Bases de datos - Práctica 2

Curso 2022

26/04/2022

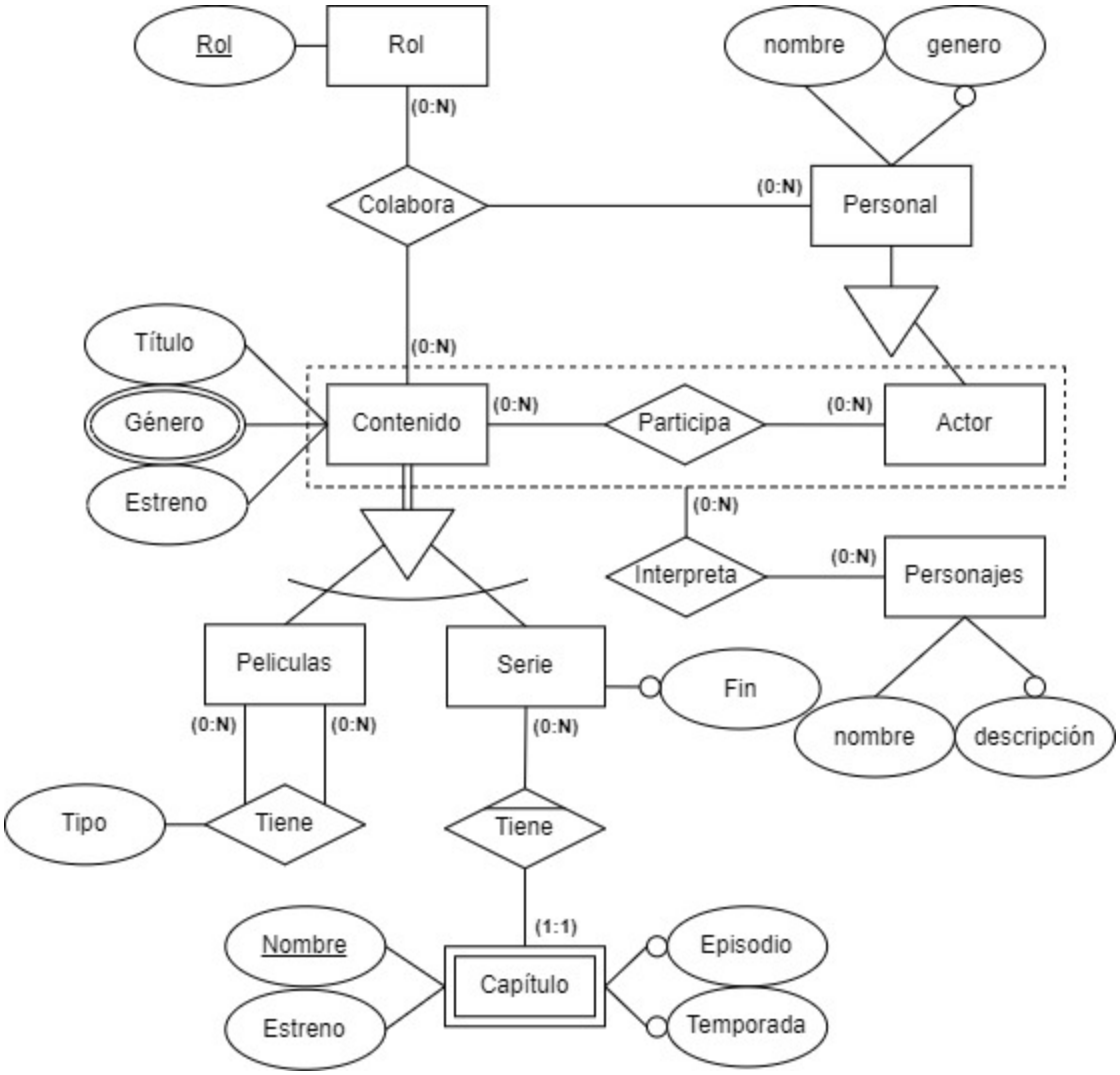
Participantes

Héctor Toral Pallás - 798095@unizar.es
Francisco Javier Pizarro Martinez - 821259@unizar.es
Pablo López Mosqueda - 779739@unizar.es

Parte 1 - Creación de la base de datos

1.1 - Modelo Entidad/Relación

En está práctica se ha de realizar una base de datos para gestionar y almacenar información acerca de una colección de series y películas españolas recopiladas de la base de datos de IMDB, para ello, se ha planteado el siguiente modelo E/R:



Restricciones del modelo:

- Un contenido asignado a la tabla de películas no puede estar a su vez en la tabla de series ni viceversa..
- Todas las fechas deben ser posteriores a 1850.
- Las temporadas y episodios deben ser mayor o igual que 1 si son no nulos.
- La fecha de inicio de una serie debe ser menor o igual que la de finalización.
- Una persona puede estar en la tabla de actores si colabora en un contenido de el rol de actor/a.

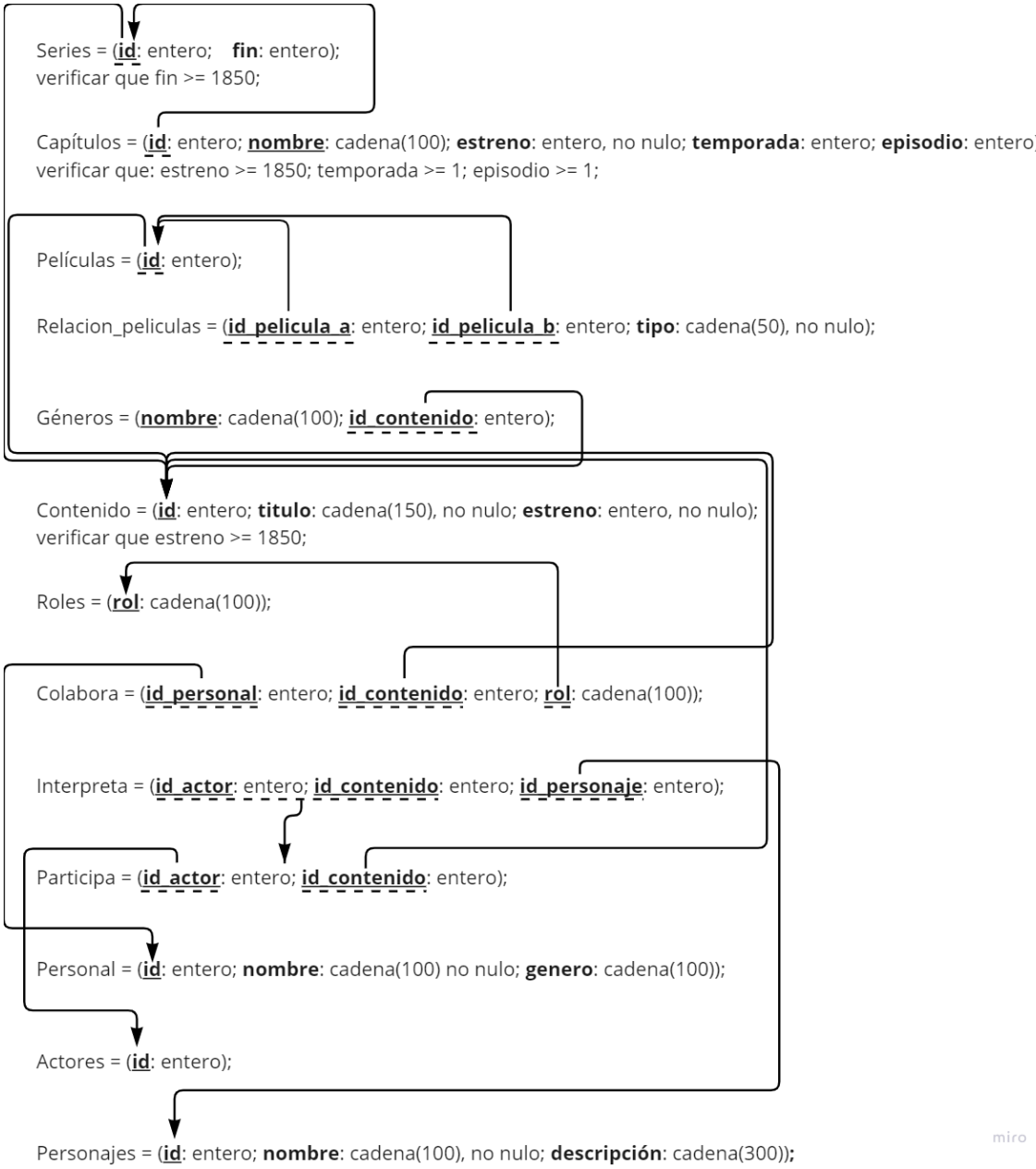
Alternativas a nuestro modelo:

El atributo multivaluado género, podría haber sido modelado como una relación N:M con contenido, pero finalmente fue definido como atributo multivaluado dado que la tabla género no era muy importante, ya que nunca se realizan consultas explícitamente sobre los géneros.

La relación ternaria formada por las entidades personal, contenido y rol, podría haber sido modelada como una relación binaria N:M con un atributo multivaluado rol, basandonos en que en los datos proporcionados para esta base de datos todos los elementos tienen un rol hemos optado por la relación ternaria obligando así a que cualquier persona que colabore en el contenido debe hacerlo mediante al menos un rol.

La agregación del modelo podría haberse sustituido por una ternaria entre contenido, actor y personaje, pero se decidió la agregación de contenido con actor para debilitar la condición de que para cada contenido un actor interpretará un personaje, debido a que en la base de datos que se nos proporcionaba como fuente escaseaban esos datos. También se tuvo en cuenta otras agregaciones como la de contenido con personaje para permitir que hubiera personajes que participarán en el contenido que no fuesen interpretados por ningún actor.

1.2 - Modelo Relacional



El modelo se encuentra en la 1ªFN debido a que los atributos multivaluados han sido traducidos al modelo relacional de manera que estos se han repartido a otras tablas (tabla: géneros).
El modelo se encuentra en la 2ªFN debido a que todos los atributos que forman parte de las claves compuestas no dependen unos de otros.
El modelo se encuentra en la 3ª FN y 3ªFNBC ya que todos los atributos dependen siempre de la clave.

1.3 - Traducción a SQL

```
CREATE TABLE contenido (
  id          NUMBER,
  titulo      VARCHAR2(150) NOT NULL,
  estreno NUMBER            NOT NULL,
  CONSTRAINT pk_Con_id  PRIMARY KEY (id),
  CONSTRAINT ck_Con_estreno CHECK (estreno >= 1850)
);

CREATE TABLE generos (
  nombre          VARCHAR2(100),
  id_contenido  NUMBER,
  CONSTRAINT pk_Gen PRIMARY KEY (nombre, id_contenido),
  CONSTRAINT fk_Gen  FOREIGN KEY (id_contenido) REFERENCES contenido(id)
);

CREATE TABLE peliculas (
  id  NUMBER,
  CONSTRAINT pk_Pel_id  PRIMARY KEY (id),
  CONSTRAINT fk_Pel_id   FOREIGN KEY (id) REFERENCES contenido(id)
);

-- Tiene N:M
CREATE TABLE relacion_peliculas (
  id_pelicula_a NUMBER,
  id_pelicula_b NUMBER,
  tipo              VARCHAR2(50)  NOT NULL,
  CONSTRAINT pk_RP    PRIMARY KEY (id_pelicula_a, id_pelicula_b),
  CONSTRAINT fk_RP_a  FOREIGN KEY (id_pelicula_a) REFERENCES peliculas(id),
  CONSTRAINT fk_RP_b  FOREIGN KEY (id_pelicula_b) REFERENCES peliculas(id)
);

CREATE TABLE series (
  id      NUMBER,
  fin     NUMBER,
  CONSTRAINT pk_Ser_id  PRIMARY KEY (id),
  CONSTRAINT fk_Ser_id   FOREIGN KEY (id)  REFERENCES contenido(id),
  CONSTRAINT ck_Ser_fin CHECK (fin >= 1850)
);
```

```
CREATE TABLE capitulos (
  id              NUMBER,
  nombre      VARCHAR2(100),
  estreno       NUMBER  NOT NULL,
  temporada NUMBER,
  episodio      NUMBER,
  CONSTRAINT pk_Cap_id  PRIMARY KEY (id, nombre),
  CONSTRAINT fk_Cap_id   FOREIGN KEY (id)  REFERENCES series(id),
  CONSTRAINT ck_Cap_estreno        CHECK (estreno >= 1850),
  CONSTRAINT ck_Cap_temporada  CHECK (temporada >= 1),
  CONSTRAINT ck_Cap_episodio       CHECK (episodio >= 1)
);

CREATE TABLE personal (
  id              NUMBER,
  nombre    VARCHAR2(100) NOT NULL,
  genero      VARCHAR2(100),
  CONSTRAINT pk_P_id PRIMARY KEY (id)
);

CREATE TABLE actores (
  id  NUMBER,
  CONSTRAINT pk_Act PRIMARY KEY (id)
);

CREATE TABLE personajes (
  id              NUMBER,
  nombre      VARCHAR2(100) NOT NULL,
  descripcion VARCHAR2(300),
  CONSTRAINT pk_Per_id PRIMARY KEY (id)
);

CREATE TABLE roles (
  rol VARCHAR2(100),
  CONSTRAINT pk_Rol PRIMARY KEY (rol)
);
```

```
CREATE TABLE colabora (
  id_personal    NUMBER,
  id_contenido   NUMBER,
  rol            VARCHAR2(100),
  CONSTRAINT pk_Col PRIMARY KEY (id_personal, id_contenido, rol),
  CONSTRAINT fk_Col_personal   FOREIGN KEY (id_personal)   REFERENCES personal(id),
  CONSTRAINT fk_Col_contenido FOREIGN KEY (id_contenido)  REFERENCES contenido(id),
  CONSTRAINT fk_Col_rol        FOREIGN KEY (rol)          REFERENCES roles(rol)
);

CREATE TABLE participa (
  id_actor      NUMBER,
  id_contenido  NUMBER,
  CONSTRAINT pk_Par            PRIMARY KEY (id_contenido, id_actor),
  CONSTRAINT fk_Par_actor      FOREIGN KEY (id_actor)      REFERENCES actores(id),
  CONSTRAINT fk_Par_contenido FOREIGN KEY (id_contenido)  REFERENCES contenido(id)
);

CREATE TABLE interpreta (
  id_actor      NUMBER,
  id_contenido  NUMBER,
  id_personaje  NUMBER,
  CONSTRAINT pk_Int            PRIMARY KEY (id_actor, id_contenido, id_personaje),
  CONSTRAINT fk_Int_personaje FOREIGN KEY (id_personaje) REFERENCES personajes(id),
  CONSTRAINT fk_Int_participa  FOREIGN KEY (id_actor, id_contenido)  REFERENCES
     participa(id_actor, id_contenido)
);
```

Parte 2 - Introducción de datos y ejecución de consultas

2.1 - Población de la Base de Datos

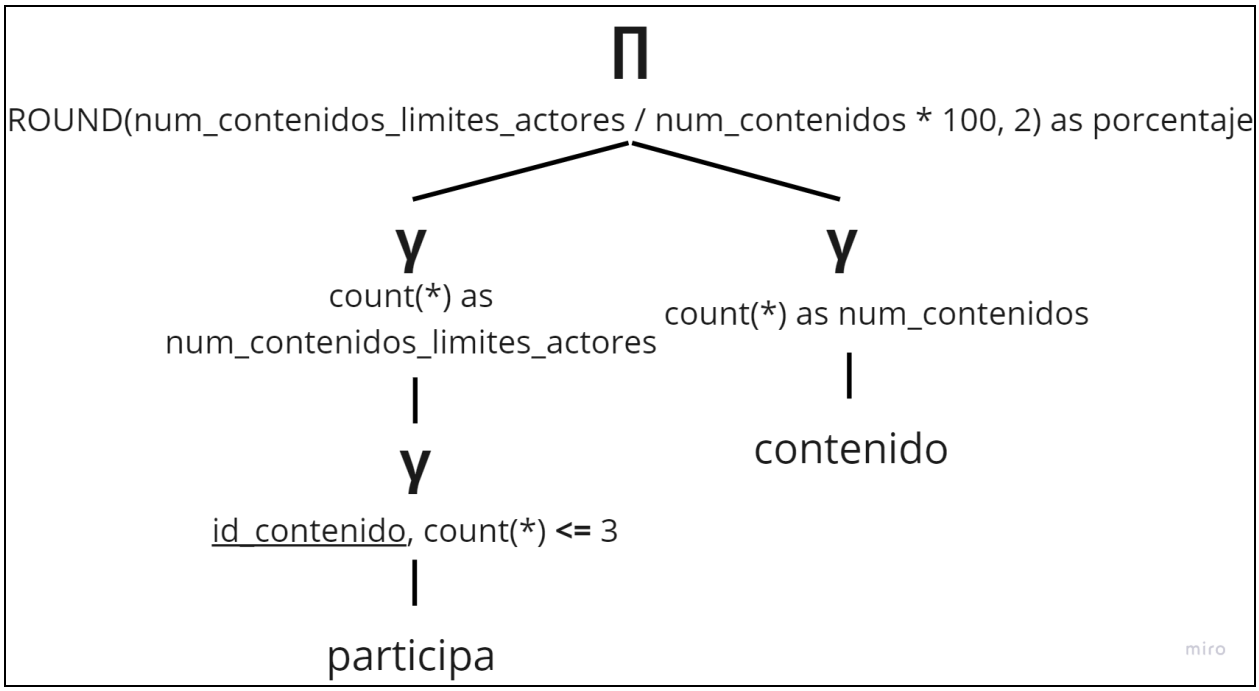
Para empezar con la parte de población se empleó la aplicación MySQL Workbench para extraer la información de la base de datos proporcionada, con la información extraída de las distintas consultas realizadas, se exportaba directamente a formato sql insert.

Después simplemente debíamos modificar los nombres de las tablas y de los valores insertados desde Visual Studio Code empleando el atajo de teclado “ctrl + h”. Recolectando los datos, algunos de los problema encontrado fue el hecho de que había nombres de personajes y del personal que contenían el carácter ” ‘ “ , la solución fue reemplazar dicho carácter por “ ” (1 espacio) ya que el escapado de caracteres también nos dio problemas, también, a la hora de poblar la tabla de géneros tuvimos que tener en cuenta que en la base de datos proporcionada no había géneros como tal y tuvimos que agrupar por palabras claves asignándoles un género concreto.

2.2 - Consultas de la Base de Datos

1. Porcentaje de películas con hasta 3 actores o actrices

```
SELECT ROUND(N.num_conteidos_limite_actores / D.num_contenidos * 100, 2)||'%'
  AS porcentaje
FROM (
  -- Número de contenidos con hasta 3 actores/actrices
  SELECT COUNT(*) AS num_conteidos_limite_actores
  FROM (
    -- Devuelve tuplas en las que el num_actores es <= 3;
    -- num_actores = count(*)
    SELECT id_contenido
    FROM participa
    GROUP BY id_contenido
    HAVING COUNT(*) <= 3
)
) N, (
  -- Devuelve el número total de contenidos
  SELECT COUNT(*) AS num_contenidos
  FROM contenido
) D;
```



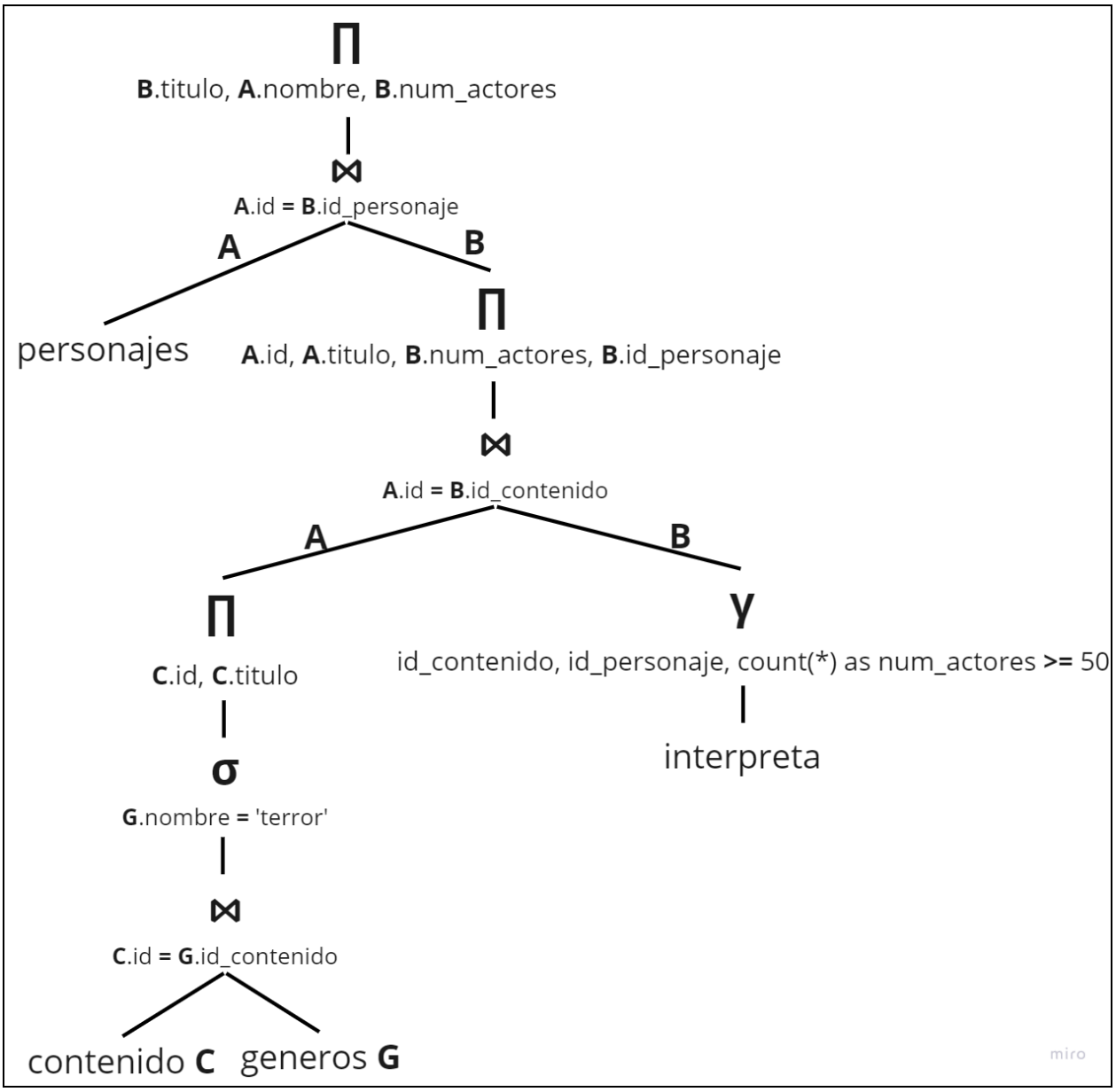
El resultado obtenido por la primera consulta es: 21,01%

2. 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

```
SELECT N.titulo, P.nombre, N.num_actores
FROM personajes P, (
  -- Devuelve tuplas (id, título, num_actores, id personaje) en las que el
  -- contenido es de terror y el número de veces que un personaje es
  -- interpretado dentro del mismo contenido es de al menos 50.
  SELECT A.id, A.titulo, B.num_actores, B.id_personaje
  FROM (
    -- Devuelve una lista de tuplas (id, titulo) de aquellos contenidos que
    -- pertenecen al género de terror.
    SELECT C.id, C.titulo
    FROM contenido C, generos G
    WHERE C.id = G.id_contenido AND G.nombre = 'terror'
  ) A, (
    -- Devuelve tuplas como (número actores, id_contenido, id_ personaje)
    -- en las que el número de actores para interpretar un personaje dentro de
    -- un mismo contenido (película/serie) es al menos de 50.
    SELECT COUNT(*) AS num_actores, id_contenido, id_personaje
    FROM interpreta I
    GROUP BY id_contenido, id_personaje
    HAVING COUNT(*) > 49
  ) B
  WHERE A.id = B.id_contenido
) N
WHERE P.id = N.id_personaje;
```

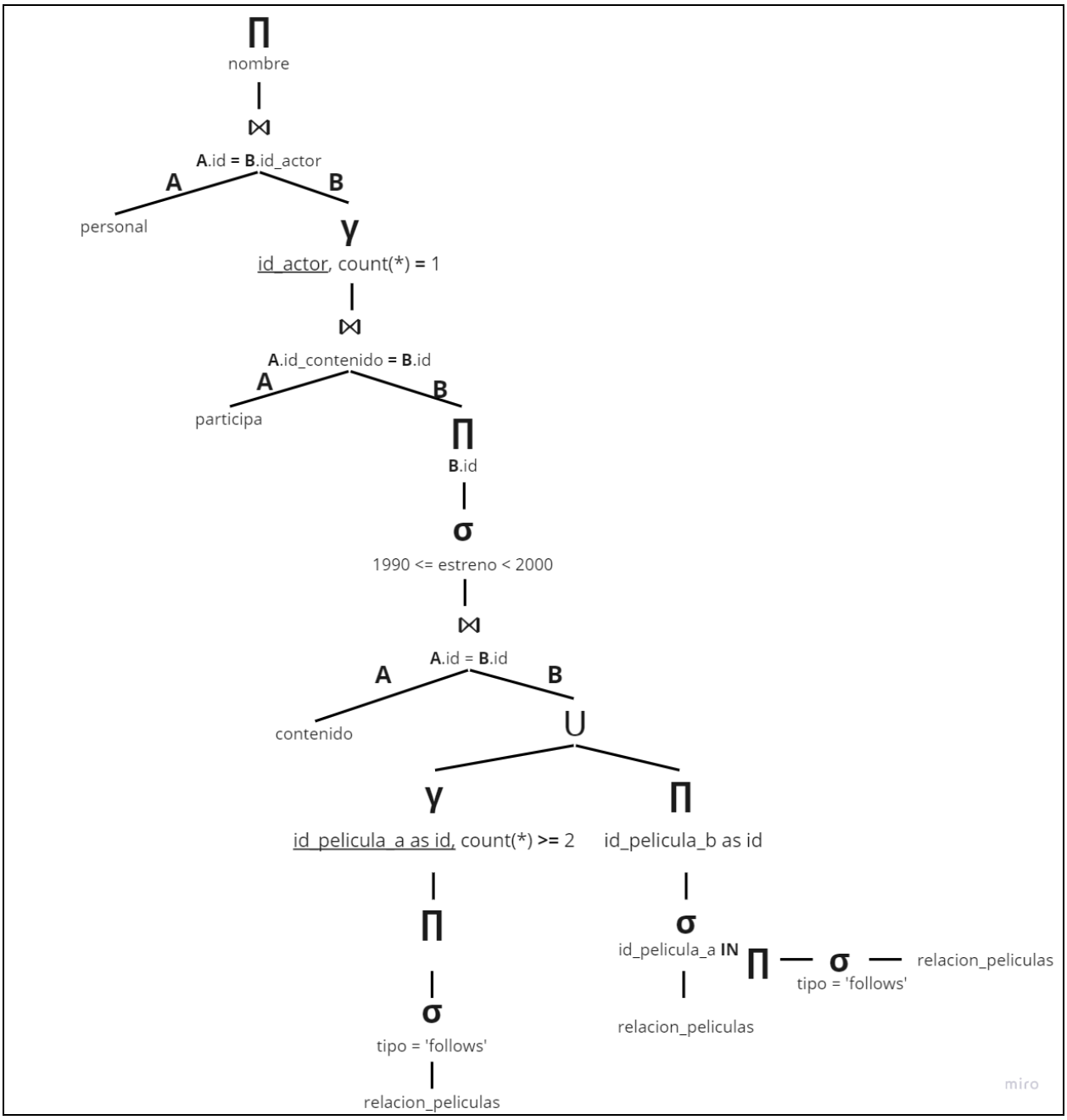
El resultado obtenido por la segunda consulta es:

titulo = Angustia, Nombre = New cinema spectator, Num_actores = 56


Π
B.titulo, A.nombre, B.num_actores
⋈
A.id = B.id_personaje
A
B
personajes
Π
A.id, A.titulo, B.num_actores, B.id_personaje
⋈
A.id = B.id_contenido
A
B
Π
C.id, C.titulo
σ
G.nombre = 'terror'
⋈
C.id = G.id_contenido
contenido C
generos G
γ
id_contenido, id_personaje, count(*) as num_actores >= 50
interpreta
miro

3. 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

```
SELECT nombre
FROM personal P, (
    SELECT id_actor FROM participa P , (
      -- Selecciona de la lista de películas que pertenecen a una saga de al
      -- menos 3 películas, las que se estrenaron en la década de los 90
      SELECT DISTINCT S.id FROM contenido C, (
          -- Saca la lista de sagas con al menos 3 películas
          SELECT id_pelicula_a AS id
          FROM (
              SELECT * FROM relacion_peliculas WHERE tipo = 'follows'
          ) T1
          GROUP BY id_pelicula_a
          HAVING COUNT(*) >= 2
          -- es 2 y no 3 ya que la primera película no se cuenta, se usa como
          -- “identificador” de saga
          UNION ALL
          -- Devuelve las películas que pertenecen a la lista de sagas siendo
          -- "saga" la primera película de la saga
          SELECT id_pelicula_b AS id
          FROM relacion_peliculas
          WHERE id_pelicula_a IN (
              SELECT id_pelicula_a FROM relacion_peliculas
              WHERE tipo = 'follows'
          )
      ) S
      WHERE C.id = S.id AND 1990 <= estreno AND estreno < 2000
    ) S
    WHERE P.id_contenido = S.id GROUP BY id_actor HAVING COUNT(*) = 1
) A
WHERE P.id = A.id_actor;
```



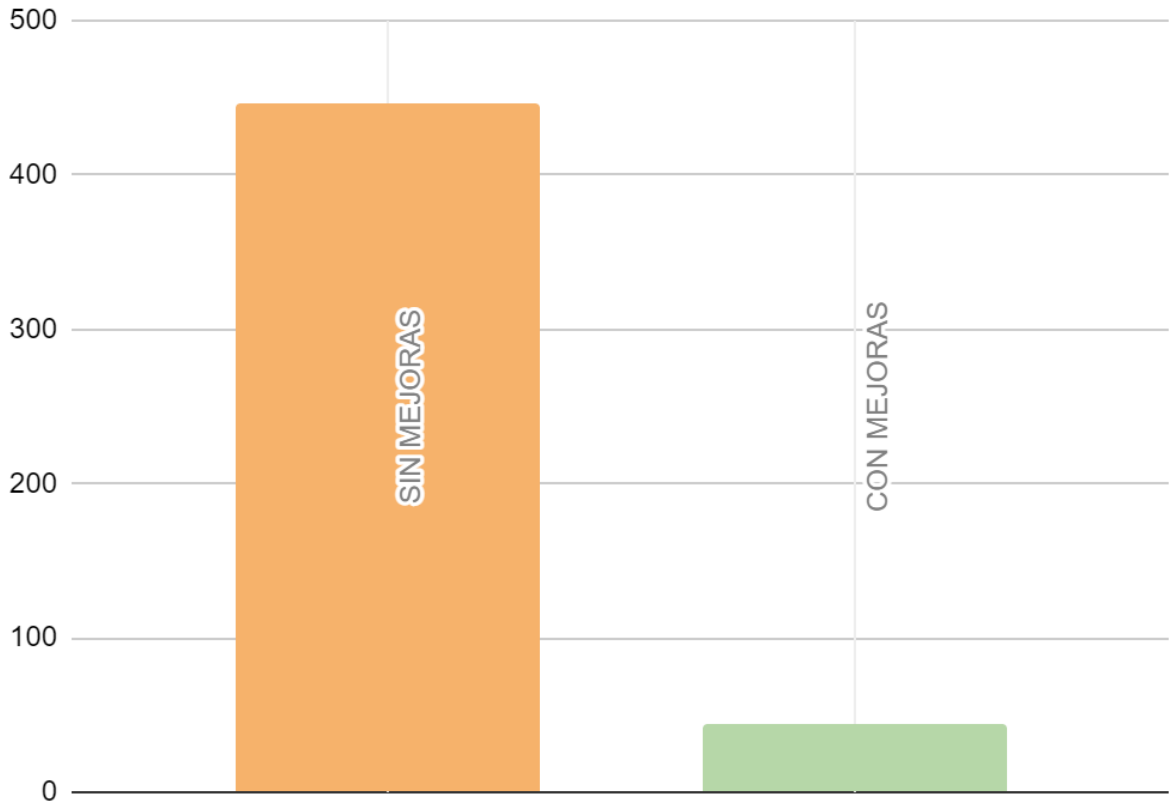
Para la tercera consulta nos salen 187 registros con los nombres de los actores.
En esta consulta tuvimos dudas sobre a que se refiere el enunciado en la parte que dice sólo participará en el conjunto de las sagas de los 90, otra opción que teníamos era que no podía participar en ninguna otra película, lo cual entendimos que como no se refería a eso explícitamente sería erróneo.

Parte 3 - Evaluación de rendimiento y triggers

3.1 - Evaluación de rendimiento

En la primera consulta el coste recae sobre la obtención del dato que indica cuantos contenidos hay con hasta 3 actores y actrices debido a que hay que realizar 2 agregaciones accediendo a toda la tabla ambas veces, dando así un coste de O(n^2) siendo n = 42584 tuplas. En total, la consulta sin mejoras mostraba un coste de 446 realizado en 8ms.

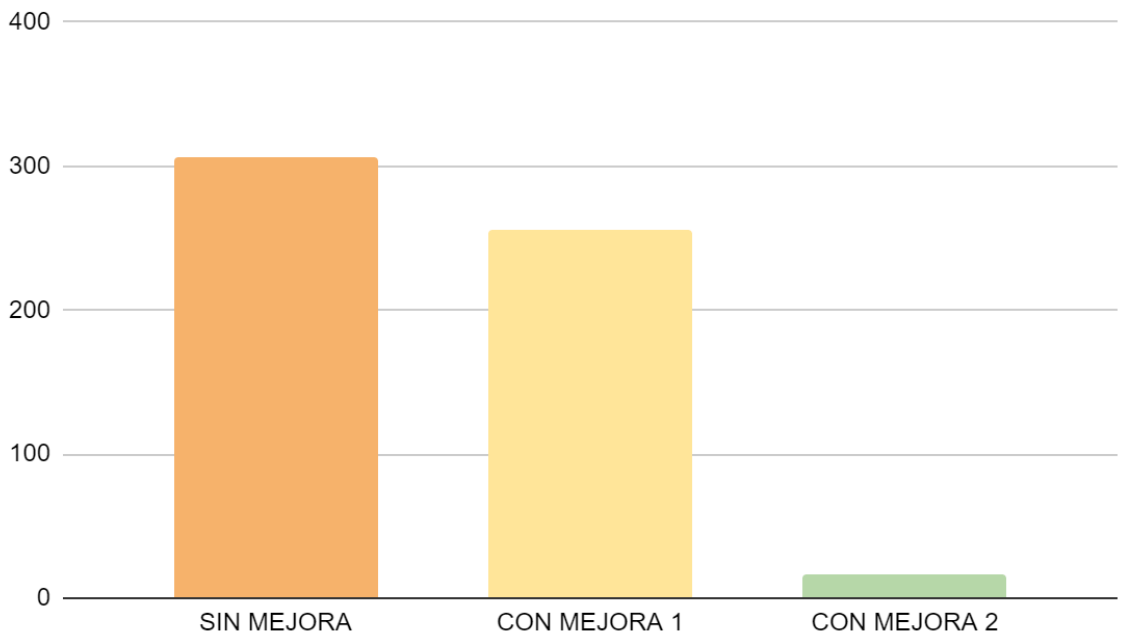
Para intentar mejorar el resultado obtenido se ha decidido tener el dato precalculado mediante el uso de una vista materializada, al crear esta, no se especificó la periodicidad de actualización ya que como es para motivos educativos y no comercial suponemos que no se van a realizar más inserciones evitando así el coste que supondría el tener que recalcular el valor de la tabla cada vez que se realizará una inserción. Finalmente, tras la mejora conseguimos un coste mucho menor, siendo este de 44 obtenido en 6ms.



En la segunda consulta el coste era de 306 obtenido en un tiempo de 9ms, las partes más costosas de las consultas era los accesos completos a las tablas interpreta y personajes, debido a que eran accesos completos a tablas de 41212 y 19596 registros, además, estos solían realizarse para la ejecución de joins lo cual incrementa todavía más su coste.

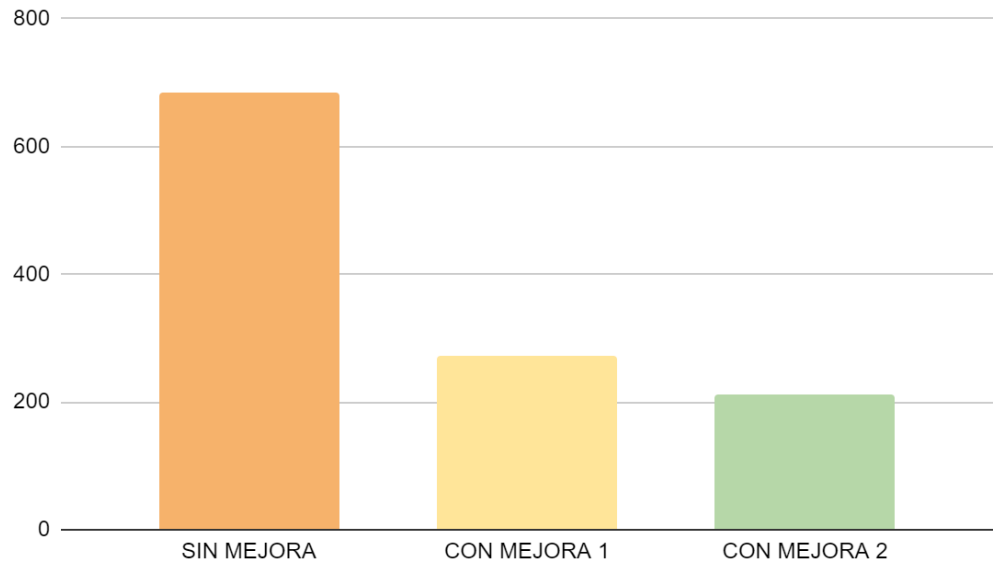
Para intentar rebajar su coste se realizó una primera mejora que redujo el coste a 256 en 7ms, para obtener esta mejoría se optó por desnormalizar el modelo y tener en una misma tabla la información de contenido y de género teniendo así un primer join precalculado, la gestión de los datos del join se delegó en el SGBD mediante el uso de vistas materializadas. Además, se decidió añadir un índice tipo BITMAX sobre la vista para facilitar la búsqueda de las películas por su género.

Finalmente, la última mejora fue dejar precalculado otro join que generaba tuplas en las que el contenido es de terror y el número de veces que se interpretaba a un personaje es al menos de 50. Esto supuso que todo ese coste de cálculo se realizará una sola vez y así la consulta en vez de calcular todo una y otra vez pudiera realizar un acceso completo a tabla con todo datos útiles. Esta mejora final supuso una mejora considerable llevando el coste a 16 en 6ms, esto se debe a que casi todos los datos los tiene ya precalculados.



En la tercera consulta el coste resultó ser de 685 obtenido en 17ms. Para intentar reducir el coste de esta se creó una vista materializada con el contenido de las sagas con al menos de 3 películas para evitar así tener que realizar operaciones de filtrado, esto consiguió reducir el coste de 685 hasta 272.

Finalmente, se realizó otra vista que contiene el id de los actores que participaron en solo una de las películas de las sagas de la década de los 90 dejando así el "resultado final" precalculado para su posterior join con la tabla de personal y obtener el nombre de los actores, esto resultó en una reducción del coste hasta 213.
Como propuesta alternativa para la tercera consulta se planteó también el desnormalizar la parte de la agregación teniendo en una sola tabla la información de actores y contenido para poder realizar la mayoría de la consulta sin joins, pero finalmente no se realizó.



Como conclusiones de la optimización podemos decir que tener los datos necesarios precalculados favorece mucho a la optimización y, la desnormalización de tablas para tener joins precalculados puede suponer ventajas en tiempo con la pega de que tienes que mantener consistencia en los datos que en algunos casos te va a suponer un coste mayor al beneficio obtenido.

3.2 - Triggers

El primer trigger verifica la restricción de exclusividad que existe entre películas y series, es decir si se intenta insertar una tupla en películas que ya está en series esta será rechazada.

- Trigger 1:

```
CREATE OR REPLACE TRIGGER EXC_PELIS
BEFORE INSERT ON peliculas
FOR EACH ROW
DECLARE
    flag NUMBER;
BEGIN
    SELECT COUNT(*) INTO flag FROM series WHERE id = :NEW.id;
    IF flag >= 1 THEN
        RAISE_APPLICATION_ERROR (-20000, 'La fila insertada ya pertenece a
        la categoría de series.');
    END IF;
END;
/
```

Este trigger aporta una restricción lógica a los datos, comprueba que la fecha de estreno de un contenido y su fecha de finalización en caso de ser una serie sean coherentes, en caso de no serlo se cancela la inserción en la tabla de series y se avisa al usuario de que tupla es la que está dando el problema para que así pueda decidir si borrar la tupla en contenido o corregir las fechas.

- Trigger 2:

```
CREATE OR REPLACE TRIGGER CLEAN_DATES
BEFORE INSERT ON series
FOR EACH ROW
DECLARE
    fechaEstr NUMBER;
BEGIN
    SELECT estreno INTO fechaEstr FROM contenido WHERE :NEW.id = id;
    IF :NEW.fin < fechaEstr THEN
        RAISE_APPLICATION_ERROR (-20002, 'La serie: ' || :NEW.id || ' tiene una
        fecha de finalización menor que la del estreno de la tabla contenido.
        Considera eliminar/corregir la inserción.');
    END IF;
END;
/
```

El tercer trigger aporta consistencia, comprobando que cada nuevo personaje que se introduzca en la tabla de actores deben de haber participado en algún contenido al menos una vez con el rol de actor o actriz.

- Trigger 3:

```
CREATE OR REPLACE TRIGGER INSERT_ACT
BEFORE INSERT ON actores
FOR EACH ROW
DECLARE
    flag NUMBER;
BEGIN
  SELECT COUNT(*) INTO flag
  FROM colabora
  WHERE id_personal = :NEW.id AND (rol = 'actor' or rol = 'actress');

  IF flag < 1 THEN
      RAISE_APPLICATION_ERROR (-20002, 'La persona insertada no es un actor.');
  END IF;
END;
/
```

Gestión del grupo

Reuniones

18/04 Reunión inicial, para organizarnos.

20/04 Verificar el modelo E/R en una tutoría.

21/04 Probar a poblar la base de datos entera. Comenzar la documentación. Plantear triggers.

24/04 Consultas 1 y 2

29/04 Consulta 3 y triggers

1/05 Diseño físico

1/05 Documentación

División del trabajo

El modelo E/R fue realizado por Héctor, puesto posteriormente en común con el resto del grupo y finalmente revisado en una tutoría. Tras esto, mientras Pablo comenzaba a realizar el modelo relacional y Héctor el fichero sql para la creación de tablas, Pizarro comenzó a pensar las primeras consultas. Una vez el relacional terminado, Pablo y Héctor compararon el trabajo realizado y procedieron con la normalización a la vez que se modificaban los archivos creados para su corrección. Una vez listo el modelo relacional, Pizarro y Héctor realizaron una serie de consultas sql exportando los datos en formato insert para su posterior población. Una vez poblada la base de datos, mientras Pizarro realizaba los primeros intentos con las consultas, Pablo comenzó a redactar la memoria y la presentación de manera que esta vez fuéramos mejor con los tiempos y Héctor realizó los triggers. Cercanos a la fecha de entrega, los tres acabamos las consultas y revisamos los árboles.Finalmente, Pablo y Héctor realizaron el diseño físico y terminaron de dejar listas la memoria y presentación.

Problemas de coordinación

Dado que esta vez todos los integrantes del grupo éramos conscientes de la carga de trabajo de la práctica así como de los problemas de coordinación de la práctica anterior, esta vez no hemos tenido tantos problemas para organizarnos y hemos repartido "mejor" el tiempo y trabajo.