BASES DE DATOS

2º Grado Ingeniería Informática
Curso 2021-2022

PRÁCTICA 2: Base de datos de cine

Álvaro Seral Gracia
819425@unizar.es

Cristian Andrei Selivanov Dobrisan
816456@unizar.es

Dorian Boleslaw Wozniak
817570@unizar.es

Entregado el 2 de mayo de 2022

1. Creación de la base de datos

1.1. Esquema Entidad-Relación

1.1.1. Diagrama

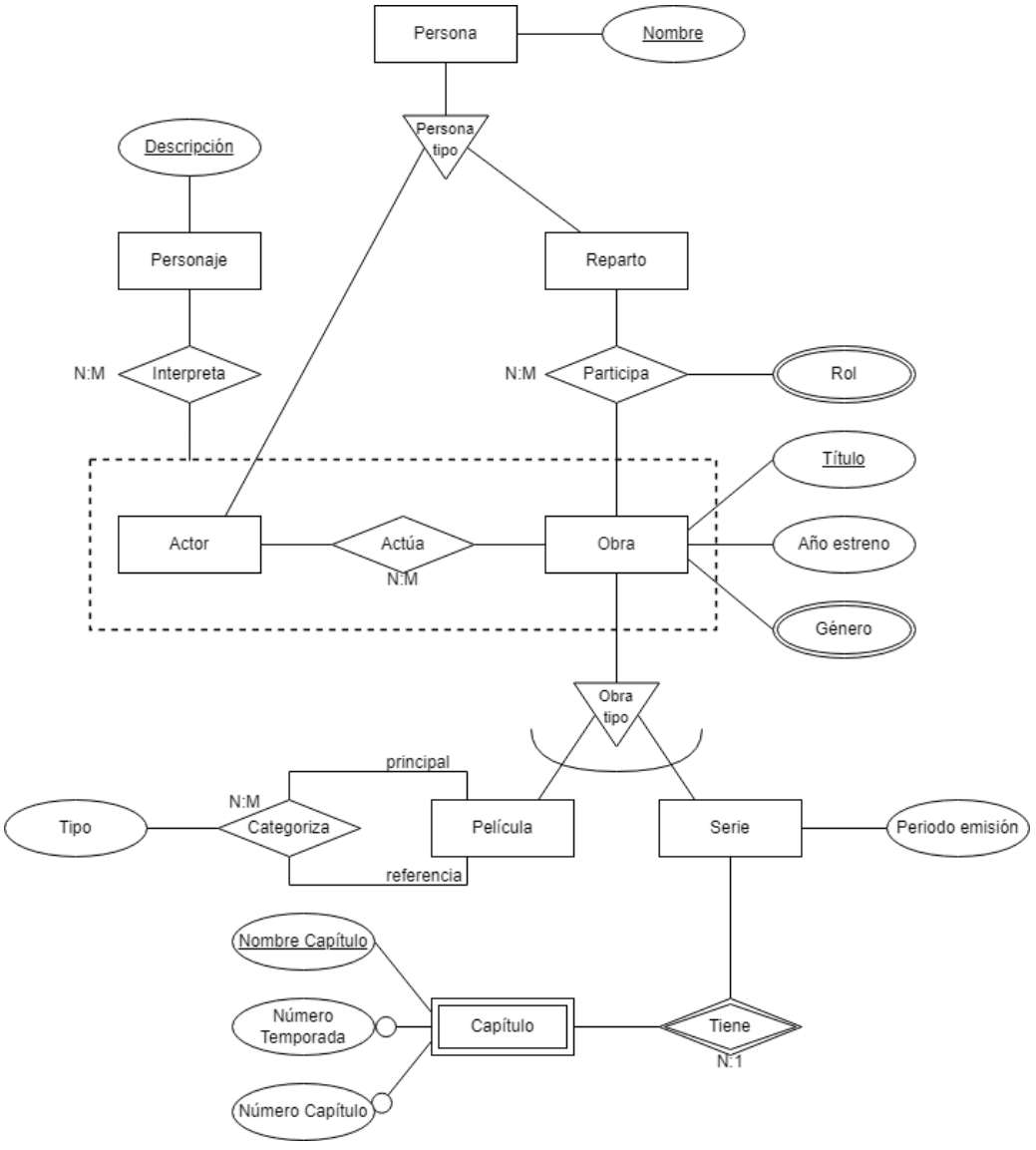

Fig.1 : Diagrama del esquema Entidad-Relación

1.1.2. Restricciones

- El Rol de Labor que cumple una persona como reparto para una Obra no puede ser ‘actor’ o ‘actriz’
- Cualquier identificador de la tabla de Películas no puede estar como identificador en la tabla de Series; y viceversa.
- Una película no puede ser precuela/secuela/remake de sí misma.
- Si una película ‘Uno’ es remake de una película ‘Dos’, entonces ‘Dos’ no puede ser remake de ‘Uno’. Lo mismo con precuelas y secuelas.
- Si una película ‘Uno’ es precuela de una película ‘Dos’, entonces ‘Dos’ es secuela de ‘Uno’; y viceversa.

1.1.3. Consideraciones

Anteriormente, la distribución de temporadas y capítulos estaba formada por medio de una serie de entidades débiles enlazadas, de modo que una serie tuviese temporadas, y una temporada tuviese capítulos. Sin embargo, nos decantamos por dejar una única entidad capítulo que tuviera el número de temporada como atributo ya que, conceptualmente hablando, cuando alguien se refiere a un capítulo se refiere principalmente por su nombre y número, y menormente por su temporada.

A la hora de categorizar ciertas películas como precuelas, secuelas o remakes, se consideró que esos tres tipos eran excluyentes entre sí, de modo que si una película ‘Uno’ es precuela de ‘Dos’, entonces ‘Uno’ no ni secuela ni remake de ‘Dos’. Es por este entendimiento que se ha decidido poner esa relación como una única unaria en vez de dos unarias, una para precuelas y secuelas y otra para remakes. Además, también se entiende que en una saga, una película es precuela de todas las posteriores y secuela de todas las anteriores, y no únicamente de las que directamente preceden o postdecen. Esta disposición de los datos facilita las consultas ya que en todo momento una película tiene constancia de todas las demás que forman parte su misma saga.

Es importante remarcar que para este modelo hemos utilizado el esquema E/R extendido, que ha permitido diseñar un sistema conceptualmente más fácil de entender. Sin embargo, aunque al escoger un modelo extendido es más eficiente en memoria, las consultas son más lentas frente a uno sin dichos elementos.

Al momento de realizar el modelo extendido, se ha tenido en cuenta la exclusión que hay entre películas y series, de modo que una obra que es serie no puede ser película, y una obra que es película no puede ser serie. Esto se ha marcado mediante el arco que corta las relaciones de Obra con Película y Serie. En cambio, ni la relación de Persona con Actor y Reparto ni la relación de Obra con Película y Serie son de participación total.

Finalmente es preciso comentar la dificultad encontrada para traducir los elementos del modelo E/R extendido al relacional, principalmente la interrelación Interpreta con la agrupación, que al final resultó ser una relación que tenía como clave principal la

composición entre la tupla que referencia a Actúa y el identificador que referencia a Personaje.

1.2. Esquema relacional

1.2.1. Esquema relacional inicial

A partir del esquema E-R realizado se ha desarrollado el siguiente esquema relacional:

DOMINIOS

tpNombre = cadena(50);
tpDescripcion = cadena(100);
tpObra = cadena(150);
tpAño = number(4);
tpNatural

ESQUEMA RELACIONAL

Persona (
idPersona : tpNatural;
Nombre : tpNombre, NO NULO);
Reparto (
idReparto : tpNatural, clave ajena de **Persona**);
Actor (
idActor : tpNatural, clave ajena de **Persona**);
Personaje (
idPersonaje : tpNatural;
Descripción : tpDescripcion, NO NULO);
Obra (
idObra : tpNatural;
Título : tpObra, NO NULO;
Estreno : tpAño, NO NULO;
Género : tpNombre, NO NULO);
** verificar que Estreno >= 0*
Participa (
clvReparto : tpNatural, clave ajena de **Reparto**;
clvObra : tpNatural, clave ajena de **Obra**;
Rol : tpNombre, NO NULO);
** verificar que Rol <> 'actor' && Rol <> 'actora'*
Actúa (
clvActor : tpNatural, clave ajena de **Actor**;
clvObra : tpNatural, clave ajena de **Obra**);
Interpreta (
clvActor, clvObra: tpNatural;
clvPersonaje : tpNatural, clave ajena de **Personaje**;
(clvActor, clvObra) clave ajena de **Actúa**);
Película (
idPelicula : tpNatural, clave ajena de **Obra**);
** verificar que idPelicula no aparece en Serie*
Categoriza (
clvPrincipal, clvReferencia : clave ajena de **Película**;
Tipo : tpNombre, NO NULO);
** verificar que clvPrincipal != clvReferencia*

Serie (
idSerie : tpNatural, clave ajena de **Obra**;
Periodo : tpAño, NO NULO);
** verificar que idSerie no aparece en Película*
Capítulo (
idCapitulo : tpNatural;
NombreC : tpNombre, NO NULO;
NumC, NumT : tpNatural;
clvSerie: tpNatural, NO NULO, clave ajena de **Serie**);

1.2.2. Normalización a forma normal Boyce-Codd

Se ha transformado el esquema relacional anterior normalizando sus relaciones hasta verificar que esté en Tercera Forma Normal Boyce-Codd. Así, se han comprobado una por una cada una de las formas normales hasta llegar a la deseada.

Para que las relaciones estén en **Primera Forma Normal**, se deben eliminar los atributos multivaluados Rol (de Participa) y Género (de Obra). Para ello, se han creado dos nuevas relaciones débiles Labor y Tema, de modo que sus claves primarias estén compuestas por la clave primaria de Participa u Obra respectivamente, y por Rol o Género. De esta manera, se han conseguido eliminar los atributos multivaluados, haciendo que se cumpla la Primera Forma Normal.

Como todos los atributos que no forman parte de la clave primaria o candidata dependen de la clave completa, está en **Segunda Forma Normal**.

Además, como todos los atributos que no son clave dependen transitivamente de la clave, también se encuentra en **Tercera Forma Normal**.

Finalmente, también verifica la **Forma Normal Boyce-Codd** ya que todas las dependencias funcionales dependen de claves.

1.2.3. Esquema relacional normalizado

Teniendo en cuenta los cambios descritos anteriormente, se obtiene el siguiente esquema:

ESQUEMA RELACIONAL
Persona (
idPersona : tpNatural;
Nombre : tpNombre, NO NULO);
Reparto (
idReparto : tpNatural, clave ajena de **Persona**);
Actor (
idActor : tpNatural, clave ajena de **Persona**);
Personaje (
idPersonaje : tpNatural;
Descripción : tpDescripcion, NO NULO);
Obra (

idObra : tpNatural;
Título : tpObra, NO NULO;
Estreno : tpAño, NO NULO);
** verificar que Estreno >= 0*

Tema (
Género : tpNombre;
clvObra : tpNatural, clave ajena de **Obra**);

Participa (
clvReparto : tpNatural, clave ajena de **Reparto**;
clvObra : tpNatural, clave ajena de **Obra**);

Labor (
Rol : tpNombre;
clvReparto, clvObra : tpNatural;
(clvReparto, clvObra) clave ajena de **Participa**);
** verificar que Rol <> 'actor' && Rol <> 'actora'*

Actúa (
clvActor : tpNatural, clave ajena de **Actor**;
clvObra : tpNatural, clave ajena de **Obra**);

Interpreta (
clvActor, clvObra: tpNatural;
clvPersonaje : tpNatural, clave ajena de **Personaje**;
(clvActor, clvObra) clave ajena de **Actúa**);

Película (
idPelicula : tpNatural, clave ajena de **Obra**);
** verificar que idPelicula no aparece en Serie*

Categoriza (
clvPrincipal, clvReferencia : clave ajena de **Película**;
Tipo : tpNombre, NO NULO);
** verificar que clvPrincipal != clvReferencia*

Serie (
idSerie : tpNatural, clave ajena de **Obra**;
Periodo : tpAño, NO NULO);
** verificar que idSerie no aparece en Película*

Capítulo (
idCapitulo : tpNatural;
NombreC : tpNombre, NO NULO;
NumC, NumT : tpNatural;
clvSerie: tpNatural, NO NULO, clave ajena de **Serie**);

1.3. Sentencias SQL de creación

Una vez definida la estructura de la base de datos, se definen las siguientes sentencias SQL para la creación de las tablas necesarias para implementar el modelo:

```
CREATE TABLE Persona (
    clvPersona      NUMBER          PRIMARY KEY,
    Nombre          VARCHAR(50)     NOT NULL
);

CREATE TABLE Reparto (
    clvReparto      NUMBER          PRIMARY KEY REFERENCES Persona(clvPersona)
);

CREATE TABLE Actor (
    clvActor        NUMBER          PRIMARY KEY REFERENCES Persona(clvPersona)
```

```
);

CREATE TABLE Personaje (
    clvPersonaje    NUMBER          PRIMARY KEY,
    Descripcion     VARCHAR(100)    NOT NULL
);

CREATE TABLE Obra (
    clvObra         NUMBER          PRIMARY KEY,
    Titulo          VARCHAR(150)    NOT NULL,
    Estreno         NUMBER(4)       NOT NULL CHECK (Estreno >= 0)
);


CREATE TABLE Tema (
    Genero          VARCHAR(50),
    clvObra         NUMBER          REFERENCES Obra(clvObra),
    PRIMARY KEY (Genero, clvObra)
);

CREATE TABLE Participa (
    clvReparto      NUMBER          REFERENCES Reparto(clvReparto),
    clvObra         NUMBER          REFERENCES Obra(clvObra),
    PRIMARY KEY (clvReparto, clvObra)
);

CREATE TABLE Labor (
    Rol             VARCHAR(100)    CHECK (Rol<>'actor' AND Rol<>'actress'),
    clvReparto      NUMBER,
    clvObra         NUMBER,
    PRIMARY KEY (Rol, clvReparto, clvObra),
    FOREIGN KEY (clvReparto, clvObra) REFERENCES Participa(clvReparto, clvObra)
);

CREATE TABLE Actua (
    clvActor        NUMBER          REFERENCES Actor(clvActor),
    clvObra         NUMBER          REFERENCES Obra(clvObra),
    PRIMARY KEY (clvActor, clvObra)
);

CREATE TABLE Interpreta (
    clvActor        NUMBER,
    clvObra         NUMBER,
    clvPersonaje    NUMBER          REFERENCES Personaje(clvPersonaje),
    PRIMARY KEY (clvActor, clvObra, clvPersonaje),
    FOREIGN KEY (clvActor, clvObra) REFERENCES Actua(clvActor, clvObra)
);

CREATE TABLE Pelicula (
    clvPelicula      NUMBER          PRIMARY KEY REFERENCES Obra(clvObra)
);

CREATE TABLE Categoriza (
    clvPrincipal    NUMBER          REFERENCES Pelicula(clvPelicula),
    clvReferencia   NUMBER          REFERENCES Pelicula(clvPelicula),
    Tipo            VARCHAR(100)    NOT NULL,
    PRIMARY KEY (clvPrincipal, clvReferencia),
    CHECK (clvPrincipal <> clvReferencia)
);

CREATE TABLE Serie (
    clvSerie        NUMBER          PRIMARY KEY REFERENCES Obra(clvObra),
    Periodo         VARCHAR(9)      NOT NULL
);
```

```sql
CREATE TABLE Capitulo (
    clvCapitulo     NUMBER          PRIMARY KEY,
    NombreC         VARCHAR(100)    NOT NULL,
    NumC            NUMBER          CHECK (NumC >= 0),
    NumT            NUMBER          CHECK (NumT >= 0),
    clvSerie        NUMBER          NOT NULL REFERENCES Serie(clvSerie)
);
```

2. Introducción de datos y ejecución de consultas

2.1. Procedimiento de población

Inicialmente fue necesario obtener las tablas completas requeridas por el esquema relacional. Esto se realizó escribiendo las consultas necesarias para que, a partir de la base de datos ofrecida por los profesores, se sacaran las tablas requeridas. Para poder insertar los valores de estas a la base de datos, se utilizó la herramienta MySQL Workbench para exportar los archivos .sql de inserción de datos para las tablas solicitadas.

Para poder poblar toda la base de datos a partir de un único fichero, se creó otro archivo main.sql de modo que este ejecutase en orden cada uno de dichos ficheros. Además, para que la tabla no se eliminase al cerrar sesión en Hendrix, se añadió la instrucción “commit;” a main.sql una vez se hubo asegurado que todos los datos eran correctos y no daban problemas.

Entre las dificultades encontradas al hacer la inserción de los datos en la base, hay que destacar 2 secuencias halladas en strings que dieron particularmente problemas: <\”> y <&>. La primera se solucionó eliminando dicha secuencia de la fila, mientras que para la segunda se sustituyeron todas sus apariciones por <AND>. Además, también se tuvieron que eliminar ciertas filas de las tablas de Categoriza y Capítulo cuyos atributos hacían referencia a claves no existentes de Películas y Series respectivamente.

2.2. Consulta 1

El objetivo de la primera consulta es obtener el porcentaje de películas con hasta 3 actores o actrices.

Sentencia SQL

```
-- (Número de películas con 3 actores o más /
-- Número de películas totales) * 100
SELECT (seccion/total) * '100' Porcentaje
FROM
(
    -- Número de películas con 3 actores o más
    SELECT count(*)
    AS seccion
    FROM
    (
        -- Tabla con películas con 3 actores o más
        SELECT P.clvPelicula
        FROM Pelicula P, Actua A;
```

```
        -- La obra en la que actua un Actor es una Película
        WHERE P.clvPelicula=A.clvObra
        -- Se agrupa por Películas
        GROUP BY P.clvPelicula
        -- 3 >= Número de Actores de la Película (agrupación)
        HAVING '3' >= count(A.clvActor)
    )
),
(
    -- Número de películas totales
    SELECT count(P.clvPelicula)
    AS total
    FROM Pelicula P
);
```

Árbol sintáctico

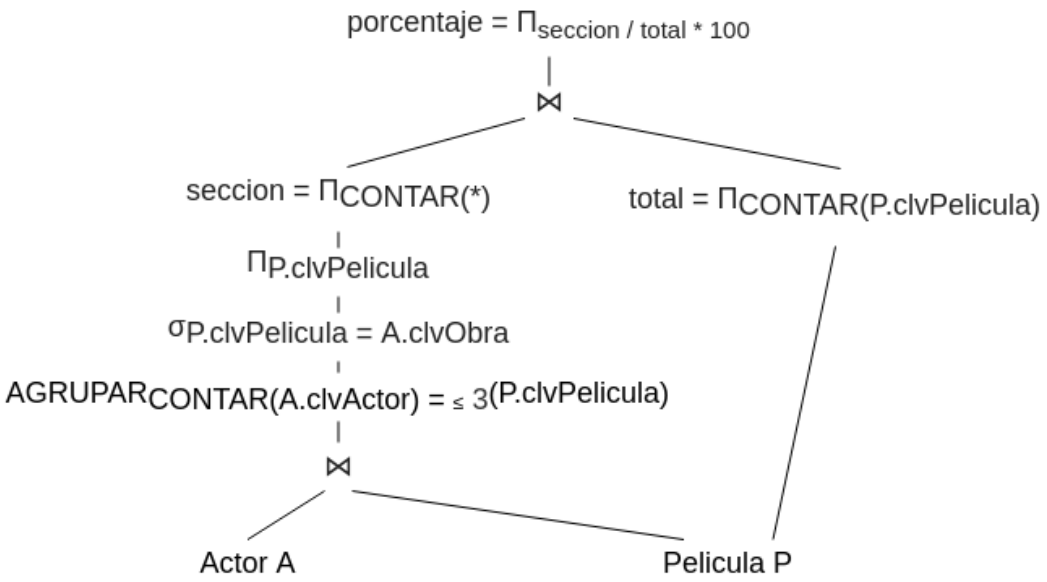

Fig.2: Árbol sintáctico de la consulta 1

Resultado

```
PORCENTAJE
----------
15,3300026
```

2.3. Consulta 2

La segunda consulta consiste en obtener el título de las películas y series de terror en las que el mismo personaje ha sido interpretado por al menos 50 diferentes actores o actrices, junto con el nombre del personaje y el número de actores distintos que lo han interpretado.

Sentencia SQL

```
-- Titulo de la Obra, Descripción del Personaje,
-- Número de Actores que lo interpretan en la Película
SELECT DISTINCT
      O.Titulo Titulo_Obra,
      Pj.Descripcion Descripcion_Personaje,
      count(I.clvActor) Numero_Actores
FROM Obra O, Personaje Pj, Interpreta I, Tema T
WHERE
      O.clvObra = I.clvObra AND
      -- El Personaje interpretado es de la Obra
      Pj.clvPersonaje = I.clvPersonaje AND
      -- El Genero a buscar está asociado a la Obra
      O.clvObra = T.clvObra AND
      -- El Género es de Terror (o semejantes)
      -- Al utilizar <%> antes y después de las palabras, cogemos
      -- cualquier género que tenga esa secuencia. En terror solo está
      -- al principio ya que todos los nuevos Géneros que surgirían al
      -- poner <%> después son de terrorismo
      (
            T.Genero LIKE '%terror' OR
            T.Genero LIKE '%horror%' OR
            T.Genero LIKE '%fear%'
      )
-- Se agrupa por Obras, Personaje y Tema
GROUP BY O.Titulo, Pj.Descripcion, T.Genero
-- 50 <= Número de Actores de la Obra con ese Tema y ese Personaje
-- (agregación)
HAVING '50' <= count(I.clvActor);
```

Árbol sintáctico

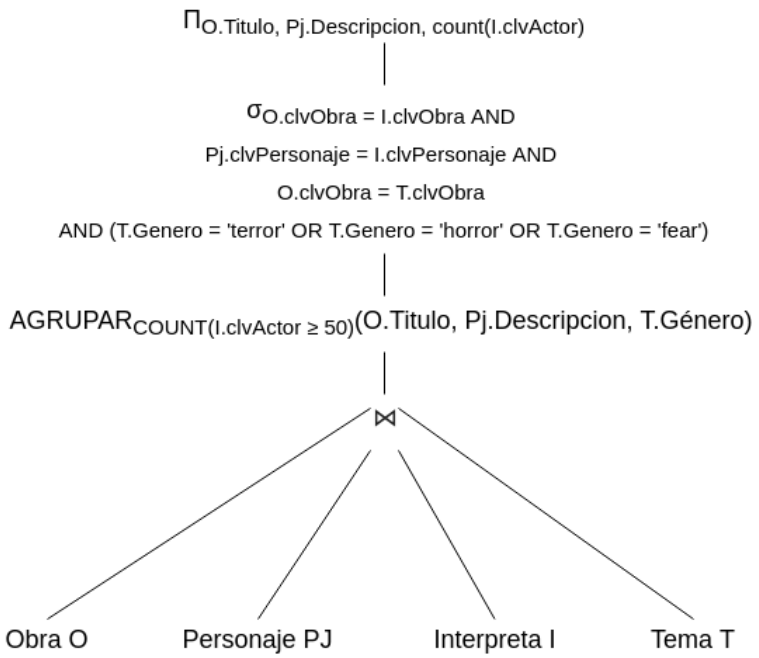

Fig. 3: Árbol sintáctico de la consulta 2

Resultado

```
TITULO_OBRA                   DESCRIPCION_PERSONAJE          NUMERO_ACTORES
----------------------------- ------------------------------ --------------
Angustia                      New cinema spectator                       56
```

2.4. Consulta 3

La última consulta obtiene los actores que solo han participado en una película de la década de los 90 que forma parte de una saga de al menos tres películas.

Sentencia SQL

```sql
-- Actores que solo han participado en una Película durante la
-- década de los 90
CREATE VIEW vistaActor AS
    SELECT A.clvActor
    FROM Pelicula P, Obra O, Actua A
    WHERE
        -- La fecha de Estreno de la Obra es de la década de los 90
        (O.Estreno BETWEEN '1990' AND '1999') AND
        -- La Obra es una Película
        O.clvObra = P.clvPelicula AND
```

```
    -- El actor ha participado en la Película
    A.clvObra = P.clvPelicula
    -- Se agrupa por Actor
    GROUP BY A.clvActor
    -- 1 = Número de veces que el Actor ha actuado (agregación)
    HAVING '1' = count(A.clvActor);

-- Nombre del Actor
SELECT P.Nombre
FROM Persona P, vistaActor V, Categoriza C, Actua A, Obra O
WHERE
    -- El Actor se encuentra en la vista anterior
    V.clvActor = A.clvActor AND
    P.clvPersona = A.clvActor AND
    -- La obra es de la década de los 90
    (O.Estreno BETWEEN '1990' AND '1999') AND
    -- El Actor ha actuado en la Obra
    O.clvObra = A.clvObra AND
    -- La Obra es una Película que además tiene otra Película
    -- que Categoriza
    A.clvObra = C.clvPrincipal AND
    -- La Película es precuela o secuela de otra
    C.Tipo IN ('follows', 'followed by')
-- Se agrega por Actor y Película
GROUP BY P.Nombre, O.Titulo
-- 2 <= Número de veces que la Película es Precuela o Saga (agregación)
HAVING '2' <= count(C.clvPrincipal);

DROP VIEW vistaActor;
```

Árbol sintáctico

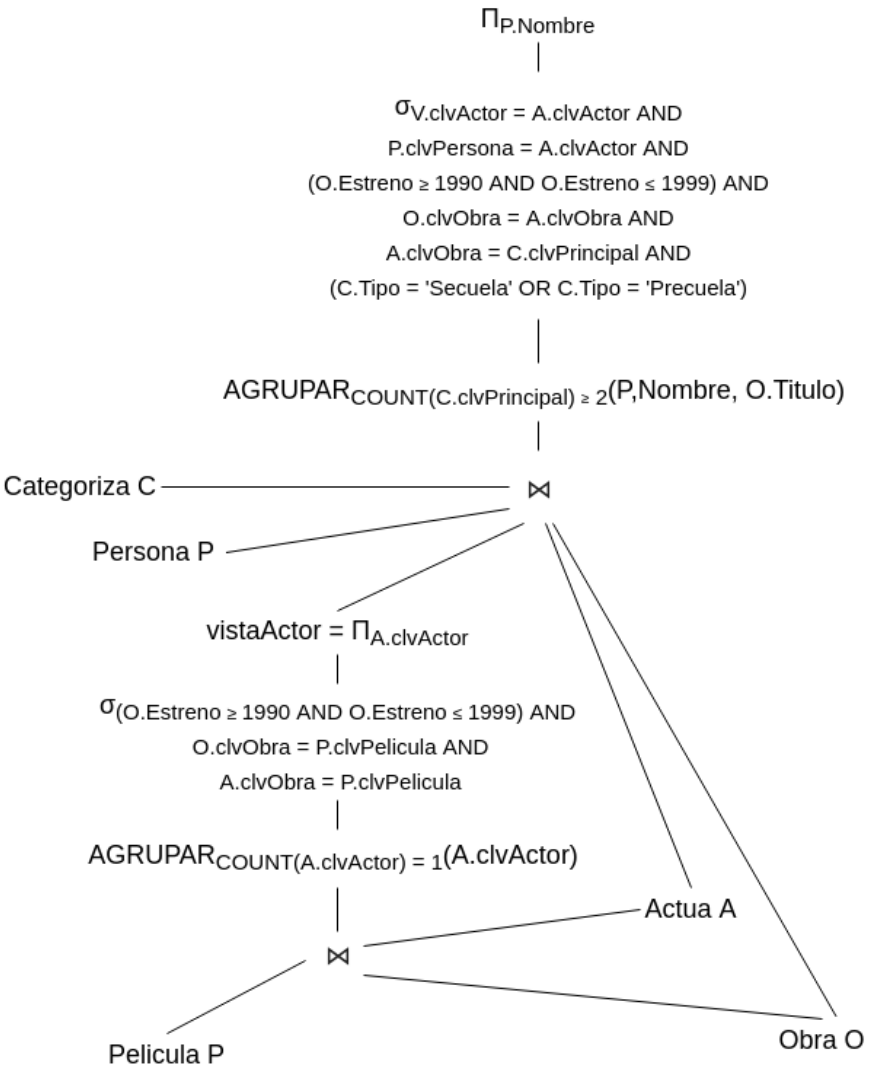

Fig. 4. Árbol sintáctico de la consulta 3

Resultado

```
NOMBRE
-----------------------------
Lucas, Jorge R.
Ortiz, Marcos
Ponce, Ester
Arraud, Mariano
Bl??zquez, ??lvaro
Gallo, Jes??s
```

```
Mullen, Rory
Ranz, Fernando
Rodr??guez, Luis Miguel
Garc??a, Palomo
L??pez, Santos
Dill, Marcus
Pica, Antonio
Mosquera, Carmen
Guerrero, Francisco
Ramo, Carlos
Fontana, Rosa
Quiroga, ??lvaro
Molero Alfonso, Marta
Urtado, Alberto

20 filas seleccionadas.
```

3. Diseño físico

3.1. Rendimiento de las consultas

3.1.1. Consulta 1

En una primera evaluación se observaron resultados bastante satisfactorios con un coste en CPU que tenía como tope 77. Sin embargo, después de realizar algunas pruebas, se consiguió mejorar la consulta reduciendo el coste máximo a 52. Esto fue posible creando un índice sobre el atributo clvObra de Actua, permitiendo identificar más rápidamente las tuplas de dicha tabla, evitando hacer “joins” en el proceso.

A continuación se puede observar como la consulta 1 inicial ya mostraba unos datos relativamente satisfactorios.

```
Plan hash value: 3256087011

-----------------------------------------------------------------------------------------
| Id  | Operation                  | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |               |     1 |    26 |    77   (6)| 00:00:01 |
|   1 |  NESTED LOOPS              |               |     1 |    26 |    77   (6)| 00:00:01 |
|   2 |   VIEW                     |               |     1 |    13 |    72   (6)| 00:00:01 |
|   3 |    SORT AGGREGATE          |               |     1 |       |            |          |
|   4 |     VIEW                   |               | 46201 |       |    72   (6)| 00:00:01 |
|*  5 |      HASH GROUP BY         |               | 46201 |  1173K|    72   (6)| 00:00:01 |
|   6 |       NESTED LOOPS         |               | 46201 |  1173K|    70   (3)| 00:00:01 |
|   7 |        INDEX FAST FULL SCAN| SYS_C00322103 | 46201 |   586K|    68   (0)| 00:00:01 |
|*  8 |        INDEX UNIQUE SCAN   | SYS_C00322109 |     1 |    13 |     0   (0)| 00:00:01 |
|   9 |   VIEW                     |               |     1 |    13 |     5   (0)| 00:00:01 |
|  10 |    SORT AGGREGATE          |               |     1 |       |            |          |
|  11 |     INDEX FAST FULL SCAN   | SYS_C00322109 |  3803 |       |     5   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   5 - filter(COUNT(*)<=3)
   8 - access("P"."CLVPELICULA"="A"."CLVOBRA")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
```

Para optimizar dicha consulta, tal y como hemos explicado anteriormente, se declarará el siguiente índice.

```
CREATE INDEX ind_act_clvObra on Actua(clvObra);
```

Podemos observar como la implementación de índices en este caso mejora considerablemente una consulta que ya era relativamente eficiente antes.

```
Plan hash value: 2394518143

-------------------------------------------------------------------------------------------------
| Id  | Operation                  | Name            | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                 |     1 |    26 |    52   (8)| 00:00:01 |
|   1 |  NESTED LOOPS              |                 |     1 |    26 |    52   (8)| 00:00:01 |
|   2 |   VIEW                     |                 |     1 |    13 |    47   (9)| 00:00:01 |
|   3 |    SORT AGGREGATE          |                 |     1 |       |            |          |
|   4 |     VIEW                   |                 | 46201 |       |    47   (9)| 00:00:01 |
|*  5 |      HASH GROUP BY         |                 | 46201 |  1173K|    47   (9)| 00:00:01 |
|   6 |       NESTED LOOPS         |                 | 46201 |  1173K|    45   (5)| 00:00:01 |
|   7 |        INDEX FAST FULL SCAN| IND_ACT_CLVOBRA | 46201 |   586K|    43   (0)| 00:00:01 |
|*  8 |        INDEX UNIQUE SCAN   | SYS_C00322109   |     1 |    13 |     0   (0)| 00:00:01 |
|   9 |   VIEW                     |                 |     1 |    13 |     5   (0)| 00:00:01 |
|  10 |    SORT AGGREGATE          |                 |     1 |       |            |          |
|  11 |     INDEX FAST FULL SCAN   | SYS_C00322109   |  3803 |       |     5   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

  5 - filter(COUNT(*)<=3)
  8 - access("P"."CLVPELICULA"="A"."CLVOBRA")

Note
-----
  - dynamic statistics used: dynamic sampling (level=2)
```

3.1.2. Consulta 2

Para la segunda consulta se observaron unos resultados complacientes, con un coste máximo de 89, que no se pudieron mejorar, ya que a la hora de añadir índices no mejoraría dicho coste.

```
Plan hash value: 1500459203

-----------------------------------------------------------------------------------------
| Id  | Operation              | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |            |   348 | 85608 |    89   (3)| 00:00:01 |
|   1 |  HASH UNIQUE           |            |   348 | 85608 |    89   (3)| 00:00:01 |
|*  2 |   FILTER               |            |       |       |            |          |
|   3 |    HASH GROUP BY       |            |   348 | 85608 |    89   (3)| 00:00:01 |
|*  4 |     HASH JOIN          |            |   348 | 85608 |    88   (2)| 00:00:01 |
|*  5 |      HASH JOIN         |            |   348 | 62988 |    66   (2)| 00:00:01 |
```

```
|*  6 |       HASH JOIN         |            |    50 |  7750 |    31   (0)| 00:00:01 |
|*  7 |        TABLE ACCESS FULL| TEMA       |    50 |  3250 |    22   (0)| 00:00:01 |
|   8 |        TABLE ACCESS FULL| OBRA       |  5682 |   499K|     9   (0)| 00:00:01 |
|   9 |       TABLE ACCESS FULL | INTERPRETA | 39729 |  1008K|    34   (0)| 00:00:01 |
|  10 |      TABLE ACCESS FULL  | PERSONAJE  | 18565 |  1178K|    22   (0)| 00:00:01 |
-------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - filter(COUNT(*)>=50)
   4 - access("PJ"."CLVPERSONAJE"="I"."CLVPERSONAJE")
   5 - access("O"."CLVOBRA"="I"."CLVOBRA")
   6 - access("O"."CLVOBRA"="T"."CLVOBRA")
   7 - filter("T"."GENERO" LIKE '%terror' OR "T"."GENERO" LIKE '%horror%' OR
              "T"."GENERO" LIKE '%fear%')

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - this is an adaptive plan
```

3.1.3. Consulta 3

Esta consulta fue diseñada originalmente con una vista, de modo que devolvía un coste de 227. Al convertir la vista normal en una vista materializada, se consiguió reducir el coste tope a 157, a base de almacenar de manera física dicha vista.

A continuación se mostrará la explicación de rendimiento ofrecida por la base de datos de Oracle sobre la consulta 3 base, con vistas sin materializar.

```
Plan hash value: 1008859153

-----------------------------------------------------------------------------------------------
| Id  | Operation                      | Name          | Rows  | Bytes | Cost (%CPU)| Time   |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT               |               |  7422 |   934K|   227   (1)| 00:00:01 |
|*  1 |  FILTER                        |               |       |       |            |        |
|   2 |   HASH GROUP BY                |               |  7422 |   934K|   227   (1)| 00:00:01 |
|   3 |    VIEW                        | VM_NWVW_1     |  7422 |   934K|   227   (1)| 00:00:01 |
|*  4 |     FILTER                     |               |       |       |            |        |
|   5 |      HASH GROUP BY             |               |  7422 |  2696K|   227   (1)| 00:00:01 |
|*  6 |       HASH JOIN                |               |  7422 |  2696K|   226   (1)| 00:00:01 |
|*  7 |        TABLE ACCESS FULL       | OBRA          |  2789 | 72514 |     9   (0)| 00:00:01 |
|   8 |        NESTED LOOPS            |               | 10121 |  3419K|   217   (1)| 00:00:01 |
|*  9 |         HASH JOIN              |               | 10121 |  3291K|   217   (1)| 00:00:01 |
|* 10 |          HASH JOIN             |               |  1274 |   381K|   149   (1)| 00:00:01 |
|* 11 |           HASH JOIN            |               |  1274 |   286K|    80   (0)| 00:00:01 |
|* 12 |            HASH JOIN           |               |  1523 |   171K|    71   (0)| 00:00:01 |
|* 13 |             TABLE ACCESS FULL  | CATEGORIZA    |   110 |  8470 |     3   (0)| 00:00:01 |
|  14 |             INDEX FAST FULL SCAN| SYS_C00322103 | 46201 |  1714K|    68   (0)| 00:00:01 |
|* 15 |            TABLE ACCESS FULL   | OBRA          |  2789 |   313K|     9   (0)| 00:00:01 |
|  16 |           TABLE ACCESS FULL    | PERSONA       | 40469 |  3043K|    68   (0)| 00:00:01 |
|  17 |          INDEX FAST FULL SCAN  | SYS_C00322103 | 46201 |  1173K|    68   (0)| 00:00:01 |
```

```
|* 18 |         INDEX UNIQUE SCAN       | SYS_C00322109 |     1 |    13 |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   1 - filter(COUNT(*)>=2)
   4 - filter(COUNT(*)=1)
   6 - access("O"."CLVOBRA"="P"."CLVPELICULA")
   7 - filter("O"."ESTRENO">=1990 AND "O"."ESTRENO"<=1999)
   9 - access("A"."CLVACTOR"="A"."CLVACTOR")
  10 - access("P"."CLVPERSONA"="A"."CLVACTOR")
  11 - access("O"."CLVOBRA"="A"."CLVOBRA")
  12 - access("A"."CLVOBRA"="C"."CLVPRINCIPAL")
  13 - filter("C"."TIPO"='followed by' OR "C"."TIPO"='follows')
  15 - filter("O"."ESTRENO">=1990 AND "O"."ESTRENO"<=1999)
  18 - access("A"."CLVOBRA"="P"."CLVPELICULA")

Note
-----
   - dynamic statistics used: dynamic sampling (level=2)
   - this is an adaptive plan
```

El siguiente código sería el resultante de materializar la vista utilizada originalmente en nuestra consulta.

```
CREATE MATERIALIZED VIEW vistaActor
BUILD IMMEDIATE
REFRESH COMPLETE
ON DEMAND
AS
   SELECT A.clvActor
   FROM Pelicula P, Obra O, Actua A
   WHERE
       (O.Estreno BETWEEN '1990' AND '1999') AND
       O.clvObra = P.clvPelicula AND
       A.clvObra = P.clvPelicula
   GROUP BY A.clvActor
   HAVING '1' = count(A.clvActor);
```

Se puede verificar que la materialización de la misma da lugar a unas consultas más eficientes en costes, ya que está almacenado en memoria física, suponiendo un empeoramiento en la eficiencia de memoria. Cabe destacar que los datos no siempre serán los más actualizados ya que la vista materializada a diferencia de las normales, no son una consulta directa de las tablas que consultan en el momento, esto último no se considera como un problema muy grave en nuestra base de datos, ya que no se tiene previsto añadir datos en la posterioridad, mucho menos de películas antiguas…

En la siguiente tabla se puede observar cómo la materialización de nuestra vista mejora el rendimiento.

```
Plan hash value: 691568779

-------------------------------------------------------------------------------------------
```

```
| Id  | Operation                  | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-----------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |               |  1274 |   338K|   157   (2)| 00:00:01 |
|*  1 |  FILTER                    |               |       |       |            |          |
|   2 |   HASH GROUP BY            |               |  1274 |   338K|   157   (2)| 00:00:01 |
|*  3 |    HASH JOIN               |               |  1274 |   338K|   156   (1)| 00:00:01 |
|*  4 |     HASH JOIN              |               |  1274 |   257K|    87   (0)| 00:00:01 |
|*  5 |      HASH JOIN             |               |  1274 |   241K|    80   (0)| 00:00:01 |
|*  6 |       HASH JOIN            |               |  1523 |   135K|    71   (0)| 00:00:01 |
|*  7 |        TABLE ACCESS FULL   | CATEGORIZA    |   110 |  7150 |     3   (0)| 00:00:01 |
|   8 |        INDEX FAST FULL SCAN| SYS_C00322103 | 46201 |  1173K|    68   (0)| 00:00:01 |
|*  9 |       TABLE ACCESS FULL    | OBRA          |  2789 |   280K|     9   (0)| 00:00:01 |
|  10 |      MAT_VIEW ACCESS FULL  | VISTAACTOR    | 10494 |   133K|     7   (0)| 00:00:01 |
|  11 |     TABLE ACCESS FULL      | PERSONA       | 40469 |  2568K|    68   (0)| 00:00:01 |
-----------------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

  1 - filter(COUNT(*)>=2)
  3 - access("P"."CLVPERSONA"="A"."CLVACTOR")
  4 - access("V"."CLVACTOR"="A"."CLVACTOR")
  5 - access("O"."CLVOBRA"="A"."CLVOBRA")
  6 - access("A"."CLVOBRA"="C"."CLVPRINCIPAL")
  7 - filter("C"."TIPO"='followed by' OR "C"."TIPO"='follows')
  9 - filter("O"."ESTRENO">=1990 AND "O"."ESTRENO"<=1999)

Note
-----
  - dynamic statistics used: dynamic sampling (level=2)
  - this is an adaptive plan
```

3.2. Restricciones no verificables

En este apartado se trataran dichas restricciones que no se pueden comprobar desde la propia base de datos de Oracle a la hora de crear las tablas con “checks”, debido a que se deben verificar consistencias de los datos introducidos por la tupla entre dos tablas y/o buscar la tupla recíproca dentro de la propia tabla.

Las restricciones de este tipo que se han encontrado en este tipo de base y que serán utilizadas para realizar triggers son:

- Cualquier identificador de la tabla de Películas no puede estar como identificador en la tabla de Series; y viceversa. -> Trigger 1
- Si una película ‘Uno’ es remake de una película ‘Dos’, entonces ‘Dos’ no puede ser remake de ‘Uno’. Lo mismo con precuelas y secuelas. -> Trigger 2
- Si una película ‘Uno’ es precuela de una película ‘Dos’, entonces ‘Dos’ es secuela de ‘Uno’; y viceversa. -> Trigger 3

3.3. Trigger 1

```
-- Cualquier identificador de la tabla de Películas no puede estar como
-- identificador en la tabla de Series
CREATE OR REPLACE TRIGGER trigger1P
BEFORE INSERT ON Pelicula
FOR EACH ROW
DECLARE
    n NUMBER;
BEGIN
    -- n = número de veces que la clvPelicula nueva aparece
    -- en la tabla Serie
    SELECT count(*)
    INTO n
    FROM Serie
    WHERE clvSerie=:NEW.clvPelicula;
    -- Si n>0, ya existe esa la clvPelicula en Series,
    -- lanza un error y no ejecuta el insert
    IF n > 0 THEN
        RAISE_APPLICATION_ERROR(-20001, 'Ya existe una serie con id');
    END IF;
END trigger1P;
/

-- Cualquier identificador de la tabla de Series no puede estar como
-- identificador en la tabla de Películas
CREATE OR REPLACE TRIGGER trigger1S
BEFORE INSERT ON Serie
FOR EACH ROW
DECLARE
    n NUMBER;
BEGIN
    -- n = número de veces que la clvSerie nueva aparece
    -- en la tabla Película
    SELECT count(*)
    INTO n
    FROM Pelicula
    WHERE clvPelicula=:NEW.clvSerie;
    -- Si n>0, ya existe esa la clvSerie en Películas,
    -- lanza un error y no ejecuta el insert
    IF n > 0 THEN
    RAISE_APPLICATION_ERROR(-20002, 'Ya existe una película con esa id');
    END IF;
END trigger1S;
/
```

3.4. Trigger 2

```
-- Si una película 'Uno' es remake de una película 'Dos', entonces 'Dos' no puede ser remake
de 'Uno'. Lo mismo con precuelas y secuelas.
```

```
CREATE OR REPLACE TRIGGER trigger2
BEFORE INSERT ON Categoriza
FOR EACH ROW
DECLARE
    n NUMBER;
BEGIN
    -- n = número de veces que la tupla de Películas opuesta a la nueva es del tipo nuevo
    SELECT count(*)
    INTO n
    FROM Categoriza
    WHERE
        :NEW.clvPrincipal=clvReferencia AND
        :NEW.clvReferencia=clvPrincipal AND
        :NEW.Tipo=Tipo;
    -- Si n>1, ya existe una relación entre las dos películas nuevas del mismo tipo pero
    -- con claves invertidas, lanza un error y no ejecuta el insert
    IF n > 1 THEN
        RAISE_APPLICATION_ERROR(-20003, 'Ya existe el contrareciproco de esta insercion.');
    END IF;
END trigger2;
/
```

3.5. Trigger 3

Es necesario remarcar que dicho trigger se ha realizado para conseguir una mayor consistencia a la hora de insertar los datos en la tabla, añadiendo tuplas que están obligadas a existir en él, ahorrando trabajo al usuario y evitando posibles errores humanos.

```
-- Si una película ‘Uno’ es precuela de una película ‘Dos’, entonces ‘Dos’ es secuela de ‘Uno’;
-- y viceversa.
CREATE OR REPLACE TRIGGER trigger3
AFTER INSERT ON Categoriza
FOR EACH ROW
DECLARE
    n NUMBER;
BEGIN
    -- Si el tipo de inserción es una secuela con respecto a la referencia:
    IF :NEW.Tipo = 'follows' THEN
        -- n = número de veces que la tupla de Películas inversa aparece con el tipo precuela
        SELECT count(*)
        INTO n
        FROM Categoriza
        WHERE
            :NEW.clvPrincipal=clvReferencia AND
            :NEW.clvReferencia=clvPrincipal AND
            Tipo='followed by';
        -- Si n = 0, no hay ninguna aparición, se inserta automáticamente la nueva fila
        -- correspondiente
        IF n = 0 THEN
            INSERT INTO Categoriza (clvPrincipal,clvReferencia,Tipo)
            VALUES (:NEW.clvReferencia,:NEW.clvPrincipal,'followed by');
        END IF;
    END IF;
    -- Si el tipo de inserción es una precuela con respecto a la referencia:
    IF :NEW.Tipo = 'followed by' THEN
        -- n = número de veces que la tupla de Películas inversa aparece con el tipo secuela
        SELECT count(*)
        INTO n
        FROM Categoriza
        WHERE
            :NEW.clvPrincipal=clvReferencia AND
            :NEW.clvReferencia=clvPrincipal AND
            Tipo='follows';
```

```
        -- Si n = 0, no hay ninguna aparición, se inserta automáticamente la nueva fila
        -- correspondiente
        IF n = 0 THEN
            INSERT INTO Categoriza (clvPrincipal,clvReferencia,Tipo)
            VALUES (:NEW.clvReferencia,:NEW.clvPrincipal,'follows');
        END IF;
    END IF;
--
END trigger3;
/
```

4. Anexo: Organización

Tiempo (h)	Alvaro Seral	Cristian Selivanov	Dorian Wozniak	TOTAL SECCIÓN
Creación	5	4	4	13
Población	5	3	2	10
Consultas	10	8	6	24
Optimización	4	4	5	13
Triggers	7	7	3	17
Memoria	5	4	5	14
TOTAL INTEGRANTE	36	30	25	TOTAL: 91