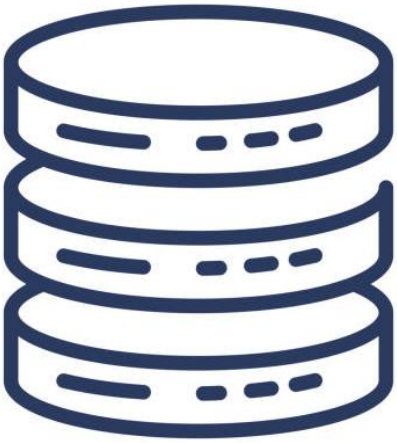
BASES DE DATOS

2023



PRÁCTICA 2

Fecha de entrega
30/06/2023

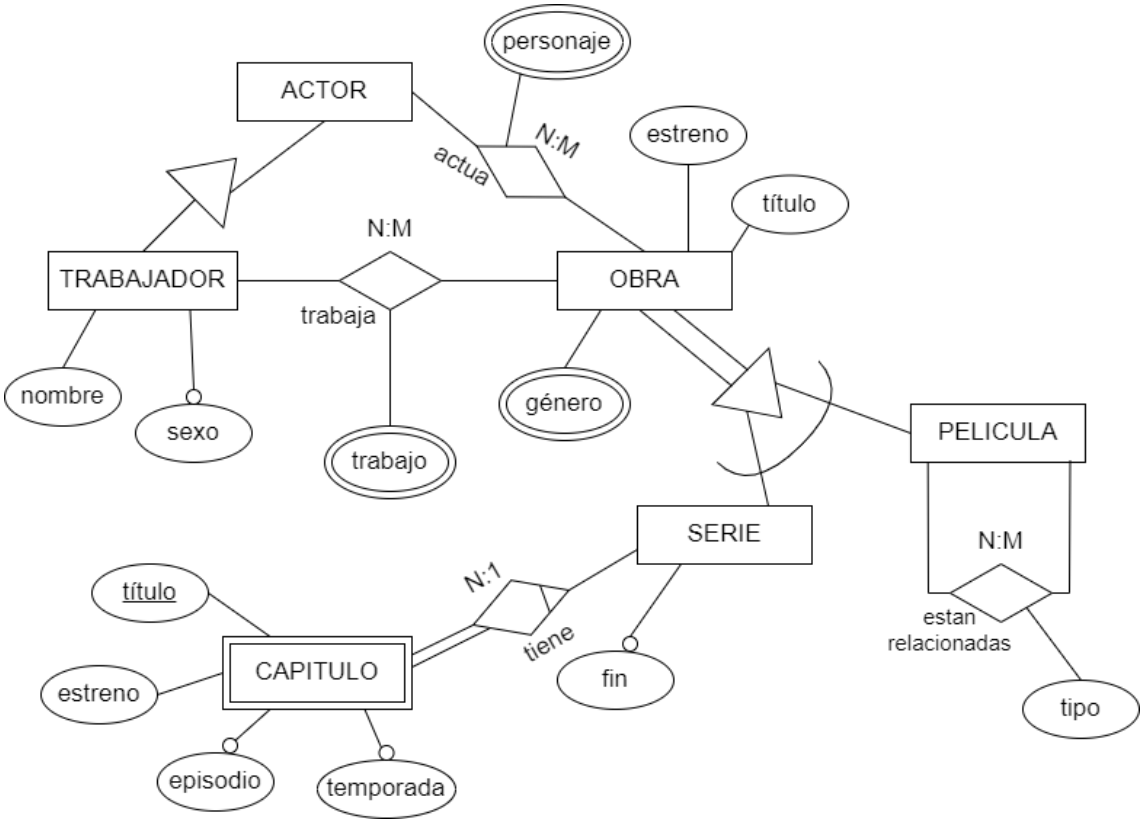
Grupo 11
Inés Román Gracia, 820731@unizar.es

ÍNDICE

Contenido

PARTE 1: CREACIÓN DE LA BASE DE DATOS

1.1. Modelo Entidad-Relación



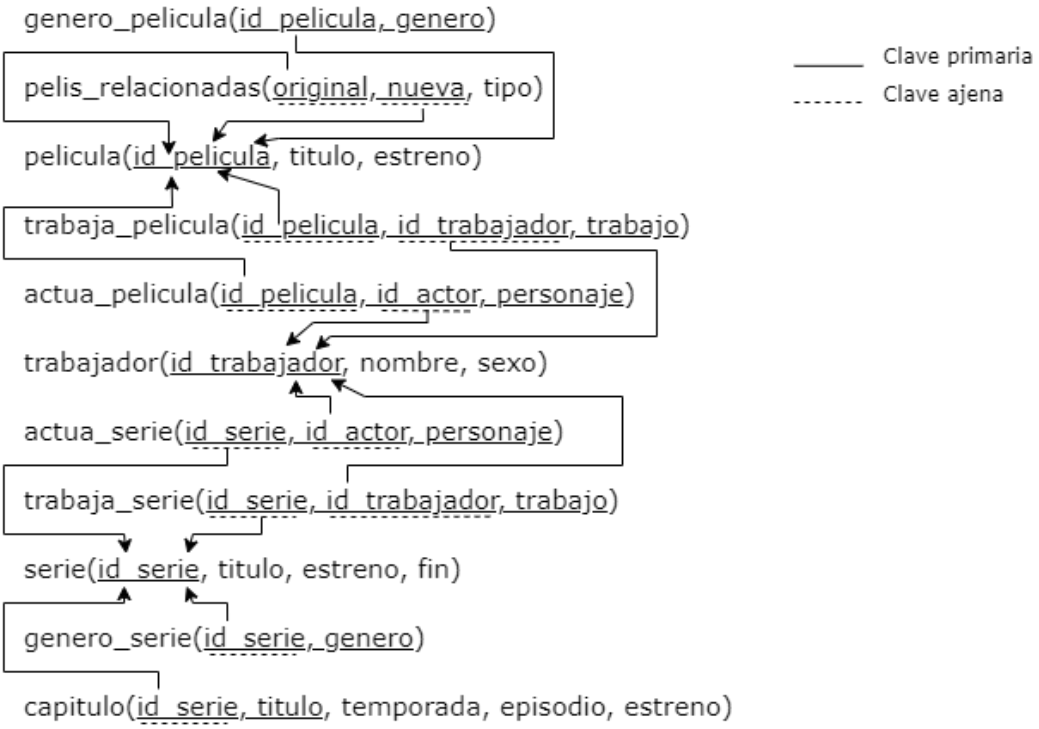
Restricciones

- Los años de estreno y de fin de las obras y capítulos tienen que estar comprendidos entre 1895 y 2023, la fecha de la primera película y la fecha actual.
- El año de fin de una serie debe ser mayor o igual que su año de estreno.
- Los capítulos que tiene una serie deben tener años de estreno comprendidos entre el año de estreno y de fin de la serie.
- En la relación de películas “están relacionadas” una película no puede estar relacionada consigo misma.

Soluciones alternativas

- Se podría haber creado una agregación entre obra y trabajador, con la que se relacionaría una nueva entidad para los personajes.
- Se podría crear una relación entre obras y trabajadores que relacione las obras con sus directores, ya que esta podría considerarse una información de relevancia para las obras.
- Podría haberse modelado la relación entre trabajador y obra con una relación ternaria con una nueva entidad para el tipo de trabajo. Esto obligaría a todos los trabajadores y obras a participar en esta relación. Por lo que todos los actores deberían participar también en esta relación con un rol de actor.

1.2. Modelo Relacional



Decisiones de diseño en las especializaciones

En la especialización de *trabajador*, se ha optado por la opción 2: mantener únicamente la tabla *trabajador*. La tabla *trabajador* debe mantenerse ya que hay trabajadores sin especialización. Frente a la opción 1 de mantener ambas tablas, esta solución tiene la ventaja de no tener que hacer una intersección más para acceder a los datos de *actor* que están en la tabla *trabajador*. Además, no resulta un inconveniente que la relación de actores y obras sea accesible para todos los trabajadores, ya que el hecho de participar en ella es lo que convierte a un trabajador en actor.

En la especialización de *obra* se ha optado por la opción 3: mantener únicamente las tablas especializadas de *película* y *serie*. La opción 2 sería muy inconveniente porque habría que añadir muchas restricciones para que películas y series no puedan participar en las relaciones de la otra. La opción 1, al igual que la 3, evitaría este problema y evitaría también tener que duplicar todas las relaciones de *obra*, pero tiene la desventaja de que hay que hacer una intersección extra para que películas y series puedan acceder a sus atributos comunes. En este caso, no resulta inconveniente la duplicación de relaciones de *obra* ya que las consultas van de series o de películas, pero no de ambas, por lo que se ha visto más conveniente evitar la intersección que trae como desventaja la opción 1.

Normalización

El modelo se encuentra en 1ªFN ya que los atributos multivaluados se han repartido a otras tablas (como genero_pelicula) o forman parte de la clave primaria (como personaje en actúa_pelicula).

El modelo se encuentra en 2ºFN ya que todos los atributos que forman parte de claves compuestas no dependen unos de otros.

El modelo se encuentra en 3ªFN y 3ªFNBC ya que todos los atributos dependes de su clave.

1.3. Sentencias SQL de creación de tablas

```
CREATE TABLE trabajador
(
   id_trabajador          NUMBER PRIMARY KEY,
   nombre                 VARCHAR(50) NOT NULL,
   sexo                   VARCHAR(1)
);

CREATE TABLE pelicula
(
   id_pelicula            NUMBER PRIMARY KEY,
   estreno                NUMBER NOT NULL,
   titulo                 VARCHAR(150) NOT NULL
);

CREATE TABLE genero_pelicula
(
   id_pelicula            NUMBER,
   genero                 VARCHAR(15),
   PRIMARY KEY(id_pelicula, genero),
   FOREIGN KEY(id_pelicula) REFERENCES pelicula(id_pelicula) ON DELETE CASCADE
);

CREATE TABLE trabaja_pelicula
(
   id_trabajador          NUMBER,
   id_pelicula            NUMBER,
   trabajo                VARCHAR(80),
   PRIMARY KEY(id_trabajador, id_pelicula, trabajo),
   FOREIGN KEY(id_trabajador) REFERENCES trabajador(id_trabajador) ON DELETE CASCADE,
   FOREIGN KEY(id_pelicula) REFERENCES pelicula(id_pelicula) ON DELETE CASCADE
);

CREATE TABLE actua_pelicula
(
   id_actor               NUMBER,
   id_pelicula            NUMBER,
   personaje              VARCHAR(80),
   PRIMARY KEY(id_actor, id_pelicula, personaje),
   FOREIGN KEY(id_actor) REFERENCES trabajador(id_trabajador) ON DELETE CASCADE,
   FOREIGN KEY(id_pelicula) REFERENCES pelicula(id_pelicula) ON DELETE CASCADE
);

CREATE TABLE pelis_relacionadas
(
   original               NUMBER,
   nueva                  NUMBER,
   tipo                   VARCHAR(15) NOT NULL,
   PRIMARY KEY(original, nueva),
   FOREIGN KEY(original) REFERENCES pelicula(id_pelicula) ON DELETE CASCADE,
   FOREIGN KEY(nueva) REFERENCES pelicula(id_pelicula) ON DELETE CASCADE
```

```
);

CREATE TABLE serie
(
   id_serie                NUMBER PRIMARY KEY,
   estreno                 NUMBER NOT NULL,
   titulo                  VARCHAR(150) NOT NULL,
   fin                      NUMBER
);

CREATE TABLE genero_serie
(
   id_serie                NUMBER,
   genero                  VARCHAR(15),
   PRIMARY KEY(id_serie, genero),
   FOREIGN KEY(id_serie) REFERENCES serie(id_serie) ON DELETE CASCADE
);

CREATE TABLE trabaja_serie
(
   id_trabajador           NUMBER,
   id_serie                NUMBER,
   trabajo                 VARCHAR(80),
   PRIMARY KEY(id_trabajador, id_serie, trabajo),
   FOREIGN KEY(id_trabajador) REFERENCES trabajador(id_trabajador) ON DELETE CASCADE,
   FOREIGN KEY(id_serie) REFERENCES serie(id_serie) ON DELETE CASCADE
);

CREATE TABLE actua_serie
(
   id_actor                NUMBER,
   id_serie                NUMBER,
   personaje               VARCHAR(80),
   PRIMARY KEY(id_actor, id_serie, personaje),
   FOREIGN KEY(id_actor) REFERENCES trabajador(id_trabajador) ON DELETE CASCADE,
   FOREIGN KEY(id_serie) REFERENCES serie(id_serie) ON DELETE CASCADE
);

CREATE TABLE capitulo
(
   id_serie                NUMBER,
   titulo                  VARCHAR(150),
   episodio                NUMBER,
   temporada               NUMBER,
   estreno                 NUMBER,
   PRIMARY KEY(id_serie, titulo),
   FOREIGN KEY(id_serie) REFERENCES serie(id_serie) ON DELETE CASCADE
);
```

PARTE 2: INTRODUCCIÓN DE DATOS Y EJECUCIÓN DE CONSULTAS

2.1. Población de la base de datos

Para poblar esta base de datos se ha usado la tabla *datosdb.datospeliculas* que se encuentra en el servidor de Oracle realizando inserciones con consultas sobre esta tabla. El código con el que se pobló la base de datos se encuentra en el fichero *poblar.sql*.

En primer lugar, se han poblado las tablas de trabajadores y de películas que no dependen de ninguna otra tabla.

Después se han poblado las tablas que dependen de las películas realizando las intersecciones necesarias entre la tabla *datosdb.datospeliculas* y las tablas de películas y trabajadores: *genero_pelicula*, *pelis_relacionadas*, *actua_peli* y *trabaja_peli*.

Finalmente, se ha poblado la tabla de series y todas las que dependen de ella realizando las intersecciones necesarias entre la tabla *datosdb.datospeliculas* y las tablas de series y trabajadores: *genero_serie*, *capitulo*, *trabaja_serie* y *actua_serie*.

Algunos de los puntos sobre la población que merece la pena destacar son:

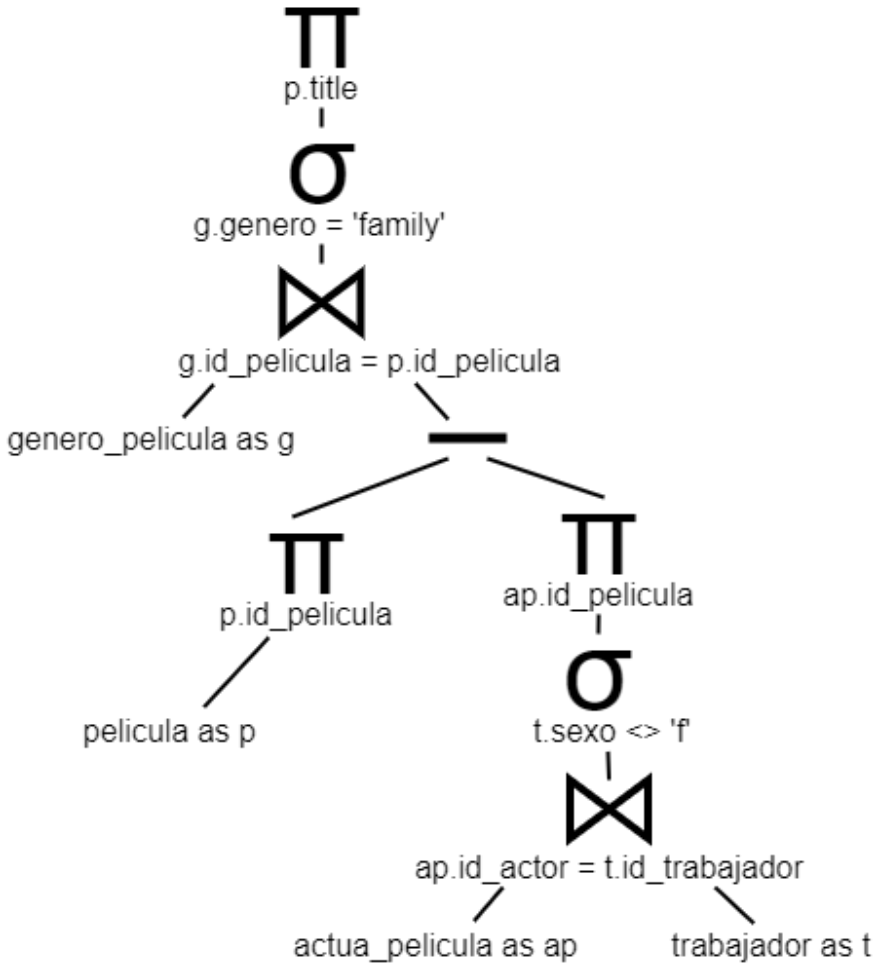
- Se ha usado el comando REGEXP_SUBSTR para separar el año final de producción y el comando TO_NUMBER para convertirlo a un número.
- Para la población de la tabla *pelis_relacionadas* se han guardado únicamente las películas cuyo valor de *link* era “version of” y “remake of” para remakes, “follows” para secuelas y “followed by” para precuelas. Además, en la columna *nueva* se ha guardado siempre la película de año de estreno más reciente de la relación, así se ha evitado guardar duplicados de las relaciones entre películas, es decir, que dos películas estén guardadas dos veces intercambiando sus valores en las columnas *nueva* y *original*.
- En las tablas que guardan las relaciones entre actores y obras se han guardado los actores que no interpreten un personaje, pero que actúan en una obra, poniendo el campo de personaje con valor “UNKNOWN”, ya que al formar el atributo *personaje* parte de la clave primaria no podía ser nulo.
- En las tablas de intersección entre obras y trabajadores no se han guardado los roles de actor y actriz. Si se quiere saber si un actor actúa en una obra deben consultarse las tablas *actua_peli* o *actua_serie*.

2.2. Consultas SQL

Consulta 1

Títulos de las películas de género familiar que sólo han sido interpretadas por actrices.

```
SELECT p.titulo
FROM pelicula p
JOIN genero_pelicula g ON p.id_pelicula = g.id_pelicula
WHERE g.genero = 'family'
  AND p.id_pelicula NOT IN (
    SELECT ap.id_pelicula
    FROM actua_pelicula ap
    JOIN trabajador t ON ap.id_actor = t.id_trabajador
    WHERE t.sexo <> 'f'
  );
```



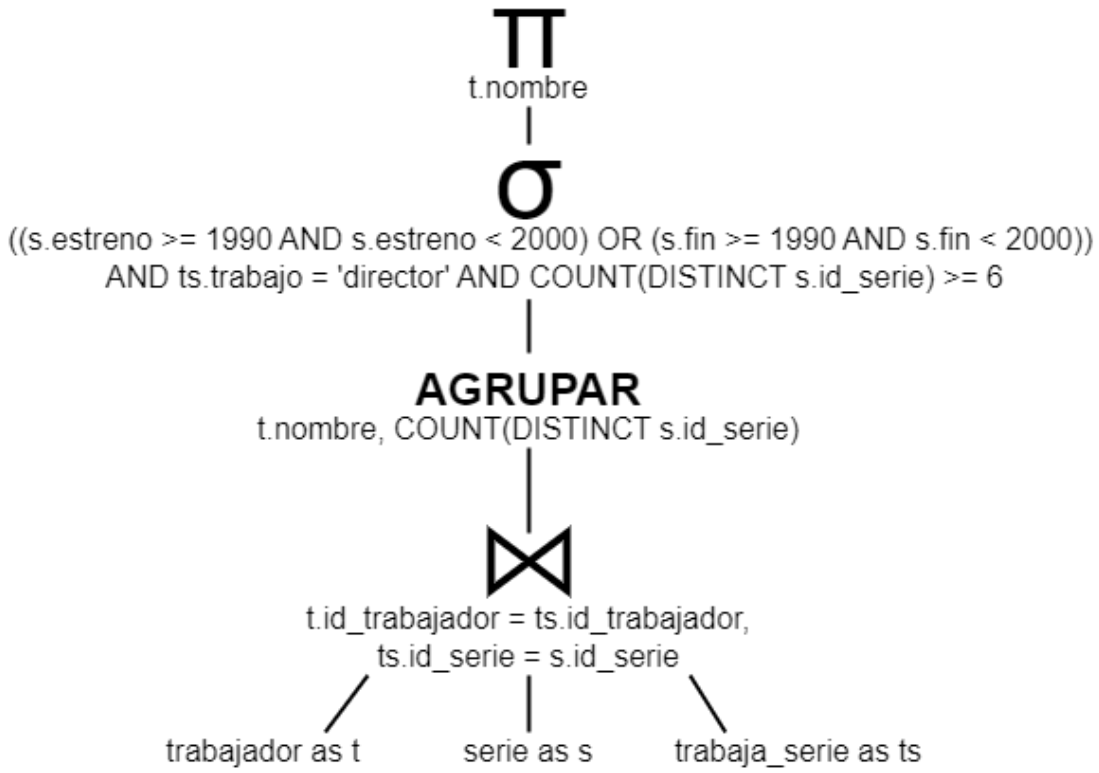
Resultado obtenido

```
TITLE
-------------------------
The Language You Cry In
```

Consulta 2

Nombre de los directores que han dirigido al menos seis series distintas en la década de los 90.

```
SELECT t.nombre
FROM trabajador t
JOIN trabaja_serie ts ON t.id_trabajador = ts.id_trabajador
JOIN serie s ON ts.id_serie = s.id_serie
WHERE  ((s.estreno >= 1990 AND s.estreno < 2000) OR (s.fin >= 1990 AND s.fin < 2000))
      AND ts.trabajo = 'director'
GROUP BY t.nombre
HAVING COUNT(DISTINCT s.id_serie) >= 6;
```



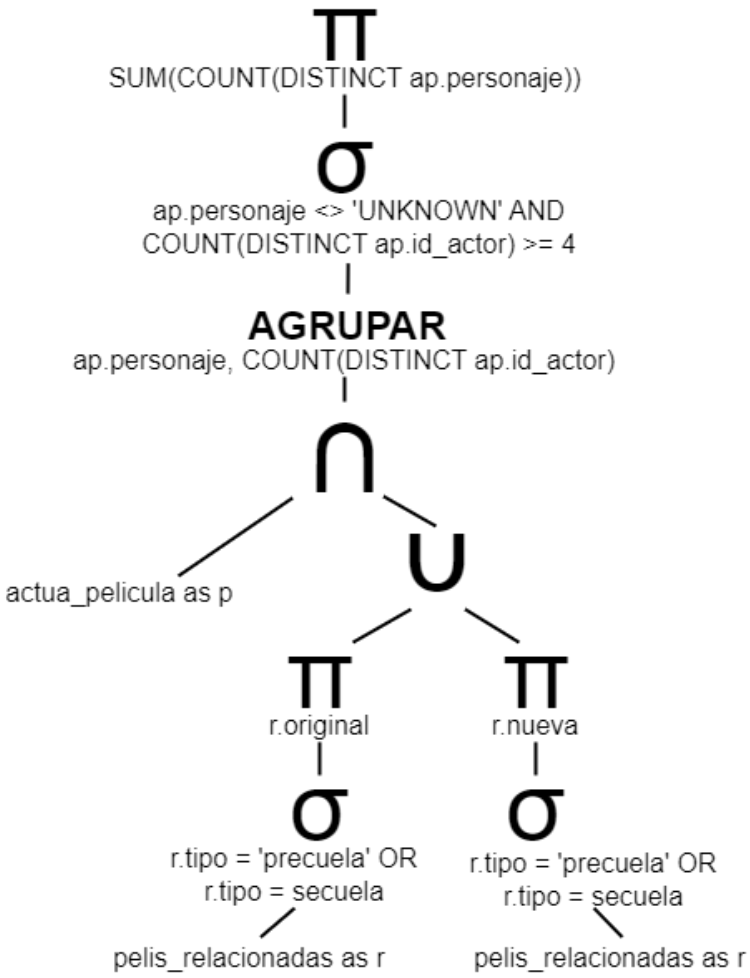
Resultado obtenido

```
NOMBRE
----------------------------------------
Garcia-Loureda, Ruben
Alvarez, Juan Luis
```

Consulta 3

Número de personajes distintos que han sido interpretados por al menos cuatro actores o actrices distintos y que aparecen en películas de alguna saga.

```
SELECT SUM(COUNT(DISTINCT ap.personaje)) as num_personajes
FROM actua_pelicula ap
WHERE ap.personaje <> 'UNKNOWN' AND ap.id_pelicula IN (
   SELECT DISTINCT r.original
   FROM pelis_relacionadas r
   WHERE r.tipo = 'precuela' OR r.tipo = 'secuela'
   UNION
   SELECT DISTINCT r.nueva
   FROM pelis_relacionadas r
   WHERE r.tipo = 'precuela' OR r.tipo = 'secuela'
)
GROUP BY ap.personaje
HAVING COUNT(DISTINCT ap.id_actor) >= 4;
```



Resultado obtenido

```
NUM_PERSONAJES
--------------
25
```

PARTE 3: DISEÑO FÍSICO

3.1. Rendimiento de las consultas

Consulta 1

Esta consulta tiene un coste N^2M siendo N el número de películas y M el número medio de actores por película, pues recorre la tabla de películas buscando aquellas que son de género familiar y después para cada una de ellas comprueba si se encuentra entre las películas que tienen solo actrices femeninas. La nueva consulta mejorada se encuentra en el archivo *consulta1_mejorada.sql*.

Coste en CPU inicial: 712

Mejora

El mayor problema de rendimiento encontrado son las intersecciones, no hay ningún índice que se pueda añadir que mejore el rendimiento. Se ha optado por desnormalizar la tabla *actua_pelicula* y se ha creado la tabla *actua_pelicula2* donde se incluirá también el género de los actores.

Coste en CPU tras la mejora: 234

```
CREATE TABLE actua_pelicula2
(
   id_actor              NUMBER,
   id_pelicula           NUMBER,
   personaje             VARCHAR(80),
   sexo                  VARCHAR(1),
   PRIMARY KEY(id_actor, id_pelicula, personaje),
   FOREIGN KEY(id_actor) REFERENCES trabajador(id_trabajador) ON DELETE CASCADE,
   FOREIGN KEY(id_pelicula) REFERENCES pelicula(id_pelicula) ON DELETE CASCADE
);

INSERT INTO actua_pelicula2(id_pelicula, id_actor, sexo, personaje)
SELECT DISTINCT p.id_pelicula, t.id_trabajador, d.gender, CASE WHEN d.role_name IS NULL
THEN 'UNKNOWN'
                                    ELSE d.role_name END
FROM pelicula p, trabajador t, datosdb.datospeliculas d
WHERE p.titulo = d.title AND p.estreno = d.production_year AND t.nombre = d.name
      AND (t.sexo = d.gender OR t.sexo IS NULL) AND d.kind = 'movie' AND
      (role = 'actor' OR role = 'actress');

SELECT p.titulo
FROM pelicula p
JOIN genero_pelicula g ON p.id_pelicula = g.id_pelicula
WHERE g.genero = 'family'
   AND p.id_pelicula NOT IN (
      SELECT ap.id_pelicula
      FROM actua_pelicula2 ap
      JOIN trabajador t ON ap.id_actor = t.id_trabajador
      WHERE ap.sexo <> 'f'
   );
```

Consulta 2

Esta consulta tiene un coste lineal en el número de series. Los problemas de rendimiento de esta consulta son principalmente la intersección con la tabla *trabaja_serie* y la intersección de esta con la tabla *trabajador*. Las mejoras se pueden ver en el archivo *consulta2_mejorada.sql.*
Coste en CPU inicial: 482

Primera mejora

El único índice que ha conseguido mejorar el rendimiento es el índice bitmap en la columna *trabajo* de la tabla *trabaja_serie*. Aunque la mejora no es significativa.
Coste en CPU tras la primera mejora: 475

```
CREATE BITMAP INDEX trabajo_serie ON trabaja_serie(trabajo);
```

Segunda mejora

En esta segunda mejora se ha creado una vista materializada que guarda los identificadores de las series, los directores de estas y sus años de estreno y fin. La mejora en rendimiento es muy grande, aunque habría que calcular el coste que tiene mantener esta vista actualizada para compararlo con el tiempo y recursos ahorrados al realizar consultas empleando esta vista.
Coste en CPU tras la segunda mejora: 24

```
CREATE MATERIALIZED VIEW director_serie AS
SELECT s.id_serie, estreno, titulo, fin, t.nombre AS director
FROM serie s
JOIN trabaja_serie ts ON s.id_serie = ts.id_serie
JOIN trabajador t ON ts.id_trabajador = t.id_trabajador
WHERE trabajo = 'director';

SELECT s.director
FROM director_serie s
WHERE (s.estreno >= 1990 AND s.estreno < 2000) OR (s.fin >= 1990 AND s.fin < 2000)
GROUP BY s.director
HAVING COUNT(DISTINCT s.id_serie) >= 6;
```

Consulta 3

El coste de esta consulta es NM siendo N el número de actores que actúan en películas y M el número de películas que se encuentran relacionadas con otras. La nueva consulta mejorada se encuentra en el archivo *consulta3_mejorada.sql*.

Coste en CPU inicial: 407

Mejora

Los mayores problemas de rendimiento son las intersecciones. No se ha podido encontrar ningún índice que no cree el SGDB por defecto que mejore la consulta. Se ha decidido crear una vista materializada que incluya los personajes de películas que pertenecen a sagas y el número de actores diferentes por los que han sido interpretados. Esta vista materializada supone una gran mejora del rendimiento de la consulta, pero habría que calcular los costes de mantenimiento de esta vista y compararlos con el tiempo ahorrado al realizar consultas empleándola para saber si es rentable.

Coste en CPU tras la mejora: 23

```
CREATE MATERIALIZED VIEW num_actores_personaje_saga AS
SELECT ap.personaje, COUNT(DISTINCT ap.id_actor) AS num_actores
FROM actua_pelicula ap
WHERE ap.personaje <> 'UNKNOWN' AND ap.id_pelicula IN (
   SELECT DISTINCT r.original
   FROM pelis_relacionadas r
   WHERE r.tipo = 'precuela' OR r.tipo = 'secuela'
   UNION
   SELECT DISTINCT r.nueva
   FROM pelis_relacionadas r
   WHERE r.tipo = 'precuela' OR r.tipo = 'secuela'
)
GROUP BY ap.personaje;

SELECT SUM(COUNT(DISTINCT personaje)) as num_personajes
FROM num_actores_personaje_saga
WHERE num_actores >= 4
GROUP BY personaje;
```

3.2. Creacción de triggers

Se han creado tres triggers que aseguran que se cumplen algunas de las restricciones del modelo descritas en el apartado del diseño del modelo entidad-relación.

Trigger 1

Este trigger asegura que se cumplen las restricciones impuestas sobre los valores de las fechas que guarda la tabla *serie*: no permite series anteriores al año 1895, no permite series que posteriores a 2023 y no permite que el año de fin sea menor que el año de estreno de la serie.

```
CREATE OR REPLACE TRIGGER valido_fechas_serie
BEFORE INSERT OR UPDATE ON serie
FOR EACH ROW
BEGIN
  IF :NEW.estreno <= 1895 THEN
    RAISE_APPLICATION_ERROR(-20001, 'El anyo de estreno debe ser mayor de 1895');
  END IF;

  IF :NEW.estreno > 2023 THEN
    RAISE_APPLICATION_ERROR(-20002, 'El anyo de estreno debe ser menor o igual que
2023');
  END IF;

  IF :NEW.fin > 2023 THEN
    RAISE_APPLICATION_ERROR(-20003, 'El anyo de fin debe ser menor o igual que 2023');
  END IF;

  IF :NEW.estreno > :NEW.fin THEN
    RAISE_APPLICATION_ERROR(-20004, 'El anyo de estreno debe ser menor o igual al anyo
de fin');
  END IF;

END;
/
```

Para comprobar el correcto funcionamiento del trigger se han probado las siguientes inserciones y visto que salieran los mensajes de error correspondientes:

```
INSERT INTO serie (id_serie, estreno, titulo, fin) VALUES (12345, 1890, 'A', 2023);
INSERT INTO serie (id_serie, estreno, titulo) VALUES (22222, 2040, 'B');
INSERT INTO serie (id_serie, estreno, titulo, fin) VALUES (54321, 2000, 'C', 2025);
INSERT INTO serie (id_serie, estreno, titulo, fin) VALUES (33333, 2012, 'D', 2011);
```

Trigger 2

Este trigger asegura que se cumplen las restricciones de la fecha de estreno de los capítulos y actúa de la siguiente manera: si se trata de insertar un capítulo con fecha de estreno anterior al estreno de la serie salta un error y si se trata de insertar un capítulo con fecha de estreno posterior al año de fin de una serie actualizará el valor de este.

```
CREATE OR REPLACE TRIGGER actualizar_fin_serie
BEFORE INSERT ON capitulo
FOR EACH ROW
DECLARE
  estreno_serie NUMBER;
BEGIN
  SELECT estreno INTO estreno_serie FROM serie WHERE id_serie = :NEW.id_serie;

  IF :NEW.estreno < estreno_serie THEN
    RAISE_APPLICATION_ERROR(-20005, 'La fecha de estreno del capitulo no puede ser menor
que la fecha de estreno de la serie');
  END IF;

  IF :NEW.estreno > estreno_serie THEN
    UPDATE serie SET fin = :NEW.estreno WHERE id_serie = :NEW.id_serie;
  END IF;
END;
/
```

Se han probado el siguiente código para probar el funcionamiento del trigger:

```
------------    EJEMPLO ACTUALIZACIÓN DE FIN    ------------------------
SELECT fin FROM serie WHERE id_serie = '1590'; -- el resultado es 19999
INSERT INTO capitulo (id_serie, titulo, episodio, temporada, estreno)
    VALUES (1590, 'Capítulo X', 1, 1, 2022);
SELECT fin FROM serie WHERE id_serie = '1590'; -- se actualiza a 2022
-------------------------------------------------------------------------------


------------      EJEMPLO DE ERROR      ----------------------------------
SELECT estreno FROM serie WHERE id_serie = '1590'; -- el resultado es 1997
INSERT INTO capitulo (id_serie, titulo, episodio, temporada, estreno)
    VALUES (1590, 'Capítulo X', 1, 1, 1990); -- sale mensaje de error
-------------------------------------------------------------------------------
```

Trigger 3

Este trigger comprueba que se almacenan nuevas relaciones entre películas correctamente: una película no puede relacionarse consigo misma y en una tupla la película en la columna *nueva* debe ser del mismo año o posterior a la almacenada en la columna *original*.

```
CREATE OR REPLACE TRIGGER comprobar_pelis_relacionadas
BEFORE INSERT ON pelis_relacionadas
FOR EACH ROW
DECLARE
   estreno_original NUMBER;
   estreno_nueva NUMBER;
BEGIN

   IF :NEW.original = :NEW.nueva THEN
      RAISE_APPLICATION_ERROR(-20006, 'Una pelicula no puede relacionarse consigo misma');
   END IF;

   SELECT estreno INTO estreno_original FROM pelicula WHERE id_pelicula = :NEW.original;
   SELECT estreno INTO estreno_nueva FROM pelicula WHERE id_pelicula = :NEW.nueva;

   IF estreno_original > estreno_nueva THEN
      RAISE_APPLICATION_ERROR(-20007, 'El anyo de estreno de la pelicula nueva debe ser
mayor o igual que el de la pelicula original');
   END IF;

END;
/
```

Para comprobar el funcionamiento del trigger se ha ejecutado el siguiente código:

```
----------    EJEMPLO FALLO RELACIONADA CONSIGO MISMA   ----------------------------
INSERT INTO pelis_relacionadas (original, nueva, tipo) VALUES (2, 2, 'precuela');
-------------------------------------------------------------------------------------------------------


---------------------    EJEMPLO FALLO FECHAS   ---------------------------------------------
SELECT estreno FROM pelicula WHERE id_pelicula = 69; -- el resultado es 1981
SELECT estreno FROM pelicula WHERE id_pelicula = '3668'; -- el resultado es 1983
INSERT INTO pelis_relacionadas (original, nueva, tipo) VALUES (3668, 69, 'precuela');
-------------------------------------------------------------------------------------------------------
```

ANEXO

Recuento de horas

Para la realización de este trabajo me he basado en gran medida en el modelo entidad-relación y el relacional que hice con mis compañeros para la primera entrega. Los ficheros de creación y población de tablas de la entrega anterior también me han permitido ir más rápido, así como haber hecho las consultas anteriormente. Por ello, para el recuento de horas dedicadas he contado también mis horas en las partes 1 y 2 de la entrega anterior que son 10 horas y 16’5 horas respectivamente.

Parte 1	Parte 2	Parte 3	Memoria	Total
2h + 10h	10h + 16’5h	12h	9h	59’5h

Sesiones

Solo incluyo las de la segunda entrega:

05/06/2023 Mañana – Correcciones al modelo entidad-relación y relacional 2h

05/06/2023 Tarde – Creación de tablas y poblar 3h

06/06/2023 Mañana – Poblar 5h

06/06/2023 Tarde – Consultas 2h

07/06/2023 Mañana – Diseño físico 5h

07/06/2023 Tarde – Diseño físico y triggers 6h

08/06/2023 Mañana – Memoria 5h

08/06/2023 Tarde – Memoria 5h