Posible Solución

Aclaración: los predicados coloreados con azul son los predicados principales y obligatoriamente deben ser **totalmente inversibles** (según lo solicitado en el enunciado).

Base de conocimiento inicial

a)

Se modelan a las canciones como functores ya que de esta se conoce tanto el nombre como su duración, por lo tanto necesitamos representarlas usando un individuo compuesto que en este caso es un functor.

```
%canta(nombreCancion, cancion)%
canta(megurineLuka, cancion(nightFever, 4)).
canta(megurineLuka, cancion(foreverYoung, 5)).
canta(hatsuneMiku, cancion(tellYourWorld, 4)).
canta(gumi, cancion(foreverYoung, 4)).
canta(gumi, cancion(tellYourWorld, 5)).
canta(seeU, cancion(novemberRain, 6)).
canta(seeU, cancion(nightFever, 5)).
```

No se agrega a kaito a la base de conocimiento inicial ya que no sabe cantar ninguna canción por lo tanto no podemos relacionarlo con una canción.

Cantantes

1)

```
novedoso(Cantante) :-
    sabeAlMenosDosCanciones(Cantante),
    tiempoTotalCanciones(Cantante, Tiempo),
    Tiempo < 15.

sabeAlMenosDosCanciones(Cantante) :-
    canta(Cantante, UnaCancion),
    canta(Cantante, OtraCancion),
    UnaCancion \= OtraCancion.
```

```
tiempoTotalCanciones(Cantante, TiempoTotal) :-
     findall(TiempoCancion,
     tiempoDeCancion(Cantante, TiempoCancion), Tiempos),
     sumlist(Tiempos,TiempoTotal).

tiempoDeCancion(Cantante,TiempoCancion):-
       canta(Cantante,Cancion),
       tiempo(Cancion,TiempoCancion).

tiempo(cancion(_, Tiempo), Tiempo).
```

2)

```
acelerado(Cantante) :-
     vocaloid(Cantante),
     not((tiempoDeCancion(Cantante,Tiempo),Tiempo > 4)).

vocaloid(Cantante):-
     canta(Cantante, _).
```

Si nos permitieran usar el forall/2 acá nos quedaría en vez del not/1:
`forall(tiempoDeCancion(Cantante,Tiempo), Tiempo <= 4).`

”No canta una canción que dure más de 4 minutos” (not/1) es lo mismo que “Todas sus canciones duran 4 o menos minutos” (forall/2).

Conciertos

1)

Se modelan los conciertos como functores. Hay 3 tipos distintos de conciertos que son: gigante/2, mediano/1 y diminuto/1.
Se agrega la base de conocimiento lo siguiente:

```
%concierto(nombre, pais, fama, tipoConcierto)%
concierto(mikuExpo, eeuu, 2000, gigante(2,6)).
concierto(magicalMirai, japon, 3000, gigante(3,10)).
concierto(vocalektVisions, eeuu, 1000, mediano(9)).
```

```
concierto(mikuFest, argentina, 100, diminuto(4)).
```

2)

```
puedeParticipar(hatsuneMiku,Concierto):-
    concierto(Concierto, _, _, _).

puedeParticipar(Cantante, Concierto):-
    vocaloid(Cantante),
    Cantante \= hatsuneMiku,
    concierto(Concierto, _, _, Requisitos),
    cumpleRequisitos(Cantante, Requisitos).

cumpleRequisitos(Cantante, gigante(CantCanciones,
TiempoMinimo)):-
    cantidadCanciones(Cantante, Cantidad),
    Cantidad >= CantCanciones,
    tiempoTotalCanciones(Cantante, Total),
    Total > TiempoMinimo.

cumpleRequisitos(Cantante, mediano(TiempoMaximo)):-
    tiempoTotalCanciones(Cantante, Total),
    Total < TiempoMaximo.

cumpleRequisitos(Cantante, diminuto(TiempoMinimo)):-
    canta(Cantante, Cancion),
    tiempo(Cancion, Tiempo),
    Tiempo > TiempoMinimo.

cantidadCanciones(Cantante, Cantidad) :-
    findall(Cancion, canta(Cantante, Cancion),
Canciones),
    length(Canciones, Cantidad).
```

3)

```
masFamoso(Cantante) :-
    nivelFamoso(Cantante, NivelMasFamoso),
    forall(nivelFamoso(_, Nivel), NivelMasFamoso >=
Nivel).

nivelFamoso(Cantante, Nivel):-
    famaTotal(Cantante, FamaTotal),
    cantidadCanciones(Cantante, Cantidad),
    Nivel is FamaTotal * Cantidad.

famaTotal(Cantante, FamaTotal):-
    vocaloid(Cantante),
    findall(Fama, famaConcierto(Cantante, Fama),
    CantidadesFama),
    sumlist(CantidadesFama, FamaTotal).

famaConcierto(Cantante, Fama):-
    puedeParticipar(Cantante,Concierto),
    fama(Concierto, Fama).

fama(Concierto,Fama):-
    concierto(Concierto,_,Fama,_).
```

4)

```
conoce(megurineLuka, hatsuneMiku).
conoce(megurineLuka, gumi).
conoce(gumi, seeU).
conoce(seeU, kaito).

unicoParticipanteEntreConocidos(Cantante,Concierto):-
    puedeParticipar(Cantante, Concierto),
    not((conocido(Cantante, OtroCantante),
    puedeParticipar(OtroCantante, Concierto))).
```

%Conocido directo

```
conocido(Cantante, OtroCantante) :-
    conoce(Cantante, OtroCantante).
```

%Conocido indirecto

```
conocido(Cantante, OtroCantante) :-
    conoce(Cantante, UnCantante),
    conocido(UnCantante, OtroCantante).
```

5)

En la solución planteada habría que agregar una claúsula en el predicado cumpleRequisitos/2 que tenga en cuenta el nuevo functor con sus respectivos requisitos

El concepto que facilita los cambios para el nuevo requerimiento es el ***polimorfismo***, que nos permite dar un tratamiento en particular a cada uno de los conciertos en la cabeza de la cláusula.