BASES DE DATOS

2º CURSO - 2ºCUATRIMESTRE

GRUPO L1.01

PROYECTO 2

Carlos Solana Melero NIP: 872815@unizar.es

Daniel Simón Gayán Nip: 870984@unizar.es

Diego Mateo Lorente Nip: 873338@unizar.es

6 marzo de 2024

Parte 1: Creación de una base de datos:

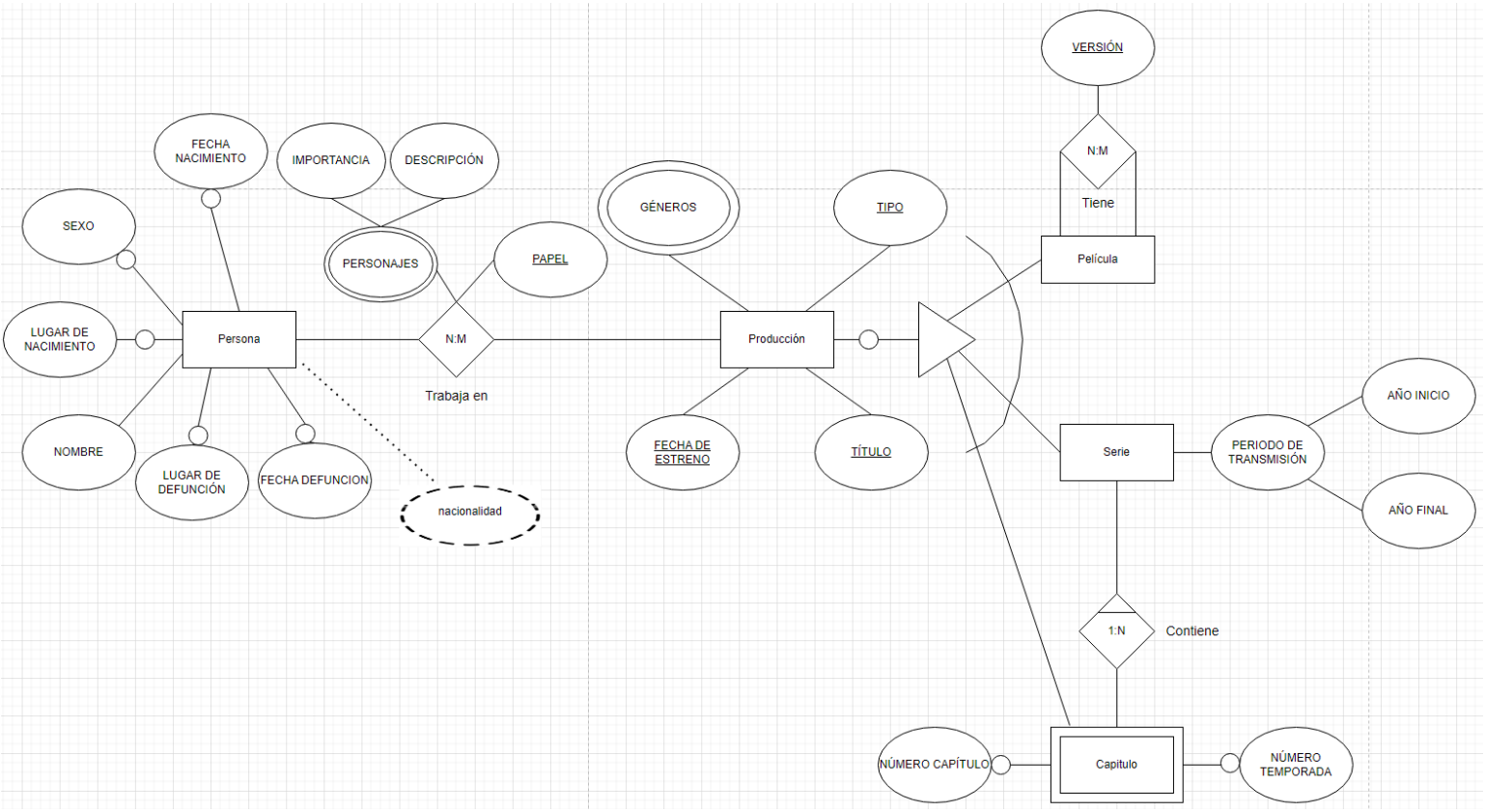
Modelo Entidad - Relación

Para poner sobre la mesa el problema que se nos plantea con esta base de datos, realizamos un esquema del modelo Entidad/Relación que plasma la realidad. Durante la realización de dicho esquema, nos hemos topado con numerosas cuestiones que daban lugar a pensar qué solución era la más óptima y adecuada para cada caso. Por supuesto, dichas cuestiones van a ser mencionadas en este apartado junto a las posibles soluciones y, finalmente, la opción seleccionada en cada caso. Comenzamos hablando de la entidad “Persona”, donde tenemos todos los atributos personales de cada una, teniendo los atributos “SEXO”, que en caso de no ser nulo sería ‘M’ o ‘F’, su nombre sería el único campo obligatorio ya que debido a como se nos proporcionan los datos solo hemos tenido en cuenta el lugar de nacimiento, aunque otros campos como fecha de nacimiento, lugar de defunción y fecha de defunción y los dejamos como opcionales ya que no los insertamos, también tenemos el campo derivado de nacionalidad que depende del lugar de nacimiento.

Por otro lado la entidad persona se relaciona con la entidad producción mediante la relación trabajo, esta relación es de N:M ya que en una producción trabajan 1 o más personas, lo mismo ocurre en sentido contrario, 1 persona trabaja en 1 o más producciones. En la propia relación hay atributos, estos son papel que a su vez es clave primaria para identificar esa relación, ya que la persona Carlos puede trabajar en Titanic como actor, y luego otra instancia de esa relación sería Carlos,la misma persona, trabaje como director en esa misma película si no fuese clave primaria no habría forma de diferenciar esas 2 instancias. Otro atributo es personajes que es multivaluado y solo existe si el papel es actor o actriz, es una restricción textual, si el papel es actor puede representar varios personajes en esa misma obra pero sigue siendo la misma instancia de la relación. Otra entidad es Producción tiene como atributos tipo, título y fecha estreno que son las claves primarias ya que consideramos que si una película se estrena el

mismo año con el mismo tipo y el mismo título es la misma. Otro atributo es el género que es multivaluado ya que una producción puede ser de comedia y terror al mismo tiempo, por ejemplo.

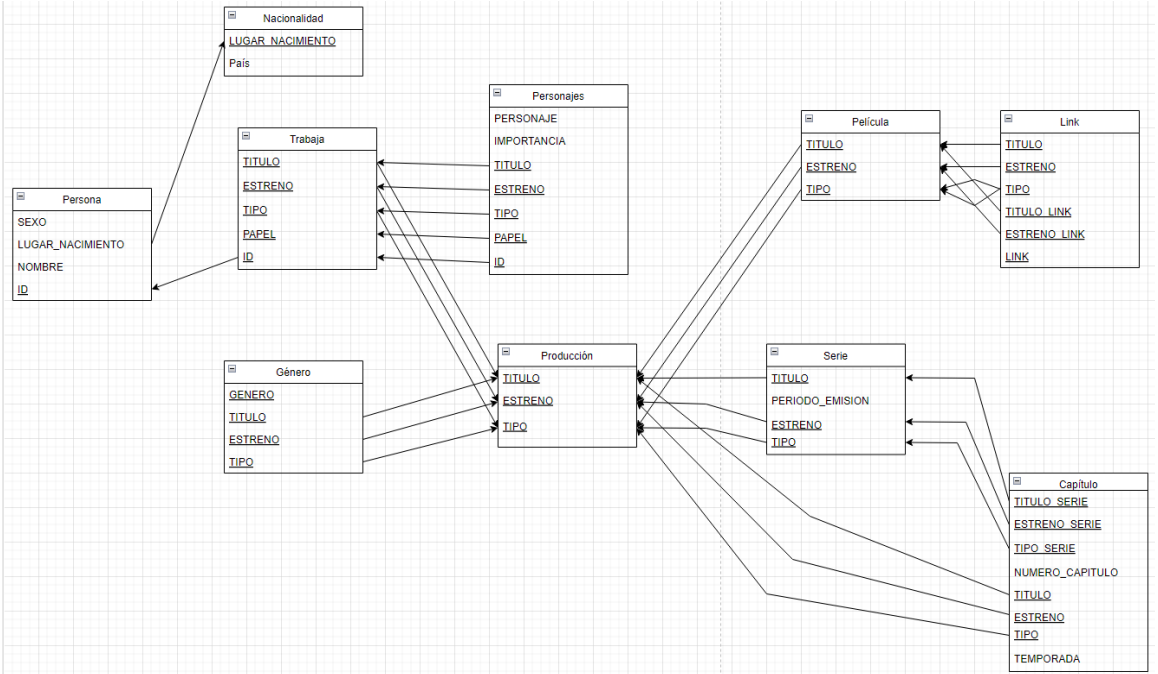
Producción actúa como superclase de las entidades Película,Serie y Capítulo, estas entidades heredan esos atributos de Producción y a su vez definen algunos propios. Tal y como está definida una producción tiene que ser si o si una película, una serie o un capítulo y además no puede ser más de 1 a la vez Serie tiene periodo de transmisión que es compuesto y Capítulo tiene numero de temporada que es opcional y número de capítulo que es clave primaria, además está relacionado con serie, 1 serie tiene N capítulos, la entidad capítulo es una entidad débil ya que un capítulo no existe sin una serie. Por otro lado película no tiene atributos extra pero sí una relación N:M con sí mismo ya que en esta base de datos hay películas relacionadas con otras películas(secuelas, precuelas…) y un atributo en la relación que es primario que nos dice precisamente que version es, secuela,precuela,remake…



Modelo relacional y normalización

Para poder crear las tablas y poblar hay que pasar el entidad relación al modelo relacional y pasarlo a la FNBC.

Inicialmente tenemos 5 entidades por lo tanto habrá 5 tablas, lo primero que hay que tener en cuenta es que tenemos 2 relaciones N:M así que necesitaremos descomponer esas relaciones, ya tendremos 2 tablas más. Por otro lado hay que formalizar el modelo, para ello hay que seguir modificando nuestro modelo, una tabla está en Primera Forma Normal si todos sus atributos contienen sólo valores únicos, desde nuestro modelo entidad relación aquí encontramos 2 problemas con los atributos personajes y géneros así que hemos tenido que descomponer las tablas producción añadiendo la tabla generos referenciando a producción y la tabla personajes referenciando a la relación de persona y producción. Luego para la segunda forma normal está en Segunda Forma Normal si está en 1NF y todos los atributos no clave dependen completamente de la clave primaria. Esto ocurre ya al pasar del entidad relación al relacional y la tercera forma normal si está en 2NF y todos los atributos no clave son mutuamente independientes y dependen solo de la clave primaria que también lo cumplía inicialmente, pero al tener en cuenta la nacionalidad hemos tenido que sacar a parte otra tabla extra ya que nacionalidad depende de un atributo no clave . La Forma Normal de Boyce-Codd se basa en que ninguno de nuestros atributos que forman parte de una clave dependen de un atributo que no es parte de esa clave, este y no nos ocurre



Sentencias SQL de creación de tablas

Ahora es momento de plasmar nuestro modelo relacional en las tablas para posteriormente poder poblar los datos. Para crear las tablas hay que seguir un orden para que las referencias no den problemas.

La primera tabla es producción que elegimos como primary key sus 3 campos y título lo definimos como varchar para optimizar el espacio. y realizamos un check para que el tipo solo sea una de esas 3 posibilidades

```
CREATE TABLE Produccion (
        TITULO VARCHAR(200) ,
        ESTRENO NUMBER,
        TIPO VARCHAR(32) CHECK (TIPO IN ('movie', 'tv series', 'episode')),
        PRIMARY KEY(TITULO, ESTRENO, TIPO)
);
```

La siguiente es género que es la tabla que hemos incluido para normalizar y que estuviese en primera forma normal, la primary key son los campos de la clave primaria de la tabla que referencia y el campo genero.

```
CREATE TABLE Genero (
        TITULO VARCHAR(200),
        ESTRENO NUMBER,
        GENERO VARCHAR(32),
        TIPO VARCHAR(32),
        PRIMARY KEY (TITULO,ESTRENO,TIPO, GENERO),
        FOREIGN KEY (TITULO,ESTRENO,TIPO) REFERENCES Produccion(TITULO,ESTRENO,TIPO)
);
```

Para película su primary key son los campos que hereda de producción que a su vez son claves ajenas apuntando a esa tabla, además usamos un check para verificar que el estreno no es posterior a este año ni anterior a la creación del cine.

```
CREATE TABLE Pelicula (
        TITULO VARCHAR(200),
        ESTRENO NUMBER CHECK (ESTRENO BETWEEN 1895 AND 2024),
        TIPO VARCHAR(32),
        PRIMARY KEY (TITULO,ESTRENO,TIPO),
        FOREIGN KEY (TITULO,ESTRENO,TIPO) REFERENCES Produccion(TITULO,ESTRENO, TIPO)
);
```

La tabla link se usa para las películas que referencian a otra película y sus campos son los propios y los campos que referencian a la película con la cual está relacionada

```
CREATE TABLE Link (
        TITULO VARCHAR(200),
        ESTRENO NUMBER,
        TITULO_LINK VARCHAR(200),
        ESTRENO_LINK NUMBER,
        LINK VARCHAR(32),
        TIPO VARCHAR(32),
        PRIMARY KEY (TITULO,ESTRENO,TIPO,LINK,TITULO_LINK,ESTRENO_LINK),
        FOREIGN KEY (TITULO,ESTRENO,TIPO) REFERENCES Pelicula(TITULO,ESTRENO,TIPO),
        FOREIGN KEY (TITULO_LINK,ESTRENO_LINK,TIPO) REFERENCES Pelicula(TITULO,ESTRENO,TIPO)
);
```

Series la hemos planteado igual que película que su primary key también es foreign key que apunta a producción

```
CREATE TABLE Serie (
        TITULO VARCHAR(200),
        ESTRENO NUMBER,
        PERIODO_EMISION VARCHAR(32),
        TIPO VARCHAR(32),
        PRIMARY KEY (TITULO, ESTRENO, TIPO),
        FOREIGN KEY (TITULO, ESTRENO, TIPO) REFERENCES Produccion(TITULO, ESTRENO,TIPO)
);
```

La tabla capítulo es lo mismo pero con más campos ya que algunos referencian a producción(los propios de capítulos) y los que referencian a la serie de los que son

```
CREATE TABLE Capitulo (
        TITULO_SERIE VARCHAR(200),
        TEMPORADA NUMBER(9),
        NUMERO_CAPITULO NUMBER(9),
        TITULO VARCHAR(200),
        ESTRENO NUMBER,
        TIPO VARCHAR(32),
        ESTRENO_SERIE NUMBER,
        TIPO_SERIE VARCHAR(32),
        PRIMARY KEY (TITULO, TITULO_SERIE, ESTRENO, TIPO, ESTRENO_SERIE, TIPO_SERIE),
        FOREIGN KEY (TITULO_SERIE, ESTRENO_SERIE, TIPO_SERIE ) REFERENCES Serie(TITULO,
        ESTRENO, TIPO),
        FOREIGN KEY (TITULO, ESTRENO,TIPO) REFERENCES Produccion(TITULO, ESTRENO,TIPO)
);
```

En persona hemos creado una ID artificial que la usamos como primary key ya que los datos eran muy ambiguos y decidimos usarla ya que no encontrábamos manera de identificarlos correctamente si muchos tenían campos nulos

```
CREATE TABLE Persona (
        SEXO VARCHAR(2) CHECK (SEXO IN ('f', 'm','')),
        LUGAR_NACIMIENTO VARCHAR(150),
        NOMBRE VARCHAR(100),
        ID VARCHAR(32),
        PRIMARY KEY (ID)
        FOREIGN KEY (LUGAR_NACIMIENTO) REFERENCES LugarNacimiento(LUGAR_NACIMIENTO)
);
```

Trabaja es la tabla resultante de la relación persona y producción usamos como primary key las primary keys de persona combinadas con las de producción

```
CREATE TABLE Trabaja (
        PAPEL VARCHAR(32),
        ID VARCHAR(32),
        ESTRENO NUMBER,
        TITULO VARCHAR(200),
        TIPO VARCHAR(32),
        PRIMARY KEY (PAPEL,ID,ESTRENO,TITULO,TIPO),
        FOREIGN KEY (TITULO,ESTRENO,TIPO) REFERENCES Produccion(TITULO,ESTRENO,TIPO),
        FOREIGN KEY (ID) REFERENCES Persona(ID)
);
```

Por último la tabla InfoPersonajes representan los personajes que interpretan los actores, como primary key cogemos las mismas que en trabaja pero combinada con el atributo personaje

```
CREATE TABLE InfoPersonaje (
        PERSONAJE VARCHAR(100),
        IMPORTANCIA FLOAT,
        PAPEL VARCHAR(32),
        ID VARCHAR(32),
        ESTRENO NUMBER,
        TITULO VARCHAR(200),
        TIPO VARCHAR(32),
        PRIMARY KEY (PAPEL,ID,ESTRENO,TITULO,TIPO, PERSONAJE),
        FOREIGN KEY (PAPEL,ID,ESTRENO,TITULO,TIPO) REFERENCES Trabaja
        (PAPEL,ID,ESTRENO,TITULO,TIPO)
);
```

Esta última tabla es para cumplir la tercera forma normal ya que la nacionalidad depende de lugar de nacimiento y no de la clave primaria.

```
CREATE TABLE LugarNacimiento(
        LUGAR_NACIMIENTO VARCHAR(150) PRIMARY KEY,
        NACIONALIDAD VARCHAR(100)
);
```

Parte 2: Introducción de datos y ejecución de consultas

Poblando la base de datos

Al igual que la anterior base de datos, hemos poblado con sentencias sql a base de inserts en las tablas que ya hemos creado, algunas eran muy sencillas, pero para otras hemos tenido que crear views auxiliares para ir insertando poco a poco.

```
INSERT INTO Produccion SELECT DISTINCT title, production_year, kind
        FROM datosdb.datospeliculas
        WHERE title IS NOT NULL AND kind IS NOT NULL AND production_year IS NOT NULL;

INSERT INTO Genero SELECT DISTINCT title, production_year, keyword, kind
        FROM datosdb.datospeliculas
        WHERE title IS NOT NULL AND production_year IS NOT NULL AND keyword IS NOT NULL AND kind IS
        NOT NULL;

INSERT INTO Serie SELECT DISTINCT title, production_year, series_years, kind
        FROM datosdb.datospeliculas
        WHERE title IS NOT NULL AND production_year IS NOT NULL AND kind IS NOT NULL AND kind = 'tv
        series';

        INSERT INTO Capitulo SELECT DISTINCT serie_title,season_nr,episode_nr,title,production_year,kind,
        serie_prod_year, 'tv series'
        FROM datosdb.datospeliculas
        WHERE title IS NOT NULL AND serie_title IS NOT NULL AND production_year IS NOT NULL AND kind IS
        NOT NULL AND serie_prod_year IS NOT NULL AND kind = 'episode';

INSERT INTO Pelicula SELECT DISTINCT TITULO, ESTRENO, TIPO
        FROM Produccion
        WHERE TITULO IS NOT NULL AND ESTRENO IS NOT NULL AND TIPO IS NOT NULL AND TIPO =
        'movie';

INSERT INTO Link SELECT DISTINCT title, production_year, titlelink, productionyearlink, link, kind
        FROM datosdb.datospeliculas dp
        WHERE dp.title IS NOT NULL
        AND dp.production_year IS NOT NULL
        AND dp.kind = 'movie'
        AND dp.productionyearlink IS NOT NULL
        AND dp.titlelink IS NOT NULL
        AND dp.link IS NOT NULL
        AND (dp.titlelink,dp.productionyearlink, 'movie') IN (SELECT TITULO,ESTRENO,TIPO FROM Pelicula);

INSERT INTO LugarNacimiento(LUGAR_NACIMIENTO) SELECT DISTINCT person_info
        FROM datosDB.datospeliculas
        WHERE info_context = 'birth notes';

CREATE SEQUENCE sec START WITH 1 INCREMENT BY 1;

CREATE VIEW PeliculaInfo AS
        SELECT DISTINCT
        name,gender,role,title,production_year,kind
```

```
        FROM datosDB.datospeliculas
        WHERE name IS NOT NULL;

CREATE VIEW PersonaInfo AS
        SELECT DISTINCT
        name,gender,person_info,role,title,production_year,kind,info_context
        FROM datosDB.datospeliculas
        WHERE info_context = 'birth notes';

INSERT INTO Persona
        SELECT
        gender, person_info, name, sec.NEXTVAL
        FROM (
        SELECT DISTINCT
        p.name, p.gender, pn.person_info
        FROM PeliculaInfo p
        LEFT JOIN PersonaInfo pn ON (p.name = pn.name AND p.gender = pn.gender AND p.role = pn.role AND
        p.title = pn.title AND p.production_year=pn.production_year AND p.kind=pn.kind)
);


DROP SEQUENCE sec;
DROP VIEW PeliculaInfo;
DROP VIEW PersonaInfo;

INSERT INTO Trabaja
        SELECT
        role,
        ID,
         production_year,
        title,
         kind
FROM (
        SELECT DISTINCT
        dp.role,
         p.ID,
         dp.production_year,
        dp.title,
        dp.kind
 FROM datosDB.datospeliculas dp
        LEFT JOIN   PERSONA p ON ((p.NOMBRE = dp.name) AND (p.SEXO = dp.gender) AND
        ((p.LUGAR_NACIMIENTO = dp.person_info )OR (p.LUGAR_NACIMIENTO IS NULL AND dp.person_info
        IS NULL)))
         WHERE p.NOMBRE IS NOT NULL
);


INSERT INTO InfoPersonaje
SELECT
         role_name,
```

```
        nr_order,
        role,
        ID,
         production_year,
        title,
         kind
FROM (
         SELECT DISTINCT
        dp.role_name,
         dp.nr_order,
         dp.role,
         p.ID,
        dp.production_year,
        dp.title,
        dp.kind
        FROM datosDB.datospeliculas dp
        LEFT JOIN  PERSONA p ON ((p.NOMBRE = dp.name) AND (p.SEXO = dp.gender) AND
        ((p.LUGAR_NACIMIENTO = dp.person_info ) OR (p.LUGAR_NACIMIENTO IS NULL AND dp.person_info
        IS NULL)))
        WHERE p.NOMBRE IS NOT NULL AND (dp.role='actor' or dp.role='actress') AND dp.role_name IS NOT
        NULL
);
```

Como hemos mencionado antes algunas tablas como producción, género o serie bastaba con hacer un select distinct de los datos y poniendo alguna condición, sin embargo las tablas persona, trabaja e infoPersonajes. Para la clave primaria ID de persona hemos utilizado una secuencia para asignarlo, hemos usado dos vistas intermedias, la primera coge todas las instancias de personas sin importar el lugar de nacimiento que es nuestro campo diferencial para las personas, después la segunda vista miramos que info context sea birth notes para saber el lugar de nacimiento, después hacemos un left join para que en la view inicial que es la que más datos tiene, se añada la columna person_info si coinciden todos los campos que ponemos en el where. Después como ya no usamos más esas views las borramos al igual que la secuencia para optimizar el almacenamiento. Para la tabla trabaja la única complicación es que el id no está en los datos así que hay que cogerlo de la tabla persona, y luego hacemos un left join con condición que el nombre sea el mismo, mismo género y mismo lugar de nacimiento o nulo, que es lo que consideraríamos personas iguales. Por último personajes es prácticamente igual que en trabaja pero con otros campos en los que PAPEL tiene que ser ACTOR o ACTRIZ ya que si no esa tabla no existe, es una restricción textual.

Consultas

Al igual que en la primera base de datos, hemos seguido utilizando las vistas para crear tablas intermedias que cumplan las condiciones que buscamos y poco a poco ir acercándonos a la solución esperada. Así, vamos a explicar en cada una de las 3 consultas todas las vistas creadas y la utilidad de las dichas para llegar a la solución correspondiente.

Consulta 1: Directores para los cuales la última película en la que han participado ha sido como actor/actriz:

Para esta primera consulta, hemos decidido crear una primera vista en la cual se guarden todos los trabajos de directores en obra, es decir, todos los directores de todas y cada una de las películas. Esto nos sirve para saber todas las obras que ha realizado cada director y el año de estreno de todas y cada una de ellas. La sentencia en SQL es la siguiente:

```
CREATE VIEW directores
AS SELECT p.NOMBRE, t.ID, t.TITULO, t.ESTRENO
FROM Trabaja t
JOIN Persona p ON t.ID = p.ID
WHERE t.PAPEL = 'director' AND t.TIPO = 'movie';
```

A continuación, creamos una vista que guarde de cada director, identificado por ID, su obra más reciente. De esta forma, mediante esta vista y una que crearemos más adelante, podemos comparar los años de estreno de ambas vistas y sacar la solución de la consulta. La sentencia SQL queda así:

```
CREATE VIEW directoresUltimaObra
AS SELECT NOMBRE, ID, TITULO, ESTRENO
FROM directores d
WHERE ESTRENO IN (SELECT MAX(ESTRENO) FROM directores t WHERE t.ID = d.ID);
```

Al igual que con los directores, creamos una vista que contenga todos los trabajos de actores/actrices en películas, cuyo uso es el mismo que en los directores. Después, también al igual que con los directores, creamos otra vista que guarde para cada actor o actriz su última obra, es decir, aquella cuyo año de estreno sea mayor. Así, las 2 vistas que hemos mencionado quedan así:

```
CREATE VIEW actores
AS SELECT p.NOMBRE, t.ID, t.TITULO, t.ESTRENO
FROM Trabaja t
JOIN Persona p ON t.ID = p.ID
WHERE (t.PAPEL = 'actor' OR t.PAPEL = 'actress') AND t.TIPO = 'movie';

CREATE VIEW actoresUltimaObra
AS SELECT a.NOMBRE, a.ID, a.TITULO, a.ESTRENO
FROM actores a
WHERE ESTRENO IN (SELECT MAX(ESTRENO) FROM actores t WHERE t.ID = a.ID);
```

Una vez tenemos todas las vistas anteriores, tenemos todo para sacar la solución. El objetivo será el siguiente: Si un director y actor tienen el mismo identificador, es decir, son la misma persona, y la última obra del actor salió en el mismo año o después que la última obra que hizo como director, cumple con las restricciones que nos plantea la consulta, por lo que se añade a la selección. Hemos considerado que si la última obra de una persona tanto de director como de actor es el mismo, hemos considerado que la última obra sería como actor. Así, la selección final de la consulta es la siguiente:

```
SELECT DISTINCT a.ID, a.NOMBRE
FROM actoresUltimaObra a, directoresUltimaObra d
WHERE a.ID = d.ID AND a.ESTRENO>=d.ESTRENO;
```

Tras ejecutar la consulta en cuestión, la solución resultante es la siguiente:

ID NOMBRE

------ ------------------------------

984 de Turckheim, Charlotte

2551 Belen, Ana

5366 Ruiz de Austri, Maite

7500 Lopez, Albert

8375 Conesa, Carmen

.

.

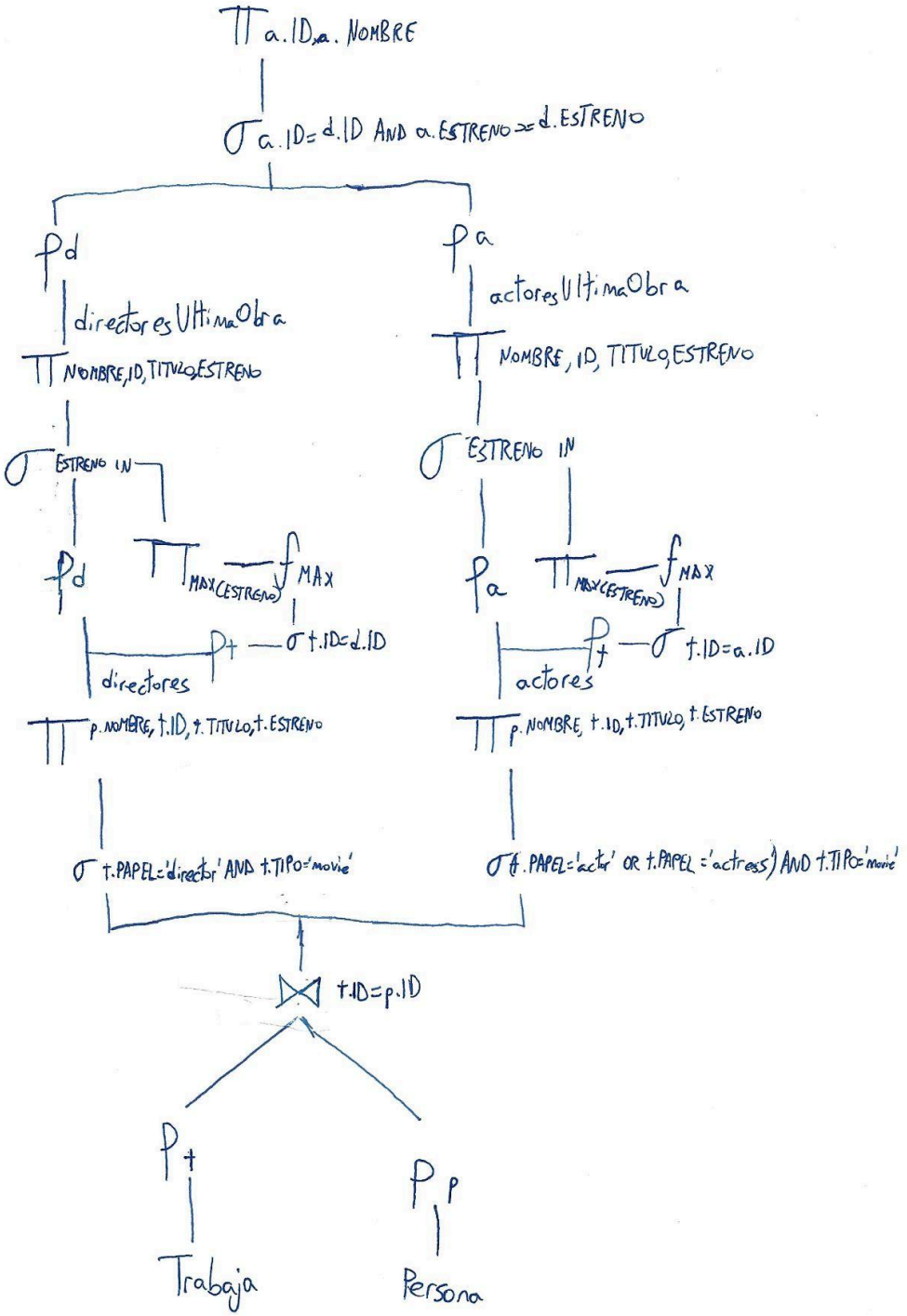
.

26152 Tortosa, Silvia

32992 Lee, Spike

30178 Walthard, Hans R.

278 filas seleccionadas.


π a.ID, a.NOMBRE
σ a.ID = d.ID AND a.ESTRENO = d.ESTRENO
ρd
directoresUltimaObra
π NOMBRE, ID, TITULO, ESTRENO
σ ESTRENO IN
ρd
π MAX(ESTRENO)
fMAX
ρt
σ t.ID = d.ID
directores
π p.NOMBRE, t.ID, t.TITULO, t.ESTRENO
σ t.PAPEL = 'director' AND t.TIPO = 'movie'
ρa
actoresUltimaObra
π NOMBRE, ID, TITULO, ESTRENO
σ ESTRENO IN
ρa
π MAX(ESTRENO)
fMAX
ρt
σ t.ID = a.ID
actores
π p.NOMBRE, t.ID, t.TITULO, t.ESTRENO
σ (t.PAPEL = 'actor' OR t.PAPEL = 'actress') AND t.TIPO = 'movie'
⋈ t.ID = p.ID
ρt
Trabaja
ρp
Persona

Consulta 2: Obtener la saga de películas más larga (en número de películas), listando los títulos de las películas que la componen (incluyendo precuelas y secuelas):

Para la segunda consulta, hemos pensado que la forma más sencilla de contar el número de obras de cada saga es coger la primera película de cada saga, ya que esta es la que más secuelas tiene. Por tanto, en la primera vista vamos a incluir todas las secuelas de películas existentes en la base de datos. La vista queda así:

```
CREATE VIEW secuelas
AS SELECT TITULO_LINK, ESTRENO_LINK, TITULO, ESTRENO
FROM Link
WHERE LINK = 'follows' AND ESTRENO > ESTRENO_LINK;
```

Ahora que tenemos la vista, si contamos todas las secuelas de cada película, sólo habrá una película por saga que tenga el máximo número de secuelas pertenecientes a dicha saga, por lo que no va a haber ningún duplicado en la solución. A continuación, creamos otra vista en la que a cada película le corresponda el número de secuelas que tiene dicha película. Esto será clave para sacar el máximo número de secuelas y, por tanto, la saga más larga de películas. La vista queda de esta forma:

```
CREATE VIEW numSecuelas
AS SELECT TITULO_LINK, ESTRENO_LINK, COUNT(*) AS NUM_SECUELAS
FROM secuelas
GROUP BY TITULO_LINK, ESTRENO_LINK;
```

La última vista que vamos a hacer guarda la película que más secuelas tiene, es decir, la primera película de la saga de películas más larga, que es la solución que buscamos. Dicha vista queda así:

```
CREATE VIEW maxSecuelas
AS SELECT TITULO_LINK AS TITULO, n.ESTRENO_LINK AS ESTRENO, n.NUM_SECUELAS AS NUMERO_DE_SECUELAS
FROM numSecuelas n
WHERE n.NUM_SECUELAS IN (SELECT MAX(NUM_SECUELAS) FROM numSecuelas s);
```

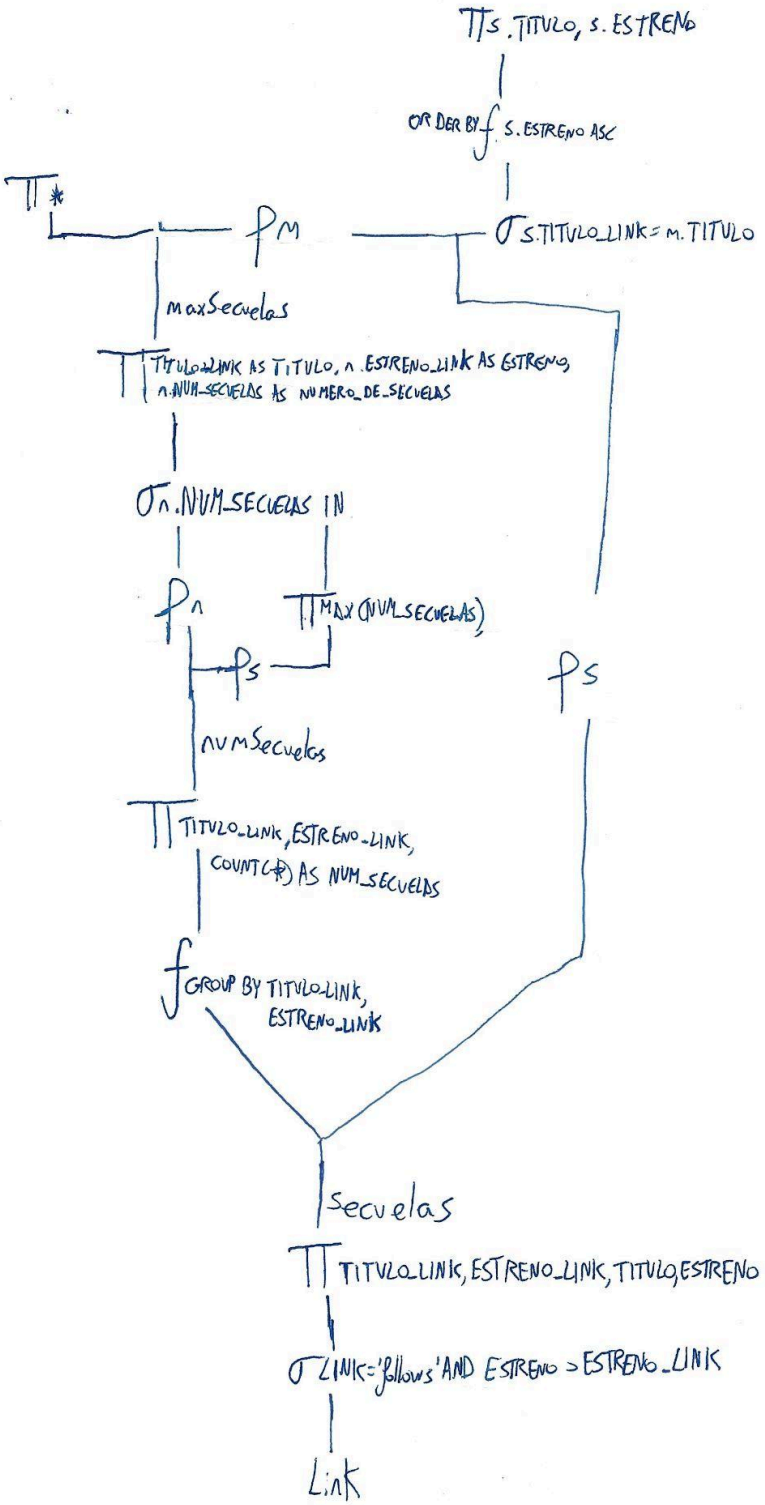
Finalmente, hacemos una selección de la única película que se guarda en la tabla maxSecuelas y después hacemos una selección de aquellas secuelas cuyo título al que referencian es el único elemento de la vista maxSecuelas. Es decir, primero mostramos la primera película de la saga con su número de secuelas y, después, mostramos todas las secuelas de dicha película. Dichas secuelas las mostramos en orden ascendente para que salgan en orden de salida de arriba a abajo. La selección queda así:

```
SELECT * FROM maxSecuelas;


SELECT DISTINCT s.TITULO, s.ESTRENO
FROM maxSecuelas m, secuelas s
WHERE s.TITULO_LINK = m.TITULO
ORDER BY s.ESTRENO ASC;
```

Tras ejecutar la consulta, la solución es la siguiente: La saga más larga películas es la siguiente:

- **La maldición de la bestia (1975)**
- **El retorno del Hombre-Lobo (1981)**
- **La bestia y la espada mágica (1983)**
- **El aullido del diablo (1987)**
- **Licántropo: El asesino de la luna llena (1996)**


Π s.TITULO, s.ESTRENO
ORDER BY f s.ESTRENO ASC
Π *
ρ m
σ s.TITULO_LINK = m.TITULO
maxSecuelas
Π TITULO_LINK AS TITULO, n.ESTRENO_LINK AS ESTRENO, n.NUM_SECUELAS AS NUMERO_DE_SECUELAS
σ n.NUM_SECUELAS IN
ρ n
Π MAX (NUM_SECUELAS)
ρ s
ρ s
numSecuelas
Π TITULO_LINK, ESTRENO_LINK, COUNT(*) AS NUM_SECUELAS
f GROUP BY TITULO_LINK, ESTRENO_LINK
Secuelas
Π TITULO_LINK, ESTRENO_LINK, TITULO, ESTRENO
σ LINK='follows' AND ESTRENO > ESTRENO_LINK
Link

Consulta 3: Listar las parejas actor-actriz de distinta nacionalidad que siempre han trabajado juntos en películas estrenadas entre 1980 y 2010, indicando en cuantas ocasiones lo han hecho (listar en orden decreciente según este dato), y el nombre y nacionalidad de cada uno:

Para la tercera y última consulta, primero guardaremos en una vista con todos los trabajos en los que participan actores y en otra vista lo mismo con las actrices. Como la consulta incluye una restricción temporal, la aplicamos en otras 2 vistas semejantes a las 2 anteriores, con la diferencia de que ahora las obras son desde 1980 hasta 2010, además de que añadimos un nuevo campo a la vista, en concreto la nacionalidad de cada actor/actriz en cada caso. Dicho nuevo campo lo definimos en los triggers, que serán explicados más adelante. Las 4 vistas quedan así:

```
CREATE VIEW actores
AS SELECT p.NOMBRE AS NOM_ACTOR, t.ID as ID_actor, t.TITULO, t.ESTRENO, p.LUGAR_NACIMIENTO
FROM Trabaja t
JOIN Persona p ON t.ID = p.ID
WHERE t.PAPEL = 'actor' AND t.TIPO = 'movie';

CREATE VIEW actoresNacionalidad
AS SELECT t.NOM_ACTOR, t.ID_actor, t.TITULO, t.ESTRENO, p.NACIONALIDAD AS NACION_ACTOR
FROM actores t
JOIN LugarNacimiento p ON t.LUGAR_NACIMIENTO = p.LUGAR_NACIMIENTO
WHERE ESTRENO >= 1980 AND ESTRENO <=2010;

CREATE VIEW actrices
AS SELECT p.NOMBRE AS NOM_ACTRIZ, t.ID AS ID_ACTRIZ, t.TITULO, t.ESTRENO, p.LUGAR_NACIMIENTO
FROM Trabaja t
JOIN Persona p ON t.ID = p.ID
WHERE t.PAPEL = 'actress' AND t.TIPO = 'movie';


CREATE VIEW actricesNacionalidad
AS SELECT t.NOM_ACTRIZ, t.ID_ACTRIZ, t.TITULO, t.ESTRENO, p.NACIONALIDAD AS NACION_ACTRIZ
FROM actrices t
JOIN LugarNacimiento p ON t.LUGAR_NACIMIENTO = p.LUGAR_NACIMIENTO
WHERE ESTRENO >= 1980 AND ESTRENO <=2010;
```

Antes de explicar la próxima vista, explicamos la lógica que hemos seguido para llegar a la solución: Primero, contamos el número de obras que ha hecho cada actor y cada actriz. Después, en otra vista cogeremos aquellas parejas que han coincidido en al menos una obra. Luego, contamos el número de coincidencias que han tenido

como pareja y, si dicho número es igual al número de obras de cada uno, quiere decir que siempre han trabajado juntos. Habiendo entendido esta lógica, es relativamente sencillo crear las vistas propuestas para llegar a la solución solicitada. Comenzamos creando las vistas que contienen el número de obras de cada actor y de cada actriz, quedando así:

```
CREATE VIEW contadorActores
AS SELECT NOM_ACTOR, ID_actor, NACION_ACTOR, COUNT(*) AS NUM_OBRAS
FROM actoresNacionalidad
GROUP BY NOM_ACTOR, ID_actor, NACION_ACTOR
ORDER BY NUM_OBRAS asc;

CREATE VIEW contadorActrices
AS SELECT NOM_ACTRIZ, ID_actriz, NACION_ACTRIZ, COUNT(*) AS NUM_OBRAS
FROM actricesNacionalidad
GROUP BY NOM_ACTRIZ, ID_actriz, NACION_ACTRIZ
ORDER BY NUM_OBRAS desc;
```

Después, creamos una vista que guarde aquellas parejas actor-actriz que hayan trabajado en el mismo número de obras, quedando así:

```
CREATE VIEW mismoNumObras
AS SELECT a.NOM_ACTOR, a.ID_actor, a.NACION_ACTOR, b.NOM_ACTRIZ, b.ID_actriz, b.NACION_ACTRIZ,
a.NUM_OBRAS
FROM contadorActores a, contadorActrices b
WHERE a.NUM_OBRAS = b.NUM_OBRAS
ORDER BY a.NUM_OBRAS asc;
```

A continuación, creamos 2 vistas más, en las que en una de ellas guardaremos aquellas parejas actor-actriz que hayan coincidido en una película, mostrando el título de la dicha en cada caso. En la otra vista, contamos el número de veces que han coincidido una pareja actor-actriz. Dichas 2 vistas quedan de esta forma:

```
CREATE VIEW coincidenciasActores
AS SELECT a.NOM_ACTOR, a.ID_actor, a.NACION_ACTOR , b.NOM_ACTRIZ, b.ID_actriz, b.NACION_ACTRIZ,
a.TITULO, a.ESTRENO
FROM actoresNacionalidad a, actricesNacionalidad b
WHERE b.TITULO = a.TITULO;

CREATE VIEW numCoincidenciasActores
AS SELECT NOM_ACTOR, ID_ACTOR, NACION_ACTOR, NOM_ACTRIZ, ID_ACTRIZ, NACION_ACTRIZ,
COUNT(*) AS NUM_COINCIDENCIAS
FROM coincidenciasActores
GROUP BY NOM_ACTOR, ID_ACTOR, NACION_ACTOR,NOM_ACTRIZ, ID_ACTRIZ, NACION_ACTRIZ;
```

Finalmente, como se ha explicado anteriormente, si el número de coincidencias de una pareja es igual al número de obras del actor, de forma implícita, también será igual al número de de obra de la actriz, es decir, siempre han trabajado juntos. También, además de comparar que los identificadores de la pareja coincidan, también comprobamos que el actor sea de distinta nacionalidad que la actriz. Finalmente, seleccionamos todas las filas de dicha vista para que se muestre por pantalla la solución esperada. El código es el siguiente:

```
CREATE VIEW parejasActores
AS SELECT DISTINCT n.NOM_ACTOR, n.ID_ACTOR, n.NACION_ACTOR, n.NOM_ACTRIZ, n.ID_ACTRIZ, n.NACION_ACTRIZ, m.NUM_OBRAS
FROM mismoNumObras m, numCoincidenciasActores n
WHERE m.NUM_OBRAS = n.NUM_COINCIDENCIAS AND n.ID_ACTOR = m.ID_ACTOR AND n.ID_ACTRIZ = m.ID_ACTRIZ AND n.NACION_ACTOR != n.NACION_ACTRIZ
ORDER BY m.NUM_OBRAS asc;

SELECT * FROM parejasActores;
```

Tras ejecutar la consulta, la solución es la siguiente:

NOM_ACTOR ID_ACT NACION_ACTOR NOM_ACTRIZ
------------------------- ------ --------------- -------------------------
ID_ACT NACION_ACTRIZ NUM_OBRAS
------ --------------- ----------
Mitchum, Christopher 26564 USA Lahaie, Brigitte
5285 France 2

Jouvet, Maurice 5591 France Beltran, Nelly
26895 Argentina 2

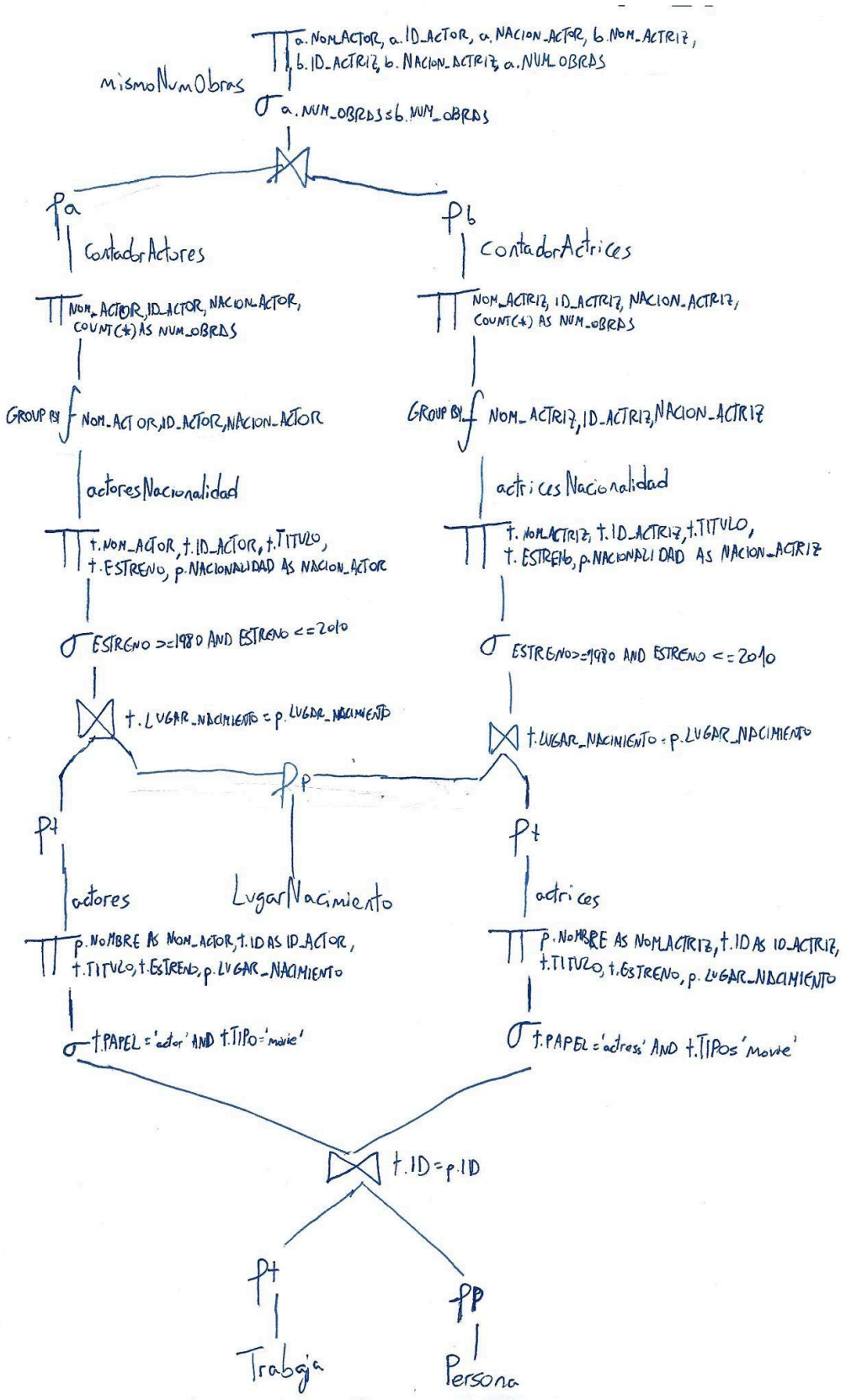
Russell, William 10030 UK Delora, Jennifer
36294 USA 1

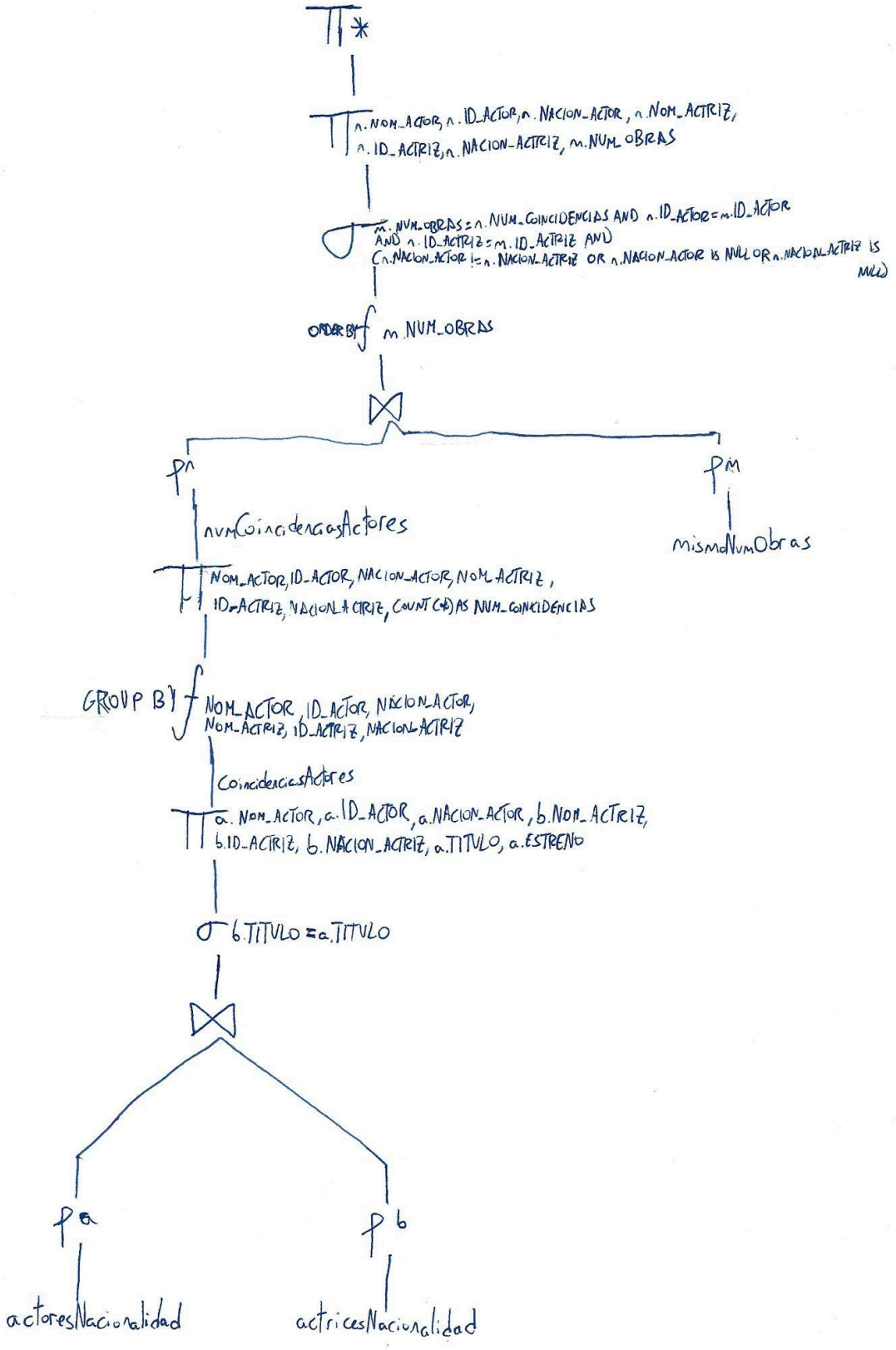
.
.
.

Bory, Jean-Marc 9857 Switzerland Vazzoler, Elsa
34843 Italy 1

Lopez Cobos, Jesus 9929 Spain Migenes, Julia
1597 USA 1

1832 filas seleccionadas.


mismoNumObras
π a.NOM_ACTOR, a.ID_ACTOR, a.NACION_ACTOR, b.NOM_ACTRIZ, b.ID_ACTRIZ, b.NACION_ACTRIZ, a.NUM_OBRAS
σ a.NUM_OBRAS=b.NUM_OBRAS
ρa
ContadorActores
π NOM_ACTOR, ID_ACTOR, NACION_ACTOR, COUNT(*) AS NUM_OBRAS
GROUP BY NOM_ACTOR, ID_ACTOR, NACION_ACTOR
actoresNacionalidad
π t.NOM_ACTOR, t.ID_ACTOR, t.TITULO, t.ESTRENO, p.NACIONALIDAD AS NACION_ACTOR
σ ESTRENO >=1980 AND ESTRENO <=2010
t.LUGAR_NACIMIENTO = p.LUGAR_NACIMIENTO
ρt
actores
π p.NOMBRE AS NOM_ACTOR, t.ID AS ID_ACTOR, t.TITULO, t.ESTRENO, p.LUGAR_NACIMIENTO
σ t.PAPEL='actor' AND t.TIPO='movie'
ρp
LugarNacimiento
ρb
ContadorActrices
π NOM_ACTRIZ, ID_ACTRIZ, NACION_ACTRIZ, COUNT(*) AS NUM_OBRAS
GROUP BY NOM_ACTRIZ, ID_ACTRIZ, NACION_ACTRIZ
actricesNacionalidad
π t.NOMACTRIZ, t.ID_ACTRIZ, t.TITULO, t.ESTRENO, p.NACIONALIDAD AS NACION_ACTRIZ
σ ESTRENO>=1980 AND ESTRENO <=2010
t.LUGAR_NACIMIENTO = p.LUGAR_NACIMIENTO
ρt
actrices
π p.NOMBRE AS NOMACTRIZ, t.ID AS ID_ACTRIZ, t.TITULO, t.ESTRENO, p.LUGAR_NACIMIENTO
σ t.PAPEL='actress' AND t.TIPOS'movie'
t.ID=p.ID
ρt
Trabaja
ρp
Persona


Π *
Π n.NOM_ACTOR, n.ID_ACTOR, n.NACION_ACTOR, n.NOM_ACTRIZ, n.ID_ACTRIZ, n.NACION_ACTRIZ, m.NUM_OBRAS
⋈ m.NUM_OBRAS = n.NUM_COINCIDENCIAS AND n.ID_ACTOR = m.ID_ACTOR AND n.ID_ACTRIZ = m.ID_ACTRIZ AND (n.NACION_ACTOR != n.NACION_ACTRIZ OR n.NACION_ACTOR IS NULL OR n.NACION_ACTRIZ IS NULL)
ORDER BY f m.NUM_OBRAS
⋈
ρn
ρm
numCoincidenciasActores
mismoNumObras
Π NOM_ACTOR, ID_ACTOR, NACION_ACTOR, NOM_ACTRIZ, ID_ACTRIZ, NACION_ACTRIZ, COUNT(*) AS NUM_COINCIDENCIAS
GROUP BY f NOM_ACTOR, ID_ACTOR, NACION_ACTOR, NOM_ACTRIZ, ID_ACTRIZ, NACION_ACTRIZ
coincidenciasActores
Π a.NOM_ACTOR, a.ID_ACTOR, a.NACION_ACTOR, b.NOM_ACTRIZ, b.ID_ACTRIZ, b.NACION_ACTRIZ, a.TITULO, a.ESTRENO
σ b.TITULO = a.TITULO
⋈
ρa
ρb
actoresNacionalidad
actricesNacionalidad

Parte 3: Diseño Físico

Análisis rendimiento

Consulta 1 :

Para garantizar que la consulta optimizada funcione de manera más eficiente, hemos creado varios índices específicos y una vista materializada. Estos elementos están diseñados para mejorar el rendimiento de las operaciones de JOIN y las condiciones WHERE que son frecuentes en las consultas.

Hemos creado una vista materializada con todos los directores que, aunque solo se use en una consulta creemos que puede resultar relevante para el futuro.

```
CREATE MATERIALIZED VIEW directorM
REFRESH ON COMMIT
AS SELECT p.NOMBRE, t.ID, t.TITULO, t.ESTRENO
FROM Trabaja t
JOIN Persona p ON t.ID = p.ID
WHERE t.PAPEL = 'director' AND t.TIPO = 'movie
```

En un principio nuestro objetivo era crear una vista materializada con todos los actores de películas que era utilizable en varias consultas y muy útil pero no nos dejaba por restricciones de espacio

```
--CREATE MATERIALIZED VIEW actoresM
--REFRESH ON COMMIT
--AS
--  SELECT
--      p.NOMBRE AS NOM_ACTOR,
--      t.ID as ID_ACTOR,
--      t.TITULO,
--      t.ESTRENO,
---      ln.NACIONALIDAD AS NACION_ACTOR,
--      t.PAPEL
--  FROM Trabaja t
 --  JOIN Persona p ON t.ID = p.ID
 --  JOIN LugarNacimiento ln ON p.LUGAR_NACIMIENTO = ln.LUGAR_NACIMIENTO
--  WHERE t.TIPO = 'movie' AND (t.PAPEL = 'actress' OR t.PAPEL = 'actor') AND t.TIPO = 'movie'
';
```

Tras eso intentamos particionar la tabla de trabaja pero nos salió también un error de espacio

```
CREATE TABLE Trabaja (
   PAPEL VARCHAR(32),
   ID VARCHAR(32),
   ESTRENO NUMBER,
   TITULO VARCHAR(200),
   TIPO VARCHAR(32),
   PRIMARY KEY (PAPEL, ID, ESTRENO, TITULO, TIPO)
)
PARTITION BY LIST (PAPEL)
(
   PARTITION p_actores VALUES ('actor', 'actress'),
   PARTITION p_otros VALUES (DEFAULT)
);
```

Al final la mejor solución que encontramos para agilizar las consultas en actores fue un índice bitmap sobre el tipo, ya que solo tiene 11 valores como máximo

```
CREATE BITMAP INDEX idx_tipo_trabaja ON Trabaja(TIPO);
```

Otros índices que ayudan a la consulta son:

```
CREATE INDEX idx_directorM_ID_ESTRENO ON directorM (ID, ESTRENO);
```

Este índice facilita las operaciones de JOIN y las búsquedas de las últimas obras de directores por su ID y ESTRENO, directamente relevante para la consulta.

```
CREATE BITMAP INDEX idx_papel_trabaja ON Trabaja(PAPEL);
CREATE BITMAP INDEX idx_tipo_trabaja ON Trabaja(TIPO);
```

Estos dos índices aceleran todos lo selects de actor o actriz (papel) para películas (tipo) lo que permite usar un full range scan que es más rápido.

Como resultado se ha conseguido reducir el coste de la consulta a la mitad gracias a la vista materializada y al índice creado sobre ella, así como a los bitmaps. El primer select se reduce en coste debido a que ya está creado y ordenado

- Costo Total del Plan No Optimizado: 242
- Costo Total del Plan Optimizado: 126

Consulta 2:

Para esta consulta hemos decidido crear una vista materializada de las secuelas, ya que hemos considerado que puede ser relevante ya que probablemente sea el tipo de link más importante y afecta directamente a la consulta

```
CREATE MATERIALIZED VIEW secuelasM

REFRESH ON COMMIT

AS SELECT TITULO_LINK, ESTRENO_LINK, TITULO, ESTRENO

FROM Link

WHERE LINK = 'follows' AND ESTRENO > ESTRENO_LINK;
```

Tras esto hemos creado un índice para ordenar en la tabla, al ser pequeña no ocupa mucho y ayuda mucho a la consulta.

```
CREATE INDEX idx_directorM_ID_ESTRENO ON directorM (ID, ESTRENO);
```

Ahora que hemos aplicado las mejoras vamos a comparar el coste en algunas operaciones principales

Plan Sin Optimizar

Costo Total: 7

Operaciones Principales:

Uso intensivo de TABLE ACCESS FULL en la tabla LINK.

Operaciones de agrupación (SORT GROUP BY) y uniones (HASH JOIN).

Plan Optimizado

Costo Total: 5

Operaciones Principales:

Uso de INDEX FULL SCAN sobre IDX_SECUELASM_LINK y acceso completo a la vista materializada SECUELASM. Ya que accedemos directamente a la vista materializada mediante un link.

Mejoras Observadas

Las operaciones ya eran muy ligeras de por sí pero hemos conseguido reducirlas ligeramente.

Consulta 3:

Para la segunda consulta hemos juntado todas las views que hemos podido en una sola tabla para lograr una mejora de aproximadamente un 6% al usar menos tablas auxiliares. La principal mejora la hemos conseguido al indexar con estos índices

idx_trabaja_titulo_estreno_tipo ON Trabaja (TITULO, ESTRENO, TIPO);

Este índice es muy útil para las vistas actoresNacionalidad y actricesNacionalidad. Al incluir TITULO, ESTRENO, y TIPO, optimizamos las condiciones del WHERE y las operaciones de JOIN que comparan estos campos.

idx_persona_on_birthplace ON Persona (LUGAR_NACIMIENTO);

Este índice ayuda en la operación de JOIN con LugarNacimiento en las vistas de actores y actrices, facilitando la búsqueda rápida por el lugar de nacimiento para conectarlo con la nacionalidad.

CREATE BITMAP INDEX idx_tipo_trabaja ON Trabaja (TIPO);

Este índice de mapa de bits ayuda en la columna TIPO cuando se filtra por películas.

CREATE BITMAP INDEX idx_papel_trabaja ON Trabaja (PAPEL);

Similar al índice por TIPO, este índice de mapa de bits ayuda al filtrar entre actores y actrices.

Otra mejora que nos planteamos sería el uso de un cluster entre trabaja y personal ya que muchas veces juntamos estas tablas en cuanto a sus id, pero al final la descartamos ya que no nos dejaba crearlo por falta de espacio.

Plan sin Optimizar

Coste Total: 977

Operaciones Optimizadas:

Acceso completo a la tabla PERSONA con 36,135 filas.

Acceso completo a la tabla TRABAJA con 21,187 y 11,472 filas en diferentes partes del plan.

Muchos HASH JOIN y TABLE ACCESS FULL que consumen gran cantidad de recursos.

Plan Optimizado

Costo Total: 177

Operaciones Claves:

Reducción significativa en el uso de TABLE ACCESS FULL, gracias a los índices, ahora también usa TABLE ACCESS BY INDEX ROWID BATCHED y BITMAP INDEX SINGLE VALUE , que es más eficiente. Menor número de filas procesadas en operaciones clave, lo que sugiere una mejor filtración de datos y menos carga de trabajo en el sistema.

Análisis de Mejora

La disminución del costo de 977 a 177 representa una mejora importante en términos de eficiencia del procesamiento. Este cambio indica que el plan optimizado es aproximadamente 82% más eficiente en términos de coste de CPU.

Triggers

Problemas encontrados:

1. Chequeo de Fechas Consistentes en Capítulos: Los capítulos de una serie deben tener fechas de estreno que no sean anteriores a la fecha de estreno de la serie a la que pertenecen.
2. Validación de Tipo en Link: Asegurar que el tipo de producción referenciado en las tablas Link para TITULO y TITULO_LINK sea consistente y correcto para las referencias cruzadas.
3. Consistencia de Tipo en Referencias: Verificar que el TIPO en Produccion, Pelicula, y Serie coincidan cuando se hagan referencias cruzadas para mantener la integridad referencial según el tipo de producción.
4. Coherencia en los Datos de Nacionalidad: Asegurar que el LUGAR_NACIMIENTO en Persona corresponda realmente a un lugar que existe en LugarNacimiento.
5. Validación de la Longitud de SEXO en Persona: Verificar que el campo SEXO solo contenga valores válidos ('f', 'm', '') y manejar correctamente el caso de cadena vacía.
6. Restricción de Periodos de Emisión en Series: Comprobar que el PERIODO_EMISION de la tabla Serie tenga un formato o rango lógico y válido (por ejemplo, no permitir fechas futuras no realistas o períodos que finalicen antes de comenzar).
7. Consistencia de la Importancia del Personaje en InfoPersonaje: Validar que el campo IMPORTANCIA se mantenga dentro de un rango lógico.

8. Verificación de Fechas Futuras en Peliculas y Produccion: Asegurar que las fechas de estreno en Pelicula y Produccion no sean futuras más allá de un año razonable, como un chequeo para evitar introducir datos erróneos o ficticios.
9. Coherencia del Título en la Relación Produccion-Pelicula-Serie: Validar que cuando un título se utilice en Pelicula o Serie, este exista previamente en Produccion y que los detalles como ESTRENO y TIPO coincidan.
10. Integridad en el vínculo entre Persona y Lugar de Nacimiento: Confirmar que cada LUGAR_NACIMIENTO asignado a una Persona realmente exista en la tabla LugarNacimiento.
11. Relación de Link con Película: Garantizar que las películas mencionadas en la tabla Link como TITULO y TITULO_LINK sean efectivamente del tipo 'movie'.

La mayoría de estos fallos son fácilmentes solucionables con triggers sencillos o checks en las tablas, hemos hecho triggers para los 3 que hemos considerado más complejos de solucionar.

1. Crear el campo derivado de nacionalidad
2. Añadir información extra a partir del campo de papel
3. Imponer una restricción de que no puede haber dos capítulos con la misma temporada y número de capítulo para una serie, teniendo en cuenta que como estos campos pueden ser nulos no podemos poner una mera restricción de UNIQUE.
4. También como opcional hemos implementado un trigger que hace un delete on cascade especial

TRIGGER 1

El trigger Before_Insert_LugarNacimiento se ejecuta antes de insertar un registro en la tabla LugarNacimiento. Su objetivo principal es derivar el campo NACIONALIDAD del campo LUGAR_NACIMIENTO, que contiene una cadena de texto con una estructura específica que puede o no incluir una coma.

- Tipo de Trigger: BEFORE INSERT or UPDATE

- Frecuencia: Se ejecuta para cada fila que se va a insertar (FOR EACH ROW).

Cómo lo hemos hecho:

1. Utilizamos la función INSTR para encontrar la última posición de una coma en el campo LUGAR_NACIMIENTO. Esto se realiza buscando desde el final hacia el inicio de la cadena.
2. Extraemos la Nacionalidad:
    - Con Coma Presente: Si se encuentra una coma (last_comma_pos > 0), el trigger extrae la parte de la cadena después de la coma utilizando SUBSTR. Luego, usamos TRIM para eliminar espacios adicionales alrededor de esta subcadena antes de asignarla al campo NACIONALIDAD.
    - Sin Coma: Si no se encuentra una coma, el campo NACIONALIDAD se asigna directamente el valor completo de LUGAR_NACIMIENTO.

Beneficios del Trigger

El objetivo de este trigger es poblar el campo derivado de nacionalidad en la tabla de lugar de nacimiento, la nacionalidad depende del lugar de nacimiento por lo que los extremos de ese campo. De esta manera tenemos poblada permanentemente esa información en la base de datos en vez de tener que extraerla cada vez que necesitamos la nacionalidad, nos planteamos una desnormalización insertando la nacionalidad en persona pero lo descartamos considerando las estrictas limitaciones de memoria con las que nos encontramos.

```
CREATE OR REPLACE TRIGGER Before_Insert_LugarNacimiento
BEFORE INSERT ON LugarNacimiento
FOR EACH ROW
DECLARE
  last_comma_pos INTEGER;
  nationality_part VARCHAR2(150);
BEGIN
  last_comma_pos := INSTR(:NEW.LUGAR_NACIMIENTO, ',', -1, 1);
  IF last_comma_pos > 0 THEN
    nationality_part := SUBSTR(:NEW.LUGAR_NACIMIENTO, last_comma_pos + 1);
```

```
        :NEW.NACIONALIDAD := TRIM(nationality_part);
    ELSE
        :NEW.NACIONALIDAD := :NEW.LUGAR_NACIMIENTO;  -- O maneja de otra manera si no hay
coma
    END IF;
END;
/
```

Trigger 2

Para este trigger en caso de que se inserte en trabaja una persona que trabaja como actor o actriz y no tiene género se añade el género correspondiente

Tipo de trigger: Before Insert or Update, se ejecuta antes de las operaciones de inserción o actualización en la tabla Trabaja.

Frecuencia: FOR EACH ROW

Cómo lo hemos hecho:

1. El trigger verifica si el rol (PAPEL) de la persona que se está insertando o actualizando es actress.
2. Si el rol es actress, el trigger consulta el género (SEXO) de la persona correspondiente en la tabla Persona utilizando el ID de la fila que se está insertando o actualizando.
3. Corrección Automática de Género:
4. Si el género obtenido es NULL, lo que indica que no está definido, el trigger automáticamente actualiza el registro en la tabla Persona y establece el SEXO a 'f' (femenino). Esto se realiza para mantener la coherencia en la representación del género femenino para las actrices. No valoramos cambiar el sexo si existía ya para aceptar todos los casos posibles.

Hay otro trigger análogo para masculino (trigger 5).

```
CREATE OR REPLACE TRIGGER CheckGenderActress

BEFORE INSERT OR UPDATE ON Trabaja

FOR EACH ROW

DECLARE

   v_gender VARCHAR2(2);

   v_count INT;

BEGIN

   IF :NEW.PAPEL = 'actress' THEN

      -- Obtener el género de la persona correspondiente

      SELECT SEXO INTO v_gender FROM Persona WHERE ID = :NEW.ID;



      -- Corregir el género si está vacío o incorrecto

      IF v_gender IS NULL THEN

         UPDATE Persona

            SET SEXO = 'f'

            WHERE ID = :NEW.ID;

      END IF;

   END IF;

END;

/
```

Trigger 3

Este trigger está diseñado para prevenir la inserción de episodios duplicados en la tabla Capitulo, asegurando la unicidad de cada episodio dentro de su respectiva serie y temporada.

Tipo de Trigger: COMPOUND TRIGGER
Frecuencia: Implementado como un trigger compuesto, se ejecuta en dos fases: antes de procesar la instrucción (BEFORE STATEMENT) y antes de insertar cada fila (BEFORE EACH ROW).

Cómo lo hemos hecho:

- Problema de Tablas Mutantes: En Oracle, un trigger no puede consultar la tabla sobre la que se está ejecutando si la operación es de inserción, actualización o eliminación, ya que esto puede llevar a inconsistencias conocidas como el "problema de la tabla mutante".
- Solución Implementada: Para evitar este problema, el trigger compuesto recoge inicialmente todos los episodios existentes en la fase BEFORE STATEMENT y los almacena en la variable episodes del tipo episode_table. Esta colección se utiliza luego para verificar duplicados en la fase BEFORE EACH ROW, sin necesidad de consultar directamente la tabla Capitulo, evitando así el error de tabla mutante.

2. Uso de Tipos de Objeto y Tabla (episode_record y episode_table):
    - Definición de Tipos: Se define episode_record como un tipo de objeto con los atributos de un episodio: serie, temporada y número de capítulo. Basado en este tipo de objeto, se crea episode_table como un tipo de tabla que permite almacenar múltiples registros de tipo episode_record.
    - Propósito: Estos tipos se utilizan para recolectar y manejar datos de episodios como una tabla auxiliar.
3. Antes de la Instrucción (BEFORE STATEMENT):

- Recolección de Datos Existentes: Recolecta todos los episodios existentes en una colección de episode_table, preparando el entorno para la verificación de duplicados en las inserciones subsecuentes.

4. Antes de Cada Fila (BEFORE EACH ROW):
    - Verificación de Duplicados: Examina si el nuevo episodio a insertar ya existe en la colección episodes basándose en serie, temporada y número de capítulo.
    - Manejo de Errores: Si se detecta un duplicado, se lanza un error específico utilizando RAISE_APPLICATION_ERROR, informando al usuario de la violación de la unicidad.

Beneficios del Trigger:

- Aplicamos una restricción a nuestra base de datos.

Trigger 4

Por completitud hemos añadido este trigger que se activa antes de que un registro sea eliminado en la tabla Produccion, asegurándose de que no se eliminen registros que tienen dependencias importantes y dando el mensaje de error correspondiente.

Tipo de Trigger: BEFORE DELETE
Frecuencia: Se ejecuta para cada fila que se va a eliminar (FOR EACH ROW).

Como lo hemos hecho:

1. Según el tipo de produccion se elige un caso
    - Películas:
        - Verificación de Referencias en Link: Antes de proceder con cualquier eliminación, verifica si la película está siendo referenciada en otra película a través de la tabla Link.
        - Manejo de Errores: Si se encuentran referencias, se impide la eliminación y se lanza un error específico.
        - Eliminación de Dependencias: Si no hay referencias, se procede a eliminar registros relacionados de las tablas InfoPersonaje, Trabaja, Genero, Link y Pelicula.

- Episodios :
    - Eliminación Directa: Se eliminan registros relacionados directamente sin verificaciones adicionales de referencias externas, afectando a las tablas InfoPersonaje, Trabaja, Género y Capítulo.
- Series:
    - Verificación de Capítulos Referenciados: Verifica si hay capítulos que dependan de la serie especificada.
    - Manejo de Errores: Si se encuentran capítulos dependientes, se impide la eliminación y se emite un mensaje de error.
    - Eliminación de Dependencias: Si no hay capítulos dependientes, se eliminan registros de las tablas InfoPersonaje, Trabaja, Genero, Capítulo y Serie.

Beneficios del Trigger:

Este trigger se usa para la eliminación de tuplas en producción, que el caso más extremo de delete ya que es referenciado por una gran cantidad de de tuplas. Necesitábamos un trigger que funcionara de manera similar al ON DELETE CASCADE pero que tuviera más flexibilidad a la hora de dar errores. Los 2 casos problemáticos que hemos detectado son en el caso de películas si la película es referenciada por otra, en este caso si borráramos de la tabla podríamos perder información sobre la relación, en este caso detenemos el borrado. El otro problema es en serie, ya que si borráramos una serie se borrarían también todos sus capítulos de los que habría que borrar también sus relaciones y obras lo que podría resultar muy complejo y eliminaría una gran cantidad de información demasiado rápido por lo que hemos decidido que para borrar series primero haya que borrar todos sus capítulos uno por uno y luego borrar la serie.

```
CREATE OR REPLACE TRIGGER Delete_Cascade_Produccion

BEFORE DELETE ON Produccion

FOR EACH ROW

DECLARE

   v_count_link INT;

   v_count_capitulo INT;

BEGIN



   IF (:OLD.TIPO = 'movie') THEN

      -- Verificar si la película está siendo referenciada en la tabla Link

      SELECT COUNT(*)

      INTO v_count_link

      FROM Link

      WHERE TITULO_LINK = :OLD.TITULO AND ESTRENO_LINK = :OLD.ESTRENO AND TIPO =
:OLD.TIPO;



      IF v_count_link > 0 THEN

         RAISE_APPLICATION_ERROR(-20002, 'No se puede borrar la película porque es referenciada
en otra película.');

      END IF;



      DELETE FROM InfoPersonaje

      WHERE TITULO = :OLD.TITULO AND ESTRENO = :OLD.ESTRENO AND TIPO = :OLD.TIPO;



      DELETE FROM Trabaja
```

```
    WHERE TITULO = :OLD.TITULO AND ESTRENO = :OLD.ESTRENO AND TIPO = :OLD.TIPO;

    DELETE FROM Genero
    WHERE TITULO = :OLD.TITULO AND ESTRENO = :OLD.ESTRENO AND TIPO = :OLD.TIPO;

    DELETE FROM Link
    WHERE TITULO = :OLD.TITULO AND ESTRENO = :OLD.ESTRENO AND TIPO = :OLD.TIPO;

    DELETE FROM Pelicula
    WHERE TITULO = :OLD.TITULO AND ESTRENO = :OLD.ESTRENO AND TIPO = :OLD.TIPO;
END IF;

IF (:OLD.TIPO = 'episode') THEN
    DELETE FROM InfoPersonaje
    WHERE TITULO = :OLD.TITULO AND ESTRENO = :OLD.ESTRENO AND TIPO = :OLD.TIPO;

    DELETE FROM Trabaja
    WHERE TITULO = :OLD.TITULO AND ESTRENO = :OLD.ESTRENO AND TIPO = :OLD.TIPO;

    DELETE FROM Genero
    WHERE TITULO = :OLD.TITULO AND ESTRENO = :OLD.ESTRENO AND TIPO = :OLD.TIPO;

    DELETE FROM Capitulo
    WHERE TITULO = :OLD.TITULO AND ESTRENO = :OLD.ESTRENO AND TIPO = :OLD.TIPO;
END IF;
```

```
    IF (:OLD.TIPO = 'tv series') THEN

        -- Verificar si la serie tiene capítulos referenciados

        SELECT COUNT(*)

        INTO v_count_capitulo

        FROM Capitulo

        WHERE TITULO_SERIE = :OLD.TITULO AND ESTRENO_SERIE = :OLD.ESTRENO AND
TIPO_SERIE = :OLD.TIPO;



        IF v_count_capitulo > 0 THEN

            RAISE_APPLICATION_ERROR(-20003, 'No se puede borrar la serie mientras tenga capítulos
referenciados.');

        END IF;



        DELETE FROM InfoPersonaje

        WHERE TITULO = :OLD.TITULO AND ESTRENO = :OLD.ESTRENO AND TIPO = :OLD.TIPO;



        DELETE FROM Trabaja

        WHERE TITULO = :OLD.TITULO AND ESTRENO = :OLD.ESTRENO AND TIPO = :OLD.TIPO;



        DELETE FROM Genero

        WHERE TITULO = :OLD.TITULO AND ESTRENO = :OLD.ESTRENO AND TIPO = :OLD.TIPO;



        DELETE FROM Capitulo

        WHERE TITULO_SERIE = :OLD.TITULO AND ESTRENO_SERIE = :OLD.ESTRENO AND
TIPO_SERIE = :OLD.TIPO;
```

```
        DELETE FROM Serie
        WHERE TITULO = :OLD.TITULO AND ESTRENO = :OLD.ESTRENO AND TIPO = :OLD.TIPO;
    END IF;

END;
/
```

CONCLUSIÓN

La división de horas en cada parte ha sido la siguiente:

	Diego	Carlos	Daniel
Parte 1	16	15	17
Parte 2	13	15	15
Parte 3	16	16	13
	45	46	45

El principal problema que hemos tenido ha sido la falta de espacio debido a la cuál no hemos podido probar la mayoría de las optimizaciones, aparte de eso nos hemos coordinado bien y no hemos tenido problemas de tiempo ya que nos hemos planificado bien.

ANEXO

CONSULTA 1 OPTIMIZADA

```
| Id  | Operation                          | Name
| Rows  | Bytes | Cost (%CPU)| Time     |

-------------------------------------------------------------------------------
----------------------------------------


PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                   |
|     1 |   199 |    24   (9)| 00:00:01 |

|   1 |  HASH UNIQUE                       |
|     1 |   199 |    24   (9)| 00:00:01 |

|   2 |   NESTED LOOPS                     |
|     1 |   199 |    20   (5)| 00:00:01 |

|   3 |    NESTED LOOPS                    |
|     1 |   199 |    20   (5)| 00:00:01 |

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------

|*  4 |     HASH JOIN SEMI                 |
|     1 |   129 |    19   (6)| 00:00:01 |

|*  5 |      HASH JOIN                     |
|    36 |  3528 |    10   (0)| 00:00:01 |

|   6 |       INDEX FAST FULL SCAN         | IDX_DIRECTORM_ID_ESTRENO
|  2769 | 85839 |     5   (0)| 00:00:01 |

|   7 |       INLIST ITERATOR              |

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
|     |     |          |        |

|*  8 |        INDEX RANGE SCAN            | SYS_C00616749
|  2234 |   146K|     5   (0)| 00:00:01 |

|   9 |        SORT AGGREGATE              |
|     1 |    85 |          |        |

|  10 |         NESTED LOOPS               |
|     1 |    85 |     3   (0)| 00:00:01 |


PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
|* 11 |          INDEX UNIQUE SCAN         | SYS_C00616747
|     1 |    18 |     1   (0)| 00:00:01 |

|* 12 |          TABLE ACCESS BY INDEX ROWID BATCHED| TRABAJA
|     1 |    67 |     2   (0)| 00:00:01 |

|* 13 |           INDEX RANGE SCAN         | IDX_TRABAJA_ID
|     1 |       |     1   (0)| 00:00:01 |

|  14 |      VIEW                          | VW_SQ_1
|  2769 | 85839 |     9  (12)| 00:00:01 |
```

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

|  15 |      HASH GROUP BY                      |
|  2769 | 85839 |     9  (12)| 00:00:01 |

|  16 |       MAT_VIEW ACCESS FULL              | DIRECTORM
|  2769 | 85839 |     8   (0)| 00:00:01 |

|* 17 |     INDEX UNIQUE SCAN                   | SYS_C00616747
|     1 |       |     0   (0)| 00:00:01 |

|  18 |    TABLE ACCESS BY INDEX ROWID          | PERSONA
```

CONSULTA 1 SIN OPTIMIZAR

```
-------------------------------------------------------------------------------
Plan hash value: 1756333012

-------------------------------------------------------------------------------
---------------

| Id  | Operation                    | Name         | Rows  | Bytes | Cost (%
CPU)| Time     |

-------------------------------------------------------------------------------
---------------


PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |              |     2 |   444 |    48
 (3)| 00:00:01 |

|   1 |  HASH UNIQUE                 |              |     2 |   444 |    48
 (3)| 00:00:01 |

|   2 |   NESTED LOOPS SEMI          |              |     2 |   444 |    42
 (0)| 00:00:01 |

|   3 |    NESTED LOOPS              |              |     2 |   408 |    42
 (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------

|*  4 |     HASH JOIN                |              |     2 |   268 |    40
 (0)| 00:00:01 |

|*  5 |      INDEX RANGE SCAN        | SYS_C00616749 |   128 |  8576 |    35
 (0)| 00:00:01 |

|   6 |      SORT AGGREGATE          |              |     1 |    85 |
   |        |

|   7 |       NESTED LOOPS           |              |     1 |    85 |     2

PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
 (0)| 00:00:01 |

|*  8 |        INDEX UNIQUE SCAN     | SYS_C00616747 |     1 |    18 |     1
 (0)| 00:00:01 |

|*  9 |        INDEX RANGE SCAN      | SYS_C00616749 |     1 |    67 |     1
 (0)| 00:00:01 |

|  10 |      INLIST ITERATOR         |              |       |       |
   |        |


PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
|* 11 |       INDEX RANGE SCAN       | SYS_C00616749 |  2234 |  146K|     5
 (0)| 00:00:01 |

|  12 |       SORT AGGREGATE         |              |     1 |    85 |
   |        |

|  13 |        NESTED LOOPS          |              |     1 |    85 |     3
 (0)| 00:00:01 |
```

```
|* 14 |         INDEX UNIQUE SCAN     | SYS_C00616747 |     1 |    18 |     1
 (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

|  15 |         INLIST ITERATOR       |               |       |       |
   |         |

|* 16 |          INDEX RANGE SCAN     | SYS_C00616749 |     1 |    67 |     2
 (0)| 00:00:01 |

|  17 |     TABLE ACCESS BY INDEX ROWID| PERSONA       |     1 |    70 |     1
 (0)| 00:00:01 |

|* 18 |      INDEX UNIQUE SCAN        | SYS_C00616747 |     1 |       |     0

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
 (0)| 00:00:01 |

|* 19 |    INDEX UNIQUE SCAN          | SYS_C00616747 | 39226 |   689K|     0
 (0)| 00:00:01 |

--------------------------------------------------------------------------------
---------------


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

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

   4 - access("T"."ID"="T"."ID")
       filter("T"."ESTRENO">="T"."ESTRENO")
   5 - access("T"."PAPEL"='director' AND "T"."TIPO"='movie')
       filter("T"."TIPO"='movie' AND "T"."ESTRENO"= (SELECT MAX("T"."ESTRENO") F
ROM

            "A870984"."PERSONA" "P","A870984"."TRABAJA" "T" WHERE "T"."ID"="P"
."ID" AND

            "T"."PAPEL"='director' AND "T"."ID"=:B1 AND "T"."TIPO"='movie' AND

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
 "P"."ID"=:B2))

   8 - access("P"."ID"=:B1)
   9 - access("T"."PAPEL"='director' AND "T"."ID"="P"."ID" AND "T"."TIPO"='movie
')

       filter("T"."ID"=:B1 AND "T"."TIPO"='movie')
  11 - access(("T"."PAPEL"='actor' OR "T"."PAPEL"='actress') AND "T"."TIPO"='mov
ie')

       filter("T"."TIPO"='movie' AND "T"."ESTRENO"= (SELECT MAX("T"."ESTRENO") F

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
ROM

            "A870984"."PERSONA" "P","A870984"."TRABAJA" "T" WHERE "T"."ID"="P"
."ID" AND
```

```
          ("T"."PAPEL"='actor' OR "T"."PAPEL"='actress') AND "T"."ID"=:B1 AN
D "T"."TIPO"='movie'

          AND "P"."ID"=:B2))
  14 - access("P"."ID"=:B1)
  16 - access(("T"."PAPEL"='actor' OR "T"."PAPEL"='actress') AND "T"."ID"="P"."I

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
D" AND

          "T"."TIPO"='movie')
      filter("T"."ID"=:B1 AND "T"."TIPO"='movie')
  18 - access("T"."ID"="P"."ID")
  19 - access("T"."ID"="P"."ID")

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

55 filas seleccionadas.
```

CONSULTA 2 OPTIMIZADA

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 1587174943

--------------------------------------------------------------------------------
---------------

| Id  | Operation                | Name             | Rows  | Bytes | Cost (%
CPU)| Time     |

--------------------------------------------------------------------------------
---------------


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT         |                  |    67 | 23048 |     5
(20)| 00:00:01 |

|   1 |  SORT UNIQUE             |                  |    67 | 23048 |     5
(20)| 00:00:01 |

|*  2 |   FILTER                 |                  |       |       |
    |          |

|   3 |    SORT GROUP BY         |                  |    67 | 23048 |     5
(20)| 00:00:01 |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

|*  4 |     HASH JOIN            |                  |    67 | 23048 |     4
 (0)| 00:00:01 |

|   5 |      INDEX FULL SCAN     | IDX_SECUELASM_LINK |    51 |  5865 |     1
 (0)| 00:00:01 |

|   6 |      MAT_VIEW ACCESS FULL | SECUELASM        |    51 | 11679 |     3
 (0)| 00:00:01 |

|   7 |    SORT AGGREGATE        |                  |     1 |    13 |
    |          |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

|   8 |     VIEW                 | NUMSECUELAS      |    51 |   663 |     4
(25)| 00:00:01 |

|   9 |      SORT GROUP BY       |                  |    51 |  5865 |     4
(25)| 00:00:01 |

|  10 |       MAT_VIEW ACCESS FULL| SECUELASM        |    51 |  5865 |     3
 (0)| 00:00:01 |


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
---------------


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

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

   2 - filter(COUNT(*)= (SELECT MAX("NUM_SECUELAS") FROM  (SELECT "TITULO_LINK"
           "TITULO_LINK","ESTRENO_LINK" "ESTRENO_LINK",COUNT(*) "NUM_SECUELAS
" FROM


PLAN_TABLE_OUTPUT
----------------------------------------------------------------------------------
           "A870984"."SECUELASM" "SECUELASM" GROUP BY "TITULO_LINK","ESTRENO_
LINK") "S"))

   4 - access("S"."TITULO_LINK"="TITULO_LINK")

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

CONSULTA 3 OPTIMIZADA

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 2212442882

--------------------------------------------------------------------------------
---------------------------------

| Id  | Operation                           | Name            | Rows
| Bytes | Cost (%CPU)| Time     |

--------------------------------------------------------------------------------
---------------------------------


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                    |                 |    1
|       |   177   (2)| 00:00:01 |

|   1 |  SORT AGGREGATE                     |                 |    1
|       |          |        |

|   2 |   VIEW                              | PAREJASACTORES  |  491
|       |   177   (2)| 00:00:01 |

|*  3 |    FILTER                           |                 |
|       |          |        |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

|   4 |     HASH GROUP BY                   |                 |  491
|   426K|   177   (2)| 00:00:01 |

|*  5 |      HASH JOIN                      |                 |  491
|   426K|   176   (1)| 00:00:01 |

|   6 |       TABLE ACCESS FULL             | LUGARNACIMIENTO |  3057
|   385K|     7   (0)| 00:00:01 |

|*  7 |       HASH JOIN                     |                 |  2459

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
|  1827K|   169   (1)| 00:00:01 |

|*  8 |        HASH JOIN                    |                 |  2459
|  1474K|   100   (0)| 00:00:01 |

|*  9 |         HASH JOIN                   |                 |  2299
|   999K|    85   (2)| 00:00:01 |

|* 10 |          TABLE ACCESS BY INDEX ROWID BATCHED| TRABAJA         | 11472
|  1893K|     9   (0)| 00:00:01 |


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
|  11 |           BITMAP CONVERSION TO ROWIDS      |                 |
|       |          |        |

|  12 |            BITMAP AND                      |                 |
|       |          |        |

|* 13 |             BITMAP INDEX SINGLE VALUE      | IDX_PAPEL_TRABAJA |
```

```
|     |          |        |

|* 14 |           BITMAP INDEX SINGLE VALUE       | IDX_TIPO_TRABAJA  |
|     |          |        |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

|* 15 |         HASH JOIN                         |                 |  7243
|  1952K|    75   (0)| 00:00:01 |

|  16 |          TABLE ACCESS FULL                | LUGARNACIMIENTO   |  3057
|   385K|     7   (0)| 00:00:01 |

|  17 |          TABLE ACCESS FULL                | PERSONA          | 36135
|  5187K|    68   (0)| 00:00:01 |

|* 18 |        TABLE ACCESS BY INDEX ROWID BATCHED | TRABAJA          | 21187

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
|  3496K|    15   (0)| 00:00:01 |

|  19 |         BITMAP CONVERSION TO ROWIDS        |                 |
|     |          |        |

|  20 |          BITMAP AND                       |                 |
|     |          |        |

|* 21 |           BITMAP INDEX SINGLE VALUE       | IDX_PAPEL_TRABAJA |
|     |          |        |


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
|* 22 |           BITMAP INDEX SINGLE VALUE       | IDX_TIPO_TRABAJA  |
|     |          |        |

|  23 |        TABLE ACCESS FULL                  | PERSONA          | 36135
|  5187K|    68   (0)| 00:00:01 |

|  24 |     SORT AGGREGATE                        |                 |     1
|   162 |          |        |

|  25 |      NESTED LOOPS                         |                 |     1
|   162 |     3   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

|* 26 |       TABLE ACCESS BY INDEX ROWID         | PERSONA          |     1
|    95 |     2   (0)| 00:00:01 |

|* 27 |        INDEX UNIQUE SCAN                  | SYS_C00625412     |     1
|       |     1   (0)| 00:00:01 |

|* 28 |       INDEX RANGE SCAN                    | SYS_C00625414     |     1
|    67 |     1   (0)| 00:00:01 |

|  29 |     SORT AGGREGATE                        |                 |     1

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
|   162 |          |        |

|  30 |      NESTED LOOPS                         |                 |     1
|   162 |     3   (0)| 00:00:01 |
```

```
|* 31 |      TABLE ACCESS BY INDEX ROWID          | PERSONA          |     1
|    95 |     2   (0)| 00:00:01 |

|* 32 |       INDEX UNIQUE SCAN                   | SYS_C00625412    |     1
|       |     1   (0)| 00:00:01 |


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
|* 33 |      INDEX RANGE SCAN                     | SYS_C00625414    |     1
|    67 |     1   (0)| 00:00:01 |

--------------------------------------------------------------------------------
----------------------------------


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

   3 - filter(COUNT(*)= (SELECT COUNT(*) FROM "A870984"."PERSONA" "P","A870984".

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
"TRABAJA" "T" WHERE

            "T"."ESTRENO"<=2010 AND "T"."ESTRENO">=1980 AND "T"."ID"="P"."ID"
AND "T"."PAPEL"='actor' AND

            "T"."ID"=:B1 AND "T"."TIPO"='movie' AND "P"."ID"=:B2 AND "P"."LUGA
R_NACIMIENTO" IS NOT NULL) AND

            COUNT(*)= (SELECT COUNT(*) FROM "A870984"."PERSONA" "P","A870984".
"TRABAJA" "T" WHERE


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
            "T"."ESTRENO"<=2010 AND "T"."ESTRENO">=1980 AND "T"."ID"="P"."ID"
AND "T"."PAPEL"='actress' AND

            "T"."ID"=:B3 AND "T"."TIPO"='movie' AND "P"."ID"=:B4 AND "P"."LUGA
R_NACIMIENTO" IS NOT NULL))

   5 - access("P"."LUGAR_NACIMIENTO"="LN"."LUGAR_NACIMIENTO")
       filter("LN"."NACIONALIDAD"<>"LN"."NACIONALIDAD")
   7 - access("T"."ID"="P"."ID")
   8 - access("T"."TITULO"="T"."TITULO" AND "T"."ESTRENO"="T"."ESTRENO")
   9 - access("T"."ID"="P"."ID")

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
  10 - filter("T"."ESTRENO">=1980 AND "T"."ESTRENO"<=2010)
  13 - access("T"."PAPEL"='actress')
  14 - access("T"."TIPO"='movie')
  15 - access("P"."LUGAR_NACIMIENTO"="LN"."LUGAR_NACIMIENTO")
  18 - filter("T"."ESTRENO">=1980 AND "T"."ESTRENO"<=2010)
  21 - access("T"."PAPEL"='actor')
  22 - access("T"."TIPO"='movie')
  26 - filter("P"."LUGAR_NACIMIENTO" IS NOT NULL)
  27 - access("P"."ID"=:B1)
  28 - access("T"."PAPEL"='actor' AND "T"."ID"="P"."ID" AND "T"."ESTRENO">=1980
AND "T"."TIPO"='movie'

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
```

```
           AND "T"."ESTRENO"<=2010)
      filter("T"."ID"=:B1 AND "T"."TIPO"='movie')
  31 - filter("P"."LUGAR_NACIMIENTO" IS NOT NULL)
  32 - access("P"."ID"=:B1)
  33 - access("T"."PAPEL"='actress' AND "T"."ID"="P"."ID" AND "T"."ESTRENO">=198
0 AND "T"."TIPO"='movie'

           AND "T"."ESTRENO"<=2010)
      filter("T"."ID"=:B1 AND "T"."TIPO"='movie')


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

76 filas seleccionadas.
```

CONSULTA 3 SIN OPTIMIZAR

```
PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
Plan hash value: 2136859915

--------------------------------------------------------------------------------
----------------------

| Id  | Operation                | Name                | Rows  | Bytes |
 Cost (%CPU)| Time     |

--------------------------------------------------------------------------------
----------------------


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |                     |     1 |      |
  977   (2)| 00:00:01 |

|   1 |  SORT AGGREGATE            |                     |     1 |      |
          |        |

|   2 |   VIEW                     | PAREJASACTORES      |     1 |      |
  977   (2)| 00:00:01 |

|   3 |    HASH UNIQUE             |                     |     1 |   319 |
  977   (2)| 00:00:01 |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

|*  4 |     HASH JOIN SEMI         |                     |     1 |   319 |
  976   (2)| 00:00:01 |

|*  5 |      HASH JOIN             |                     |   146 | 42048 |
  735   (1)| 00:00:01 |

|   6 |       VIEW                 | CONTADORACTORES     |  4247 |   128K|
  241   (2)| 00:00:01 |

|   7 |        NESTED LOOPS        |                     |  4247 |  1273K|

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
  241   (2)| 00:00:01 |

|*  8 |         HASH JOIN          |                     | 21187 |  3682K|
  240   (1)| 00:00:01 |

|   9 |          VIEW              | VW_GBF_108          | 21187 |   641K|
  172   (2)| 00:00:01 |

|  10 |           HASH GROUP BY    |                     | 21187 |  1386K|
  172   (2)| 00:00:01 |


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
|* 11 |            TABLE ACCESS FULL | TRABAJA             | 21187 |  1386K|
  171   (1)| 00:00:01 |

|  12 |          TABLE ACCESS FULL   | PERSONA             | 36135 |  5187K|
   68   (0)| 00:00:01 |

|* 13 |         INDEX UNIQUE SCAN    | SYS_C00625409       |     1 |   129 |
```

```
    0   (0)| 00:00:01 |

|  14 |      VIEW                | NUMCOINCIDENCIASACTORES | 25476 |  6393K|
   494   (1)| 00:00:01 |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

|  15 |       HASH GROUP BY        |                         | 25476 |    17M|
   494   (1)| 00:00:01 |

|* 16 |        HASH JOIN           |                         | 25476 |    17M|
   492   (1)| 00:00:01 |

|  17 |         VIEW               | VW_GBF_72               |  7243 |   884K|
    75   (0)| 00:00:01 |

|* 18 |          HASH JOIN         |                         |  7243 |  1952K|

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
    75   (0)| 00:00:01 |

|  19 |           TABLE ACCESS FULL | LUGARNACIMIENTO         |  3057 |   385K|
     7   (0)| 00:00:01 |

|  20 |           TABLE ACCESS FULL | PERSONA                 | 36135 |  5187K|
    68   (0)| 00:00:01 |

|* 21 |         HASH JOIN          |                         | 26011 |    15M|
   417   (1)| 00:00:01 |


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
|* 22 |          HASH JOIN         |                         |  2299 |   999K|
   246   (1)| 00:00:01 |

|* 23 |           TABLE ACCESS FULL | TRABAJA                 | 11472 |  1893K|
   171   (1)| 00:00:01 |

|* 24 |          HASH JOIN         |                         |  7243 |  1952K|
    75   (0)| 00:00:01 |

|  25 |           TABLE ACCESS FULL| LUGARNACIMIENTO         |  3057 |   385K|
     7   (0)| 00:00:01 |

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------

|  26 |           TABLE ACCESS FULL| PERSONA                 | 36135 |  5187K|
    68   (0)| 00:00:01 |

|* 27 |          TABLE ACCESS FULL  | TRABAJA                 | 21187 |  3496K|
   171   (1)| 00:00:01 |

|  28 |      VIEW                  | CONTADORACTRICES        |  2299 | 71269 |
   241   (2)| 00:00:01 |

|  29 |       NESTED LOOPS         |                         |  2299 |   689K|

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
   241   (2)| 00:00:01 |

|* 30 |        HASH JOIN           |                         | 11472 |  1994K|
   240   (1)| 00:00:01 |
```

```
|  31 |        VIEW               | VW_GBF_90             | 11472 |   347K|
   172   (2)| 00:00:01 |

|  32 |         HASH GROUP BY     |                       | 11472 |   750K|
   172   (2)| 00:00:01 |


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
|* 33 |          TABLE ACCESS FULL  | TRABAJA               | 11472 |   750K|
   171   (1)| 00:00:01 |

|  34 |        TABLE ACCESS FULL    | PERSONA               | 36135 |  5187K|
    68   (0)| 00:00:01 |

|* 35 |       INDEX UNIQUE SCAN     | SYS_C00625409         |     1 |   129 |
     0   (0)| 00:00:01 |

--------------------------------------------------------------------------------
------------------------

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------


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

   4 - access("N"."ID_ACTRIZ"="B"."ID_ACTRIZ" AND "A"."NUM_OBRAS"="B"."NUM_OBRAS
")

   5 - access("A"."NUM_OBRAS"="N"."NUM_COINCIDENCIAS" AND "N"."ID_ACTOR"="A"."ID
_ACTOR")


PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
   8 - access("ITEM_1"="P"."ID")
  11 - filter("T"."TIPO"='movie' AND "T"."PAPEL"='actor' AND "T"."ESTRENO"<=2010
 AND

            "T"."ESTRENO">=1980)
  13 - access("P"."LUGAR_NACIMIENTO"="P"."LUGAR_NACIMIENTO")
  16 - access("T"."ID"="ITEM_2")
       filter("ITEM_1"<>"P"."NACIONALIDAD" OR "ITEM_1" IS NULL OR "P"."NACIONALI
DAD" IS NULL)

  18 - access("P"."LUGAR_NACIMIENTO"="P"."LUGAR_NACIMIENTO")

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
  21 - access("T"."TITULO"="T"."TITULO")
  22 - access("T"."ID"="P"."ID")
  23 - filter("T"."PAPEL"='actress' AND "T"."TIPO"='movie' AND "T"."ESTRENO">=19
80 AND

            "T"."ESTRENO"<=2010)
  24 - access("P"."LUGAR_NACIMIENTO"="P"."LUGAR_NACIMIENTO")
  27 - filter("T"."PAPEL"='actor' AND "T"."TIPO"='movie' AND "T"."ESTRENO">=1980
 AND

            "T"."ESTRENO"<=2010)

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------
  30 - access("ITEM_1"="P"."ID")
```

```
  33 - filter("T"."TIPO"='movie' AND "T"."PAPEL"='actress' AND "T"."ESTRENO"<=20
10 AND

            "T"."ESTRENO">=1980)
  35 - access("P"."LUGAR_NACIMIENTO"="P"."LUGAR_NACIMIENTO")

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

71 filas seleccionadas.
```