BASES DE DATOS

Curso 2023 - 2024

Fecha de entrega: 06/05/2024

Grupo: L1.02

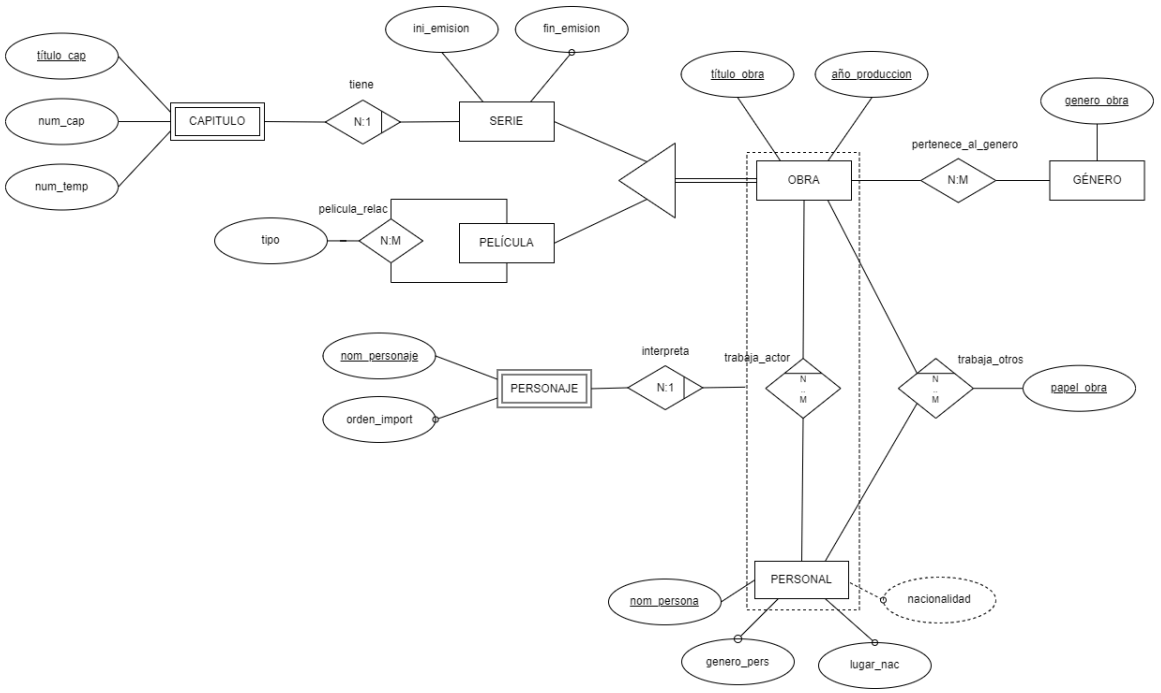
Irene Pascual Albericio 871627@unizar.es

Ariana Porroche Llorén 874055@unizar.es

Iván Vayad Díaz 874762@unizar.es

PARTE 1 - CREACIÓN DE UNA BASE DE DATOS

ESQUEMA E/R



El esquema E/R que hemos diseñado consta de 7 entidades: Obra, Pelicula, Serie, Capitulo, Genero, Personal y Personaje. En este modelo ya hemos tenido en cuenta algunas características de los datos, como la opcionalidad de ciertos atributos. En el enunciado se nos pide guardar el año de estreno de las películas y de las series, pero en los datos este aparece como año de producción, de ahí el nombre de nuestro atributo.
Las restricciones textuales de nuestro modelo son:

- la nacionalidad se calcula a partir de lugar_nac, quedándonos sólo con el país
- el genero_pers sólo puede ser ‘m’, ‘f’ o NULL
- el tipo de la película sólo puede ser ‘remake’, ‘precuela’ o ‘secuela’
- el año ini_emision debe ser anterior o igual al de fin_emision

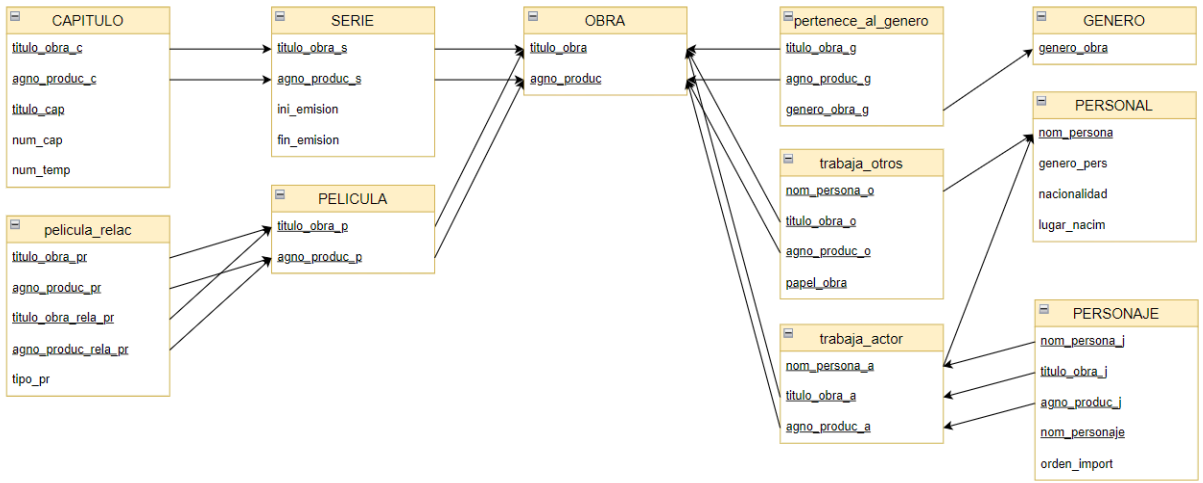
En cuanto a las soluciones alternativas, la principal duda que tuvimos fue con la relación de la entidad Personaje, ya que al principio pensamos relacionarla con la entidad Personal. Finalmente decidimos utilizar una agregación, ya que entendimos que un personaje lo interpreta una persona en una determinada obra, por lo que debía estar relacionado con ambas entidades. También creamos una especialización para representar las obras, de tal manera que tanto las series como las películas son obras de las que nos guardamos su título y su año de producción, pero cada entidad tiene sus propios atributos y/o relaciones.

Con respecto a los atributos de la entidad Personal, sólo nos hemos quedado con el género de la persona y el lugar de nacimiento, ya que el enunciado no nos pedía representar los demás datos ni los necesitábamos para las consultas, eran datos extras, pero añadimos el atributo de la nacionalidad.

ESQUEMA RELACIONAL

Primero, para poder representar el modelo relacional de nuestro E/R tuvimos que comprobar que estuviese tanto en 1FNBC como en 2FNBC y en 3FNBC. Para ello, comprobamos que no hubiese ningún atributo multievaluado. Después, comprobamos que ningún atributo no clave dependiese funcionalmente de parte de la clave (todos los atributos no clave dependen de toda la clave principal). Por último, comprobamos que ningún atributo no clave dependiera funcionalmente de un conjunto de atributos no clave, sin embargo, esto no lo cumple el atributo “nacionalidad” de la entidad Personal. Hemos decidido desnormalizarlo, es decir, mantenerlo como atributo derivado y hacer un trigger que siempre calcule la nacionalidad al insertar o actualizar una tupla, ya que es simplemente quedarnos con parte del campo del lugar de nacimiento. Esta decisión ha sido a nivel de optimización, para mejorar el tiempo de acceso a dicho dato.

Así pues, tras todas estas comprobaciones, obtuvimos el siguiente modelo relacional:



SENTENCIAS SQL DE CREACIÓN DE TABLAS

Como en nuestro modelo tenemos 7 entidades y 4 relaciones N:M, necesitamos crear 11 tablas: Obra, Pelicula, pelicula_relac, Serie, Capitulo, Genero, pertenece_al_genero, Personal, trabaja_otros, trabaja_actor y Personaje. Necesitamos crearlas en el mismo orden en el que están descritas abajo, para lo que utilizamos las siguientes sentencias SQL:

```
CREATE TABLE Obra (
      titulo_obra       VARCHAR(165),
      agno_produc       NUMBER(4),
      PRIMARY KEY (titulo_obra, agno_produc));
```

```
CREATE TABLE Pelicula (
      titulo_obra_p          VARCHAR(165),
      agno_produc_p          NUMBER(4),
      FOREIGN KEY (titulo_obra_p, agno_produc_p) REFERENCES
       Obra(titulo_obra, agno_produc),
      PRIMARY KEY (titulo_obra_p, agno_produc_p));
```

```
CREATE TABLE pelicula_relac (
      titulo_obra_pr         VARCHAR(165),
      agno_produc_pr               NUMBER(4),
      titulo_obra_rela_pr    VARCHAR(165),
      agno_produc_rela_pr          NUMBER(4),
      tipo_pr                      VARCHAR(15) NOT NULL CHECK (tipo_pr
IN ('secuela', 'precuela', 'remake')),
      FOREIGN KEY (titulo_obra_pr, agno_produc_pr) REFERENCES
       Pelicula(titulo_obra_p, agno_produc_p),
      FOREIGN KEY (titulo_obra_rela_pr, agno_produc_rela_pr)
       REFERENCES Pelicula(titulo_obra_p, agno_produc_p),
      PRIMARY KEY (titulo_obra_pr, agno_produc_pr,
       titulo_obra_rela_pr, agno_produc_rela_pr));
```

```
CREATE TABLE Serie (
      titulo_obra_s          VARCHAR(165),
      agno_produc_s          NUMBER(4),
      ini_emision       NUMBER(4),
      fin_emision       NUMBER(4),
      FOREIGN KEY (titulo_obra_s, agno_produc_s) REFERENCES
       Obra(titulo_obra, agno_produc),
      PRIMARY KEY (titulo_obra_s, agno_produc_s));
```

```
CREATE TABLE Capitulo (
      titulo_obra_c          VARCHAR(165),
      agno_produc_c          NUMBER(4),
      titulo_cap        VARCHAR(75),
      num_cap                NUMBER(2),
      num_temp          NUMBER(2),
```

```
      FOREIGN KEY (titulo_obra_c, agno_produc_c) REFERENCES
       Serie(titulo_obra_s, agno_produc_s),
      PRIMARY KEY (titulo_obra_c, agno_produc_c, titulo_cap));
```

```
CREATE TABLE Genero (
      genero_obra      VARCHAR(20) PRIMARY KEY);
```

```
CREATE TABLE pertenece_al_genero (
      titulo_obra_g          VARCHAR(165),
      agno_produc_g          NUMBER(4),
      genero_obra_g          VARCHAR(20),
      FOREIGN KEY (titulo_obra_g, agno_produc_g) REFERENCES
       Obra(titulo_obra, agno_produc),
      FOREIGN KEY (genero_obra_g) REFERENCES Genero(genero_obra),
      PRIMARY KEY (titulo_obra_g, agno_produc_g, genero_obra_g));
```

```
CREATE TABLE Personal (
      nom_persona      VARCHAR(50) PRIMARY KEY,
      genero_pers      VARCHAR(1) CHECK (genero_pers IN ('m', 'f',
'NULL')),
      lugar_nac        VARCHAR(110),
      nacionalidad           VARCHAR(50));
```

```
CREATE TABLE trabaja_otros (
      nom_persona_o          VARCHAR(50),
      titulo_obra_o          VARCHAR(165),
      agno_produc_o          NUMBER(4),
      papel_obra       VARCHAR(20),
      FOREIGN KEY (nom_persona_o) REFERENCES Personal(nom_persona),
      FOREIGN KEY (titulo_obra_o, agno_produc_o) REFERENCES
       Obra(titulo_obra, agno_produc),
      PRIMARY KEY (nom_persona_o, titulo_obra_o, agno_produc_o,
       papel_obra));
```

```
CREATE TABLE trabaja_actor (
      nom_persona_a          VARCHAR(50),
      titulo_obra_a          VARCHAR(165),
      agno_produc_a          NUMBER(4),
      FOREIGN KEY (nom_persona_a) REFERENCES Personal(nom_persona),
      FOREIGN KEY (titulo_obra_a, agno_produc_a) REFERENCES
       Obra(titulo_obra, agno_produc),
      PRIMARY KEY (nom_persona_a, titulo_obra_a, agno_produc_a));
```

```
CREATE TABLE Personaje (
      nom_persona_j          VARCHAR(50),
      titulo_obra_j          VARCHAR(165),
      agno_produc_j          NUMBER(4),
      nom_personaje          VARCHAR(80),
      orden_import           NUMBER(4),
      FOREIGN KEY (nom_persona_j, titulo_obra_j, agno_produc_j)
       REFERENCES trabaja_actor(nom_persona_a, titulo_obra_a,
```

```
    agno_produc_a),
  PRIMARY KEY (nom_persona_j, titulo_obra_j, agno_produc_j,
    nom_personaje));
```

PARTE 2 - INTRODUCCIÓN DE DATOS Y EJECUCIÓN DE CONSULTAS

POBLACIÓN DE TABLAS

Para poblar la base de datos, hemos optado por hacer un programa en c++ que lea el archivo .csv y cree los distintos archivos .sql para poblar las tablas.

En el .csv había datos que tenían el carácter '&', que por ser un carácter especial, no puede insertarse de igual manera que cualquier otro. Por ello, nuestro programa en c++ incorpora dichos datos como …' || '&' || '... Por otro lado, al haber incorporado el atributo derivado nacionalidad, hicimos que el programa, en caso de no ser nulo el lugar de nacimiento, se quedase únicamente con el país. Por ejemplo, si el lugar de nacimiento era 'Toledo, Castilla-La Mancha, Spain' entonces nacionalidad será 'Spain'. Además, algunos lugares de nacimiento incorporaban el carácter '', por lo que decidimos eliminarlo ya que utilizamos dicho carácter para delimitar un dato de tipo varchar. Por ejemplo 'L'Hospitalet de Llobregat, Barcelona, Catalonia, Spain' lo insertamos como 'LHospitalet de Llobregat, Barcelona, Catalonia, Spain'. Como carácter separador por defecto aparece el ";", pero dado que el contenido de algunos campos lo incluía, lo hemos cambiado por el carácter "=", el cual es utilizado por el programa en c++ para separar los campos.

Con respecto a la interpretación de los datos para añadirlos como 'remake', 'secuela' o 'precuela', decidimos utilizar las siguientes reglas:

- para 'remake', insertaremos aquellas tuplas cuya palabra clave en el .csv sea "version of", "remake of", "remade as", "spoofs" y "spoofed in"
- para 'secuela' y 'precuela' insertamos aquellas tuplas cuya palabra clave en el .csv sea "follows" o "follows by", pero siempre verificando que el año de estreno de la película (año de producción) sea posterior al año de estreno de la película en la que se basan

Dicho lo cual, subimos los datos (DatosPeliculas.csv) y el programa en c++ (lecturaPelis.cpp) a lab000, compilamos y ejecutamos el programa. Así obtenemos los distintos archivos de inserción en las tablas: insert_Obra.sql, insert_Pelicula.sql, insert_pelicula_relac.sql, insert_Serie.sql, insert_Capitulo.sql, insert_Genero.sql, insert_pertenece_al_genero.sql, inset_Personal.sql, insert_trabaja_otros.sql, insert_trabaja_actor.sql e insert_Personaje.sql.

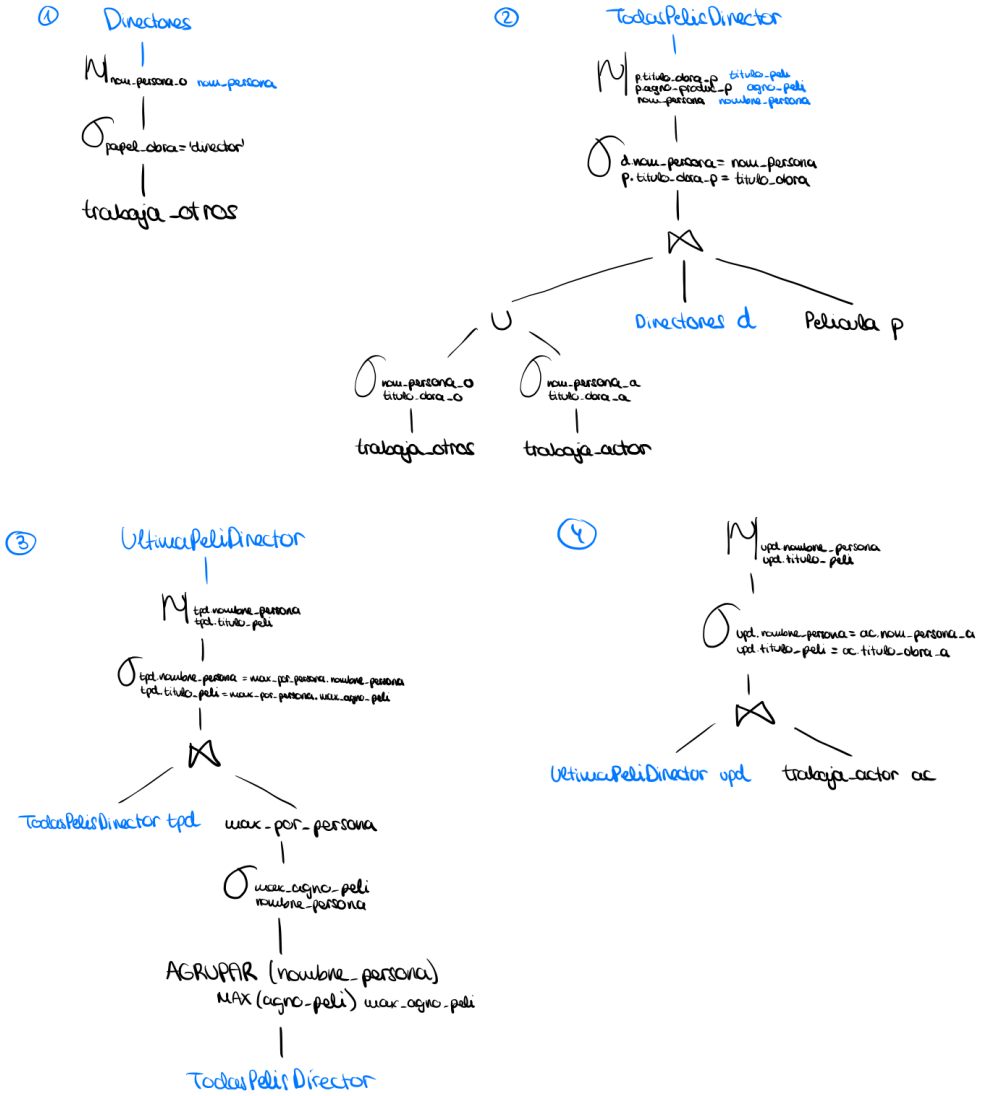
Para poblar la base de datos, antes de nada, fijamos la codificación en UTF-8 con el comando: "export NLS_LANG=.AL32UTF8" para las tildes principalmente. Ahora ejecutamos @insert_<nombre_tabla> en Oracle en el mismo orden indicado anteriormente y así poblamos todas las tablas correctamente. Para comprobar que se habían insertado todas las tuplas, comparamos el número de líneas de cada fichero .sql con el número de tuplas de cada tabla tras la inserción, el cual obtuvimos mediante la sentencia: "SELECT count(*) FROM <nombre_tabla>".

CONSULTAS SQL

CONSULTA 1 - DIRECTORES CUYA ÚLTIMA PELÍCULA EN LA QUE HAN PARTICIPADO HA SIDO COMO ACTOR O ACTRIZ

Para esta consulta, teníamos que listar los directores para los cuales la última película en la que han participado ha sido como actor o actriz.
Por ello hemos decidido seguir ciertos pasos e ir creando poco a poco vistas que nos lleven finalmente al resultado.

El árbol sintáctico de álgebra relacional que hemos seguido es el siguiente:



Lo primero de todo, hemos creado una vista para obtener todos los nombres de los directores de nuestra base de datos.

```
CREATE VIEW Directores AS
   SELECT DISTINCT nom_persona_o AS nombre_persona
   FROM trabaja_otros
   WHERE papel_obra = 'director';
```

A continuación, hemos usado esos nombres obtenidos en la vista Directores, para conseguir una tabla con los nombres de aquellas personas junto todas las obras en las que aparecen (teniendo en cuenta que pueden aparecer como actores u otros). Además, hemos aprovechado para seleccionar solo las obras que fuesen películas.

```
CREATE VIEW TodasPelisDirector AS
   SELECT p.titulo_obra_p AS titulo_peli, p.agno_produc_p AS
agno_peli, pers.nom_persona AS nombre_persona
  FROM (
    SELECT nom_persona_o AS nom_persona, titulo_obra_o AS titulo_obra
     FROM trabaja_otros
    UNION
    SELECT nom_persona_a AS nom_persona, titulo_obra_a AS titulo_obra
    FROM trabaja_actor
  ) pers
  JOIN Directores d ON d.nombre_persona = pers.nom_persona
  JOIN Pelicula p ON pers.titulo_obra = p.titulo_obra_p;
```

Una vez tenemos todas las películas de estas personas, vamos a coger solo los máximos años de producción para cada persona que aparece (por eso lo agrupamos según el nombre de la persona). Cabe destacar que si hay varias películas que coincidan en el máximo, se seleccionarán todas ellas.

```
CREATE VIEW UltimaPeliDirector AS
   SELECT tpd.nombre_persona AS nombre_persona, tpd.titulo_peli AS
titulo_peli
   FROM TodasPelisDirector tpd
   JOIN (
      SELECT MAX(agno_peli) AS max_agno_peli, nombre_persona
      FROM TodasPelisDirector
      GROUP BY nombre_persona
   ) max_por_persona ON tpd.nombre_persona =
max_por_persona.nombre_persona AND tpd.agno_peli =
max_por_persona.max_agno_peli;
```

Por último, procedemos a mostrar por la pantalla aquellas películas que aparezcan sus nombres en la tabla “trabaja_actor”, ya que eso significa que ha sido actor para esa película.

```
SELECT DISTINCT upd.nombre_persona AS nombre_persona, upd.titulo_peli
AS titulo_peli
FROM UltimaPeliDirector upd
JOIN trabaja_actor ac ON upd.nombre_persona = ac.nom_persona_a AND
upd.titulo_peli = ac.titulo_obra_a;
```

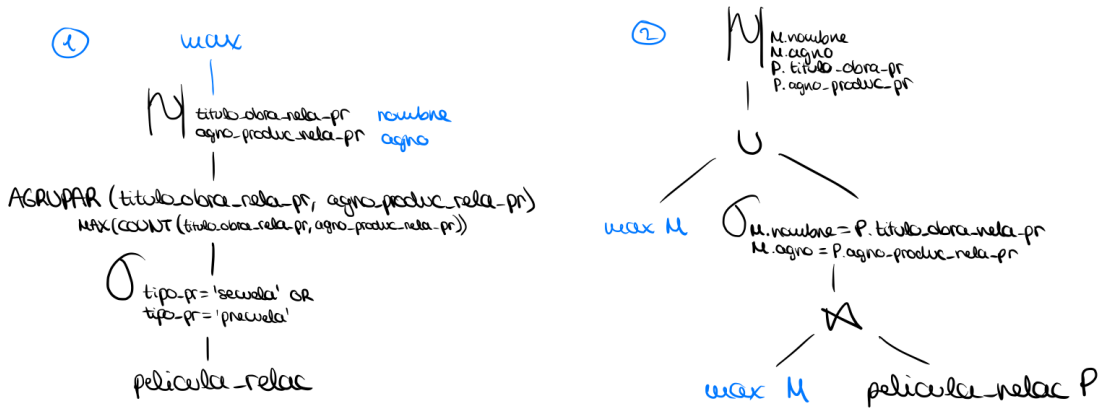
Al terminar la consulta, eliminamos todas las vistas realizadas para no ocupar ese espacio de nuestra base de datos.

Como resultado de esta consulta, hemos obtenido 356 tuplas distintas.

CONSULTA 2 - SAGA DE PELÍCULAS MÁS LARGA

Para esta consulta, teníamos que listar los títulos de las películas que componen la saga de películas más larga (en número de películas), incluyendo secuelas y precuelas.

El árbol sintáctico de álgebra relacional que hemos seguido es el siguiente:



Primero, hemos creado una vista donde obtenemos la película que más secuelas y precuelas tiene (su nombre y su año de producción).

```
CREATE VIEW max (nombre, agno) AS
   SELECT titulo_obra_rela_pr, agno_produc_rela_pr
   FROM pelicula_relac
   GROUP BY titulo_obra_rela_pr, agno_produc_rela_pr
   HAVING COUNT(*) = (
      SELECT MAX(num_veces)
      FROM (
         SELECT titulo_obra_rela_pr, agno_produc_rela_pr, COUNT(*) AS
num_veces
         FROM pelicula_relac
         WHERE tipo_pr='secuela' OR tipo_pr='precuela'
         GROUP BY titulo_obra_rela_pr, agno_produc_rela_pr
      )
   );
```

Por último, mostramos el nombre y el año de producción de todas las películas que componen la saga.

```
SELECT M.nombre, M.agno
FROM max M
UNION
SELECT P.titulo_obra_pr, P.agno_produc_pr
FROM max M, pelicula_relac P
WHERE M.nombre = P.titulo_obra_rela_pr AND M.agno =
P.agno_produc_rela_pr;
```

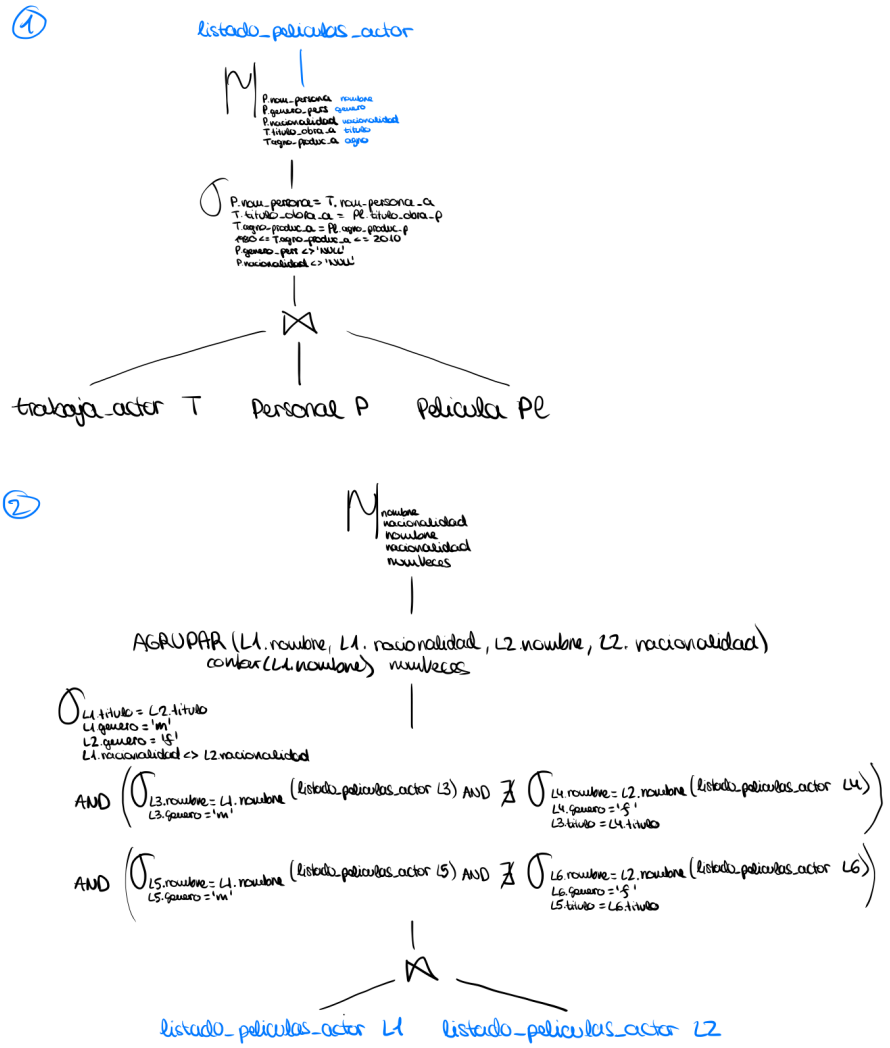
El resultado que obtenemos es el siguiente:

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

CONSULTA 3 - PAREJAS ACTOR-ACTRIZ DE DISTINTA NACIONALIDAD QUE SIEMPRE HAN TRABAJADO JUNTOS EN PELÍCULAS ENTRE 1980 Y 2010

Para esta consulta, teníamos que listar el nombre y la nacionalidad de las parejas actor-actriz de distinta nacionalidad que siempre han trabajado juntos en películas estrenadas entre 1980 y 2010, además de el número de veces que lo han hecho (en orden decreciente).

El árbol sintáctico que hemos seguido para la consulta es el siguiente:



Hemos creado 1 vista para facilitar la legibilidad de la consulta. Esta vista selecciona aquellas obras que sean películas y cuyo año de producción esté comprendido entre 1980 y 2010.

```
CREATE VIEW listado_peliculas_actor (nombre, genero, nacionalidad,
titulo, agno) AS
   SELECT P.nom_persona, P.genero_pers, P.nacionalidad,
T.titulo_obra_a, T.agno_produc_a
   FROM Personal P
```

```
   JOIN trabaja_actor T ON P.nom_persona = T.nom_persona_a
   JOIN Pelicula Pl ON T.titulo_obra_a = Pl.titulo_obra_p AND
T.agno_produc_a = Pl.agno_produc_p
   WHERE T.agno_produc_a BETWEEN 1980 AND 2010 AND P.genero_pers <>
'NULL' AND P.nacionalidad <> 'NULL';
```

Ya por último, seleccionamos aquellas parejas actores-actriz de distinta nacionalidad que siempre han trabajado juntos, tales que la primera persona sea de género ‘m’ (actor) y la segunda de género ‘f’ (actriz). Por ello, su listado de películas (todas las películas en las que han participado entre 1980 y 2010) debe ser igual.

```
SELECT DISTINCT L1.nombre, L1.nacionalidad, L2.nombre,
L2.nacionalidad, COUNT(L1.nombre) numVeces
FROM listado_peliculas_actor L1
JOIN listado_peliculas_actor L2 ON L1.titulo = L2.titulo
WHERE L1.genero = 'm' AND L2.genero = 'f' AND L1.nacionalidad <>
L2.nacionalidad
AND NOT EXISTS (
SELECT 1
FROM listado_peliculas_actor L3
WHERE L3.nombre = L1.nombre AND L3.genero = 'm'
   AND NOT EXISTS (
      SELECT 1
      FROM listado_peliculas_actor L4
      WHERE L4.nombre = L2.nombre AND L4.genero = 'f' AND L3.titulo =
L4.titulo
   )
)
AND NOT EXISTS (
   SELECT 1
   FROM listado_peliculas_actor L5
   WHERE L5.nombre = L2.nombre AND L5.genero = 'f'
   AND NOT EXISTS (
      SELECT 1
      FROM listado_peliculas_actor L6
      WHERE L6.nombre = L1.nombre AND L6.genero = 'm' AND L5.titulo =
L6.titulo
   )
)
GROUP BY (L1.nombre, L1.nacionalidad, L2.nombre, L2.nacionalidad)
ORDER BY numVeces DESC;
```

Como resultado de la consulta, obtenemos 1830 tuplas.

PARTE 3 - DISEÑO FÍSICO

PROBLEMAS DE RENDIMIENTO

Para poder comprobar los costes de CPU que conllevan realizar cada consulta, hemos utilizado el comando “SELECT PLAN FOR” para que al ejecutar a continuación este otro “SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY())”, apareciese el plan de ejecución desglosado.
Concretamente hacemos hincapié en el campo “COST (%CPU)” para verificar la estimación del costo de cada paso para el sistema.

Cabe destacar que debido a la falta de espacio en Oracle, en ocasiones hemos probado los respectivos índices y vistas materializadas para comprobar el coste de una consulta. No obstante, se debe ejecutar en conjunto el archivo “optimizaciones” antes de ejecutar las consultas (la “consulta1” y la “consulta3” ejecutarán las consultas optimizadas debido a que se ha modificado su estructura).

CONSULTA 1

Para mejorar el rendimiento de la primera consulta, en primer lugar decidimos unificar las vistas de “TodasPelisDirector” y “UltimaPeliDirector”, donde directamente devolvemos las últimas películas que había hecho cada persona que había sido director (esto último lo comprobamos con la vista “Directores”.
Sin embargo, al unificar estas dos vistas, observamos que se debía acceder 2 veces a la misma tabla (los nombres de todos los actores/actrices junto con sus obras, en unión de los nombres del resto de las personas junto con sus obras). Por tanto, aprovechamos para crear una vista materializada y poder acceder directamente sin tener que calcular todo el rato la misma tabla.

A continuación, hemos creado una vista materializada también de todos los nombres de los directores (ya que esta consulta en general, puede ser bastante recurrente para una base de datos que almacene datos de películas/series). Entonces, de momento hemos unificado en una sola vista y el select final para hacer la consulta.

Después, determinamos que sería útil crear varios índices para poder acceder más rápidamente a ciertas tuplas. Por ello creamos un índice de los años de las películas (para poder realizar la vista materializada de “pelis ordenadas” más rápida. Esta vista materializada nos servirá para poder seleccionar rápidamente las últimas películas para cada persona (haciendo uso de “max”) debido a que queda ordenado ascendentemente.
Ya para terminar hemos realizado algún otro índice como: ind_nom_persona (para ayudar a la vista materializada de los nombres de directores), ind_trabaja_otros_persona_obra_1, ind_trabaja_otros_persona_obra_2 y ind_trabaja_actor_persona_obra.
Finalmente, tras realizar estos cambios, los resultados obtenidos en cuanto al coste es el siguiente: primera consulta, 4363 ; segunda consulta, 253.

Acciones realizadas antes de ejecutar la consulta optimizada:

```
CREATE MATERIALIZED VIEW otrosYactores AS
SELECT nom_persona_o AS nom_persona, titulo_obra_o AS titulo_obra
FROM trabaja_otros
UNION
SELECT nom_persona_a AS nom_persona, titulo_obra_a AS titulo_obra
FROM trabaja_actor;

CREATE INDEX ind_pelis_agno ON Pelicula(agno_produc_p);

CREATE MATERIALIZED VIEW pelis_ordenadas REFRESH ON COMMIT AS
SELECT titulo_obra_p, agno_produc_p
FROM Pelicula
ORDER BY agno_produc_p ASC;

CREATE INDEX ind_pelis_ordenadas_titulo ON
pelis_ordenadas(titulo_obra_p);

CREATE INDEX ind_trabaja_otros_persona_obra_1 ON
trabaja_otros(nom_persona_o, titulo_obra_o);
CREATE INDEX ind_trabaja_otros_persona_obra_2 ON
trabaja_otros(papel_obra);
CREATE INDEX ind_trabaja_actor_persona_obra ON
trabaja_actor(nom_persona_a, titulo_obra_a);

CREATE UNIQUE INDEX ind_nom_persona ON Personal(nom_persona);

CREATE MATERIALIZED VIEW nombres_directores REFRESH ON COMMIT AS
SELECT nom_persona_o AS nombre_persona
FROM trabaja_otros
WHERE papel_obra = 'director';
```

El código utilizado para la nueva consulta es el siguiente:

```
CREATE VIEW Directores AS
   SELECT DISTINCT nom_persona_o AS nombre_persona
   FROM trabaja_otros
   WHERE papel_obra = 'director';

CREATE VIEW UltimaPeliDirector AS
  SELECT po.titulo_obra_p AS titulo_peli, po.agno_produc_p AS
agno_peli, pers.nom_persona AS nombre_persona
  FROM otrosYactores pers
  JOIN nombres_directores d ON d.nombre_persona = pers.nom_persona
  JOIN pelis_ordenadas po ON pers.titulo_obra = po.titulo_obra_p
  WHERE po.agno_produc_p = (
```

```
    SELECT MAX(po2.agno_produc_p)
    FROM pelis_ordenadas po2
    JOIN otrosYactores pers2 ON pers2.titulo_obra = po2.titulo_obra_p
    WHERE pers2.nom_persona = pers.nom_persona
  );

SELECT DISTINCT upd.nombre_persona AS nombre_persona, upd.titulo_peli
AS titulo_peli
FROM UltimaPeliDirector upd
JOIN trabaja_actor ac ON upd.nombre_persona = ac.nom_persona_a AND
upd.titulo_peli = ac.titulo_obra_a;
```

CONSULTA 2

Para poder realizar la optimización de la consulta 2, pensamos que sería muy útil crear el índice ind_pelicula_relac para poder realizar la búsqueda de los datos que nos interesan en película_relac más rápidamente.

```
CREATE INDEX ind_pelicula_relac ON
pelicula_relac(titulo_obra_rela_pr, agno_produc_rela_pr);
```

El resto (la consulta y la vista max) lo hemos dejado exactamente igual ya que hemos visto que añadiendo este índice nuestra consulta mejoraba significativamente.

```
CREATE VIEW max (nombre, agno) AS
    SELECT titulo_obra_rela_pr, agno_produc_rela_pr
    FROM pelicula_relac
    GROUP BY titulo_obra_rela_pr, agno_produc_rela_pr
    HAVING COUNT(*) = (
        SELECT MAX(num_veces)
        FROM (
            SELECT titulo_obra_rela_pr, agno_produc_rela_pr, COUNT(*)
AS num_veces
            FROM pelicula_relac
            WHERE tipo_pr='secuela' OR tipo_pr='precuela'
            GROUP BY titulo_obra_rela_pr, agno_produc_rela_pr
        )
    );
SELECT M.nombre, M.agno
FROM max M
UNION
SELECT P.titulo_obra_pr, P.agno_produc_pr
FROM max M, pelicula_relac P
WHERE M.nombre = P.titulo_obra_rela_pr AND M.agno =
P.agno_produc_rela_pr;
```

Tras añadir este índice, hemos conseguido mejorar el resultado de 10 a 6 de coste de CPU.

CONSULTA 3

Esta consulta la hemos optimizado gracias a realizar la consulta de distinta forma (hemos reordenado para conseguir obtener los mismos datos más eficientemente).

```
CREATE VIEW solo_peliculas (nombre, titulo, agno) AS
    SELECT DISTINCT T.nom_persona_a, T.titulo_obra_a, T.agno_produc_a
    FROM trabaja_actor T, pelis_ordenadas P
    WHERE T.titulo_obra_a = P.titulo_obra_p AND T.agno_produc_a =
P.agno_produc_p AND T.agno_produc_a BETWEEN 1980 AND 2010;

CREATE VIEW listado_peliculas_actor (nombre, genero, nacionalidad,
titulo, agno) AS
    SELECT S.nombre, P.genero_pers, P.nacionalidad, S.titulo, S.agno
    FROM solo_peliculas S, Personal P
    WHERE S.nombre = P.nom_persona AND P.genero_pers <> 'NULL' AND
P.nacionalidad <> 'NULL';

--CREATE VIEW consulta3 (nombre1, nac1, nombre2, nac2, numVeces) AS
SELECT DISTINCT L1.nombre, L1.nacionalidad, L2.nombre,
L2.nacionalidad, COUNT(L1.nombre) numVeces
FROM listado_peliculas_actor L1
JOIN listado_peliculas_actor L2 ON L1.titulo = L2.titulo
WHERE L1.genero = 'm' AND L2.genero = 'f' AND L1.nacionalidad <>
L2.nacionalidad
AND NOT EXISTS (
    SELECT 1
    FROM listado_peliculas_actor L3
    WHERE L3.nombre = L1.nombre AND L3.genero = 'm'
    AND NOT EXISTS (
        SELECT 1
        FROM listado_peliculas_actor L4
        WHERE L4.nombre = L2.nombre AND L4.genero = 'f' AND L3.titulo
= L4.titulo
    )
)
AND NOT EXISTS (
    SELECT 1
    FROM listado_peliculas_actor L5
    WHERE L5.nombre = L2.nombre AND L5.genero = 'f'
    AND NOT EXISTS (
        SELECT 1
        FROM listado_peliculas_actor L6
        WHERE L6.nombre = L1.nombre AND L6.genero = 'm' AND L5.titulo
= L6.titulo
    )
)
GROUP BY (L1.nombre, L1.nacionalidad, L2.nombre, L2.nacionalidad)
ORDER BY numVeces DESC;
```

Además hemos usado la vista materializada creada en la optimización de la consulta 1 y el índice para ordenar esos datos obtenidos mediante el año de producción (los ordena de mayor a menor).

```
CREATE INDEX ind_pelis_agno ON Pelicula(agno_produc_p);

CREATE MATERIALIZED VIEW pelis_ordenadas REFRESH ON COMMIT AS
SELECT titulo_obra_p, agno_produc_p
FROM Pelicula
ORDER BY agno_produc_p ASC;

CREATE INDEX ind_pelis_ordenadas_titulo ON
pelis_ordenadas(titulo_obra_p);
```

Gracias a estos cambios, hemos conseguido pasar de 964 como coste de CPU hasta 361.

TRIGGERS

TRIGGER 1

Hemos decidido crear este trigger para generar la nacionalidad (un atributo que hemos creado para facilitarnos otras consultas) ya que es dependiente del lugar de nacimiento.
Primero se comprueba si el lugar de nacimiento es nulo, y en caso de que no lo sea, pasaremos a buscar qué debemos poner en nacionalidad (que será el país). El dato del país puede estar al principio (si no se especifica nada más) o al final.
Cabe destacar que en ocasiones puede haber texto explicativo entre paréntesis al final del dato de lugar de nacimiento, es por ello que también contemplamos ese caso para eliminarlo.
Para hacer la búsqueda de la posición del país, consultamos si hay alguna coma en el dato (si no hay coma, estará directamente ahí el país) mediante el comando “INSTR”. Si hay coma, tendremos que buscar la última coma (para ello se utiliza el “-1” del comando “INSTR”) y extraer lo que hay a partir de esa coma (se pone también “+1” para eliminar el espacio en blanco que hay tras la coma) con el comando “SUBSTR”.
Lo que hemos extraído lo ponemos en el atributo de nacionalidad.
Una vez tenemos en el atributo de la nacionalidad el país, comprobamos si hay algún paréntesis con el comando “INSTR”, extraemos y guardamos de nuevo en el atributo de nacionalidad todo lo que hay antes del primer paréntesis (se hace “-2”, porque se quiere eliminar el paréntesis y el espacio en blanco que hay antes del paréntesis).

```
CREATE OR REPLACE TRIGGER crearNacionalidad
BEFORE INSERT OR UPDATE ON Personal
FOR EACH ROW
BEGIN
  --Verificar si el lugar de nacimiento no es NULL
  IF :NEW.lugar_nac IS NOT NULL THEN
    --Si el lugar de nacimiento no contiene comas,
    --asignar la cadena completa como la nacionalidad
      IF INSTR(:NEW.lugar_nac, ',') = 0 THEN
        :NEW.nacionalidad := :NEW.lugar_nac;
      ELSE
        --Si el lugar de nacimiento contiene comas,
        --extraer la nacionalidad después de la última coma (con el
-1)
        --y empezamos a guardar desde una posición más (+1) para
quitar
        --el espacio en blanco
        :NEW.nacionalidad := SUBSTR(:NEW.lugar_nac,
INSTR(:NEW.lugar_nac, ',', -1) + 1);
      END IF;
    END IF;
```

```
  IF INSTR(:NEW.nacionalidad, '(') > 0 THEN
    :NEW.nacionalidad := SUBSTR(:NEW.nacionalidad, 1,
INSTR(:NEW.nacionalidad, '(') -2);
  END IF;
END;
/
```

TRIGGER 2

En este trigger, ponemos en práctica la restricción textual impuesta en el modelo E/R de que el año de ini_emision debe ser menor o igual al de fin_emision

Es un trigger sencillo, donde debemos comprobar que antes de insertar una tupla o actualizar una tupla de la tabla Serie, que el valor ini_emision sea inferior o igual que el de fin_emision, únicamente en caso de que fin_emision no sea nulo. En esta situación, lanzamos un mensaje de error informando al usuario.

```
CREATE OR REPLACE TRIGGER trigger2
BEFORE INSERT OR UPDATE ON Serie
FOR EACH ROW
BEGIN
  IF :NEW.fin_emision <> 'NULL' THEN
    IF :NEW.ini_emision > :NEW.fin_emision THEN
      raise_application_error(-20000,'El año ini_emision debe ser
menor o igual que fin_emision');
    END IF;
  END IF;
END;
/
```

TRIGGER 3

Como último trigger, hemos creado uno para la tabla pelicula_relac. Lo que hace es mostrar un mensaje de error en caso de que quieras insertar una secuela o una precuela o un remake y las obras a las que haces referencia no sean películas, para ayudar al usuario a entender los errores en caso de una inserción fallida.

Para ello, contamos el número de apariciones de la película que queremos insertar en la tabla Pelicula, y si aparece un total de 0 veces, entonces no está y lanzamos el mensaje de error. Realizamos lo mismo con la película a la que hace referencia.

```
CREATE OR REPLACE TRIGGER trigger3
BEFORE INSERT OR UPDATE ON pelicula_relac
FOR EACH ROW
DECLARE
  contador INTEGER;
BEGIN
  SELECT COUNT(*) INTO contador FROM Pelicula WHERE
(:NEW.titulo_obra_pr = titulo_obra_p AND :NEW.agno_produc_pr =
agno_produc_p);
  IF contador = 0 THEN
    raise_application_error(-20000,'La película que quieres insertar
no está en el sistema');
  END IF;
  SELECT COUNT(*) INTO contador FROM Pelicula WHERE
(:NEW.titulo_obra_rela_pr = titulo_obra_p AND
:NEW.agno_produc_rela_pr = agno_produc_p);
  IF contador = 0 THEN
    raise_application_error(-20000,'La película que quieres insertar
no está en el sistema');
  END IF;
END;
/
```

PARTE FINAL

DATOS TRABAJO

Al crear nuestra base de datos, nos topamos con varios problemas que nos hicieron pensar y replantear algunas decisiones. Aquí detallamos los principales obstáculos que encontramos:

- **Formato del CSV:** Empezamos con un archivo csv que no encajaba bien con lo que necesitábamos. Lo modificamos para que fuera más manejable, asegurándonos de mantener los mismos datos.
- **Nuevo Atributo Nacionalidad:** Para hacer las consultas más fáciles, añadimos el atributo de nacionalidad. Esto nos ayudó a simplificar mucho las búsquedas.
- **Paso a modelo Relacional:** Al pasar al modelo relacional, vimos que si queríamos que todo estuviera normalizado, algunos atributos como la nacionalidad tendrían que ser su propia entidad. Esto iba a complicar demasiado las cosas sin mucha ganancia, así que decidimos dejarlo como estaba.
- **Vistas Materializadas:** Intentamos usar vistas materializadas para mejorar el rendimiento, pero algunas eran tan complejas que el sistema no las aceptaba, por lo que tuvimos que plantear de distinta forma en varias ocasiones nuestra manera de optimizarlas.
- **Espacio en Oracle:** El espacio limitado en Oracle nos dificulta poblar en ocasiones la base de datos correctamente o añadir otras funcionalidades (índices, vistas materializadas…).
- **Triggers y Checks:** Al principio pensábamos usar ciertos triggers, pero luego nos dimos cuenta que podíamos manejarlo mejor con checks en la creación de la base de datos.

Cada uno de estos puntos fue un reto, pero también una oportunidad para aprender y mejorar el manejo de nuestra base de datos. Al final, estas experiencias nos ayudaron a encontrar un equilibrio entre funcionalidad, eficiencia y facilidad de uso.

En cuanto a las horas dedicadas y el reparto del trabajo, ha sido bastante equitativo. Al principio, tanto el modelo entidad-relación como el relacional lo hicimos entre todos, discutiendo las posibles soluciones y mejorando ciertos aspectos del modelo conforme encontrábamos ciertos problemas. En cuanto a las consultas, cada uno nos centramos más en una consulta: Irene en la primera, Iván en la segunda y Ariana en la tercera, pero pusimos las soluciones en común y las mejoras las realizamos entre todos. Respecto a los triggers, seguimos las restricciones textuales impuestas por los modelos y desarrollamos los triggers en común.

Irene Pascual Albericio	36h
Ariana Porroche Llorén	38h
Iván Vayad Díaz	34,5h