NULO,* km *y* tiempo *son NULOS.*

☛ *verificar que para toda parada de una ruta, excepto una (la primera) existe otra parada anterior, de la misma ruta ≡ para todo* id_Ruta *de* Parada, *hay* <u>*una y sólo una*</u> *parada que permite acceder al resto de paradas de la ruta.*

☛ *Obsérvese que la especificación de que (*id_Ruta, sigParada*) es UNICO equivale a decir que cada parada puede tener, a lo sumo, una parada anterior.*

ESQUEMAS DE RELACION *corresp. al esquema E/R de la* ***fig.5*** *y al E/R de la* ***fig.6***

```
Ruta (
  Id_Ruta : tpNombre;
  importe : entero, NO NULO;
  pSalida : tpNombre;
  (Id_Ruta, pSalida), clave ajena de Parada);
```

☛ *verificar que* (Id_Ruta, pSalida) *es la primera parada de la ruta (se puede acceder al resto de paradas de la ruta).*

```
Parada (
  id_Ruta : tpNombre, clave ajena de Ruta;
  id_Lugar : tpNombre;
  sigParada : tpNombre;
  actividad : tpActividad;
  t_parada : entero;
  Km : entero;
  tiempo : entero;
  (id_Ruta, sigParada), UNICO, clave ajena de Parada);
```

☛ *verificar que sii* sigParada *es NULO,* km *y* tiempo *son NULOS.*

☛ *verificar que para todo* id_Ruta *de* Parada, pSalida *en Ruta es NO NULO.*

☛ *verificar que toda parada de una ruta tiene una parada anterior, o es la primera de la ruta.*

ESQUEMAS DE RELACION *corresp. al esquema E/R de la* ***fig.7***

```
Ruta (
  Id_Ruta : tpNombre;
  importe : entero, NO NULO;
  pSalida : entero, UNICO, clave ajena de Parada);

Parada (
  id_Parada : entero;
  id_Lugar  : tpNombre, NO NULO;
  sigParada : tpNombre, UNICO, clave ajena de Parada;
  actividad : tpActividad;
  t_parada : entero;
  Km : entero;
  tiempo  : entero;
  la_Ruta : tpNombre);
```

☛ *verificar que sii* sigParada *es NULO,* km *y* tiempo *son NULOS.*

☛ *verificar que para todo* id_Parada *de* Parada, *o bien existe ese valor para* pSalida *en* Ruta, *o para* sigParada *en* Parada

☛ ***Si*** *existe* Ruta [Id_Ruta] | Ruta.pSalida = id_Parada
entonces la_Ruta *es el valor de* Id_Ruta
si no la_Ruta *es el valor de* la_Ruta *de la parada anterior*
FSi

☛ *Obsérvese que esta solución, si bien es la más simple de implementar, tiene el inconveniente de que es algo más costoso obtener la ruta a la que pertenece una parada concreta (hay que recorrer las paradas anterores hasta llegar a la primera. Es por esto por lo que se ha añadido el atributo derivado id_Ruta (la última restricción especifica cómo obtener su valor)*

Nota: Todas las restricciones de los esquemas relacionales anteriores se han expresado de modo que resulte fácil entender su significado, en vez de expresarlas de un modo más formal (se propone como ejercicio), por lo que es posible que haya alguna redundancia.

Ejercicio: Modificar los esquemas E/R y relacionales anteriores (si es necesario) para representar *circuitos turísticos*; es decir, recorridos en los que el lugar de llegada es el mismo que el lugar de salida.