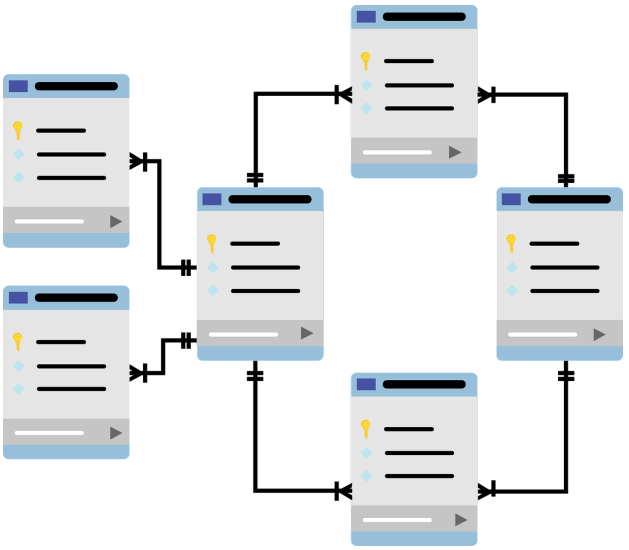
BASES DE DATOS: PRACTICA 2

CURSO 2022-2023

17 de Abril de 2023



Grupo: T9

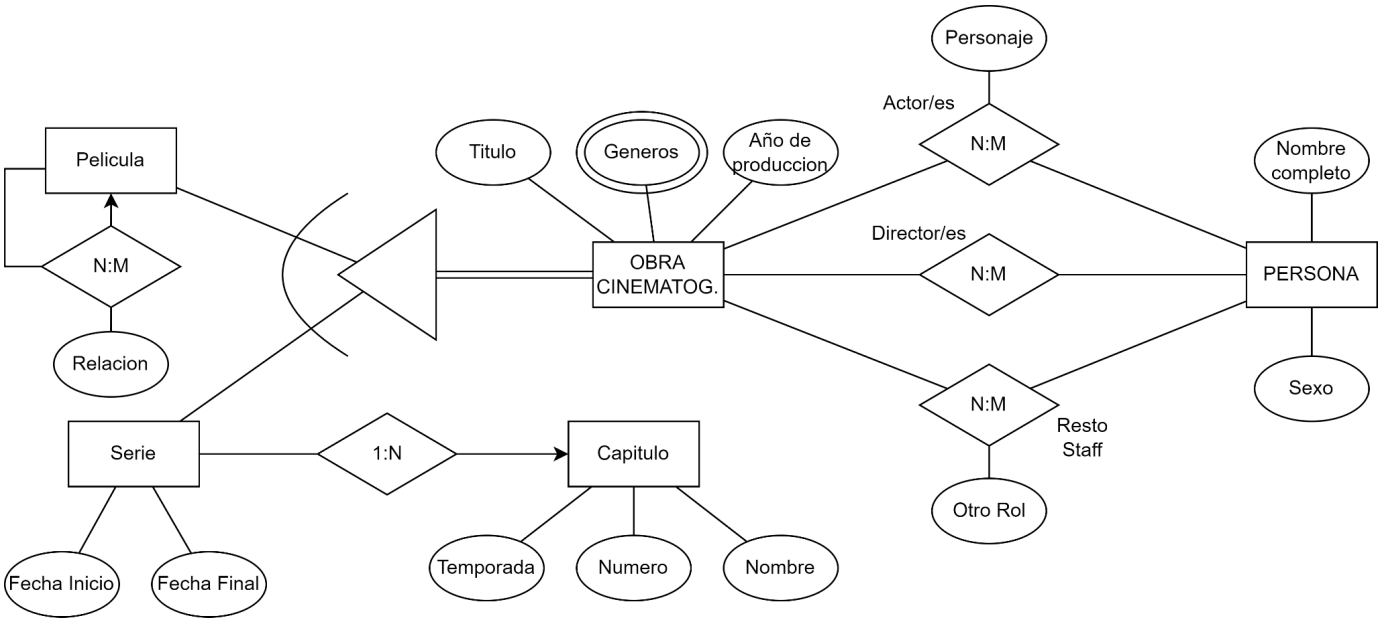
Alejandro Benedi Andrés - 843826@unizar.es

Javier Julve Yubero - 840710@unizar.es

Álvaro de Francisco Nievas - 838819@unizar.es

PARTE 1: Creación de una base de datos

Esquema entidad / relación.



Entidad *Serie*:

- Una serie tiene título único y periodo de emisión.
- Cada serie se relaciona con una o más temporadas

Entidad *Temporada*:

- Cada temporada tiene uno o más capítulos y pertenece a una serie.

Entidad *Obra cinematográfica*:

- El título no es atributo único (remakes) y tendrá uno o más géneros.
- Una obra cinematográfica forma parte de una serie, en cuyo caso se relaciona con una temporada; o es una película y no se relaciona con ninguna temporada.
- Una obra cinematográfica se relaciona con 0 o más obras cuando es remake, precuela o secuela.

Entidad *Persona*:

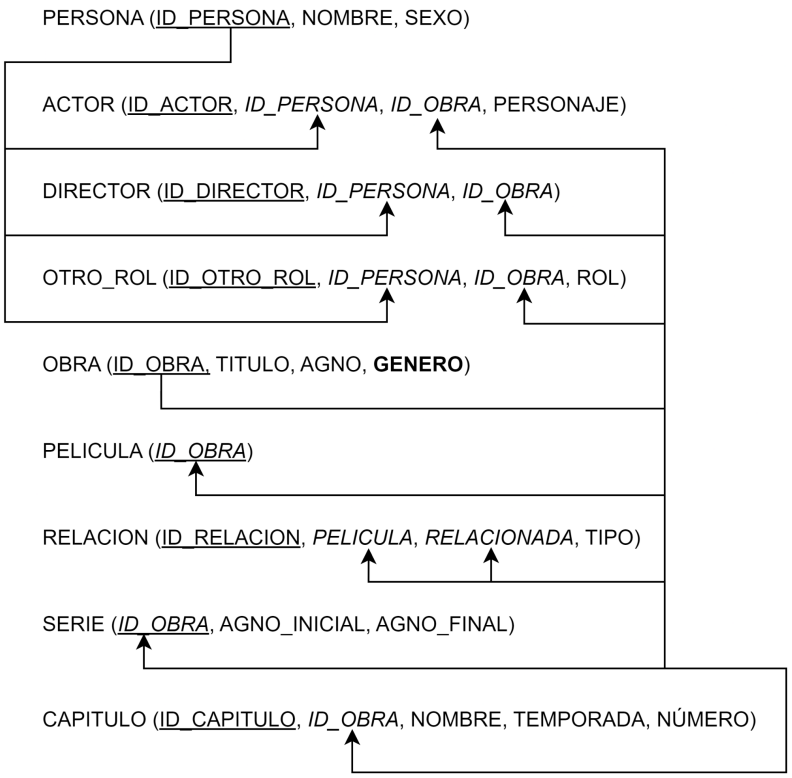
- Nombre completo no es identificador único.
- Cada persona participa en una o más obras cinematográficas, teniendo una o más roles distintos en esta.

Soluciones alternativas.

En el caso del género, si tuviera una mayor importancia podría ser una entidad independiente. Hemos decidido ponerlo como un atributo multi evaluable dado que no tiene datos asociados, como una lista de géneros similares, por ejemplo.
En vez de hacer 3 relaciones independientes, se podría haber hecho una única relación que englobe todas las participaciones. Sin embargo, hemos pensado que es mejor tener por separado los actores y los directores que son el personal más representativo.

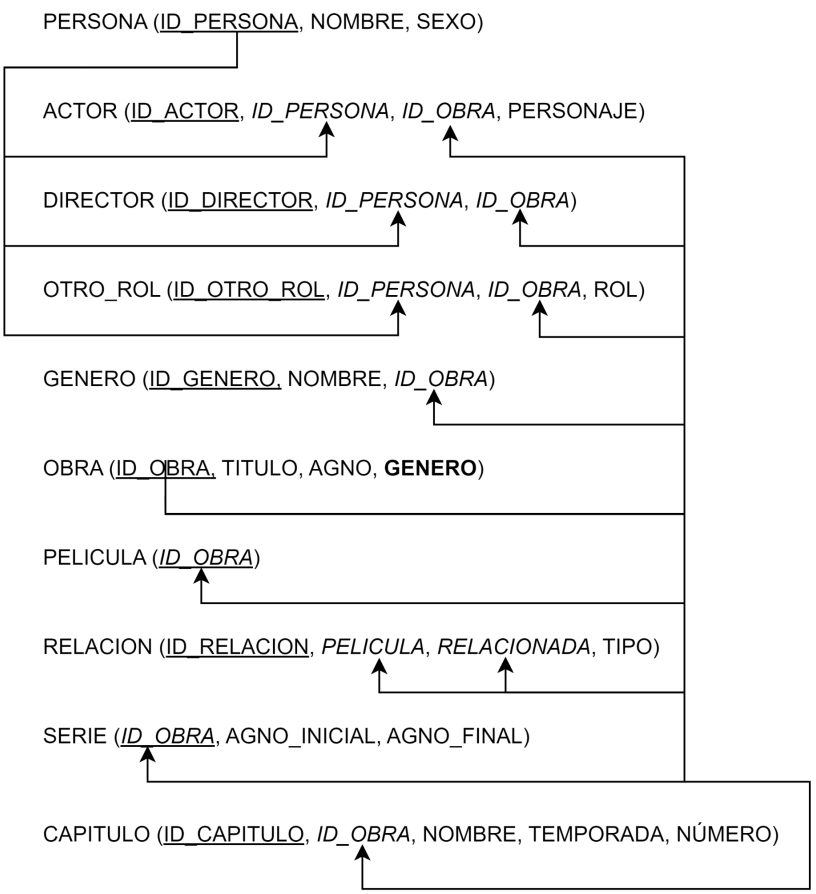
Esquema relacional.

Tras una traducción del Entidad-Relación obtenemos el siguiente modelo relacional.



Normalización.

El atributo "**Género"** de la tabla **Obra** es multivaluado sin número fijo de atributos. Por ello, es necesario crear una relación independiente que permita guardar los géneros que pueda tener una obra.
En cuanto a las claves primarias, no encontramos ningún atributo como clave candidata.
Ya que encontramos películas y series "remakes", el nombre de la obra no es único. "**Nombre"** en la tabla **Persona** tampoco podrá ser clave primaria, ya que dos personas pueden llamarse igual. Tampoco podemos utilizar "**Nombre"** de la nueva tabla **Género**, ya que al ser multivaluado tenemos que hacer referencia a la obra en la propia tabla y tendremos entradas diferentes con el mismo nombre.
Por tanto, utilizaremos claves artificiales como clave primaria para todas las tablas, con el objetivo de facilitar la implementación en SQL.
Todas las dependencias se basan en la clave primaria, por lo tanto el siguiente modelo relacional cumple con la forma normal de Boyce-Codd (FNBC).



El esquema ha sido modificado para cumplir con la primera forma normal (1FN). Sin embargo, como resultado de esta modificación, también cumple con la segunda forma normal (2FN) porque no hay dependencias funcionales que involucren ninguna parte de la clave. Además, el hecho de que no haya dependencias funcionales transitivas significa que cumple con la tercera forma normal (3FN).

Creación de tablas.

Creamos 10 tablas SQL. Además de los datos, hemos incluido CONSTRAINTs para comprobar propiedades de los datos en la inserción. Por ejemplo, comprobamos que el sexo sea **m**(asculino) o **f**(emenino); que los años sean mayores que **1895**, por ser el año en el que se grabó el primer vídeo; que las películas relacionadas tengan distinto identificador; o que las temporadas o capítulos sean mayores o iguales a 0.

```
CREATE TABLE PERSONA (
    ID_PERSONA  VARCHAR(10)     CONSTRAINT PK_PERSONA   PRIMARY KEY,
    NOMBRE      VARCHAR(100)    NOT NULL,
    SEXO        VARCHAR(1),
    CONSTRAINT CHK_PERSONA_SEXO CHECK (SEXO IN ('m', 'f')));

CREATE TABLE OBRA (
    ID_OBRA     VARCHAR(10)     CONSTRAINT PK_OBRA PRIMARY KEY,
    TITULO      VARCHAR(200)    NOT NULL,
    AGNO        NUMBER(4),
    CONSTRAINT CHK_OBRA_AGNO CHECK (AGNO >= 1928));

CREATE TABLE GENERO (
    NOMBRE      VARCHAR(100),
    ID_OBRA     VARCHAR(10),
    CONSTRAINT PK_GENERO PRIMARY KEY (NOMBRE, ID_OBRA),
    CONSTRAINT FK_GENERO_OBRA FOREIGN KEY (ID_OBRA) REFERENCES OBRA (ID_OBRA) ON
DELETE CASCADE);

CREATE TABLE PELICULA (
    ID_PELICULA     VARCHAR(5)     CONSTRAINT PK_PELICULA PRIMARY KEY,
    CONSTRAINT FK_PELICULA_OBRA FOREIGN KEY (ID_PELICULA) REFERENCES OBRA
(ID_OBRA) ON DELETE CASCADE);

CREATE TABLE SERIE (
    ID_SERIE        VARCHAR(5)     CONSTRAINT PK_SERIE PRIMARY KEY,
    AGNO_INICIAL    NUMBER(4),
    AGNO_FINAL      NUMBER(4),
    CONSTRAINT FK_SERIE_OBRA FOREIGN KEY (ID_SERIE) REFERENCES OBRA (ID_OBRA) ON
DELETE CASCADE,
    CONSTRAINT CHK_SERIE_AGNO_I CHECK (AGNO_INICIAL >= 1928),
    CONSTRAINT CHK_SERIE_AGNO_F CHECK (AGNO_FINAL >= 1928));

CREATE TABLE CAPITULO (
    ID_CAPITULO VARCHAR(5)     CONSTRAINT PK_CAPITULO PRIMARY KEY,
    ID_SERIE     VARCHAR(5)     NOT NULL,
    NOMBRE      VARCHAR(200),
    TEMPORADA   NUMBER(4),
    NUMERO      NUMBER(4),
```

```
    CONSTRAINT FK_CAPITULO_SERIE FOREIGN KEY (ID_SERIE) REFERENCES SERIE
(ID_SERIE) ON DELETE CASCADE,
    CONSTRAINT CHK_CAPITULO_TEMPORADA CHECK (TEMPORADA >= 0),
    CONSTRAINT CHK_CAPITULO_NUMERO CHECK (NUMERO >= 0));

CREATE TABLE RELACION (
    ID_PELICULA    VARCHAR(5),
    ID_RELACIONADA VARCHAR(5),
    TIPO        VARCHAR(100),
    CONSTRAINT PK_RELACION PRIMARY KEY (ID_PELICULA, ID_RELACIONADA, TIPO),
    CONSTRAINT FK_RELACION_PELICULA FOREIGN KEY (ID_PELICULA) REFERENCES PELICULA
(ID_PELICULA) ON DELETE CASCADE,
    CONSTRAINT FK_RELACION_RELACION FOREIGN KEY (ID_RELACIONADA) REFERENCES
PELICULA (ID_PELICULA) ON DELETE CASCADE,
    CONSTRAINT CK_RELACION_IDS_DIFERENTES CHECK (ID_PELICULA <> ID_RELACIONADA));

CREATE TABLE ACTOR (
    ID_ACTOR    VARCHAR(10)     CONSTRAINT PK_ACTOR_ID_ACTOR PRIMARY KEY,
    ID_PERSONA  VARCHAR(10)     NOT NULL,
    ID_OBRA     VARCHAR(10)     NOT NULL,
    PERSONAJE   VARCHAR(100),
    CONSTRAINT FK_ACTOR_PERSONA FOREIGN KEY (ID_PERSONA) REFERENCES PERSONA
(ID_PERSONA) ON DELETE CASCADE,
    CONSTRAINT FK_ACTOR_OBRA FOREIGN KEY (ID_OBRA) REFERENCES OBRA (ID_OBRA) ON
DELETE CASCADE);

CREATE TABLE DIRECTOR (
    ID_DIRECTOR  VARCHAR(10)    CONSTRAINT PK_DIRECTOR  PRIMARY KEY,
    ID_PERSONA  VARCHAR(10)     NOT NULL,
    ID_OBRA     VARCHAR(10)     NOT NULL,
    CONSTRAINT FK_DIRECTOR_PERSONA FOREIGN KEY (ID_PERSONA) REFERENCES PERSONA
(ID_PERSONA) ON DELETE CASCADE,
    CONSTRAINT FK_DIRECTOR_OBRA FOREIGN KEY (ID_OBRA) REFERENCES OBRA (ID_OBRA) ON
DELETE CASCADE);

CREATE TABLE OTRO_ROL (
    ID_ROL      VARCHAR(10)     CONSTRAINT PK_OTRO_ROL_ID_ROL PRIMARY KEY,
    ID_PERSONA  VARCHAR(10)     NOT NULL,
    ID_OBRA     VARCHAR(10)     NOT NULL,
    ROL         VARCHAR(100),
    CONSTRAINT FK_OTRO_ROL_PERSONA FOREIGN KEY (ID_PERSONA) REFERENCES PERSONA
(ID_PERSONA) ON DELETE CASCADE,
    CONSTRAINT FK_OTRO_ROL_OBRA FOREIGN KEY (ID_OBRA) REFERENCES OBRA (ID_OBRA) ON
DELETE CASCADE);
```

PARTE 2: Introducción de datos y ejecución de consultas

Población de la base de datos.

En el .csv de la práctica se encuentra toda la información necesaria para poblar, sin embargo toda ella está desordenada y se nos hacía imposible poblar toda la base de datos con únicamente ese .csv. Por tanto la solución fue hacer 10 fichero .csv diferentes (uno por tabla) en los cuales cada columna del .csv será un tipo de datos de la tabla SQL.

Una vez realizados los 10 ficheros se ha utilizado la aplicación SQL *Loader de ORacle (sqlldr2) para poblar toda la base de datos a partir de estos 10 csv. Es necesario implementar unos ficheros .ctl (ficheros (ficheros de control) uno para cada fichero .csv en los cuales se especifican el nombre del fichero .csv donde se va extraer los datos, el nombre del la tabla de destino, el carácter que separa los datos en el fichero csv (en este caso el carácter separador es ‘;’) y los datos que se van a introducir (columnas de la tabla).

Aunque este proceso ha costado menos que en la práctica anterior, la población de la base se nos ha alargado más de lo deseado debido a diferentes problemas que van a ser comentados a continuación:

1. El fichero .csv con todos los datos estaba muy mal distribuido y era muy poco explicativo. Por tanto antes de ponerse a crear los .csv para poblar era necesario revisar muy bien los datos para saber qué tipos de casos problemáticos nos podíamos encontrar. Esta parte fue la más compleja y larga de todo el proceso de población.

2. Nos encontramos casos en los cuales se repetía el título de la obra y lo que cambiaba era el año de producción, por lo tanto eran 2 obras diferentes y había que tener ambas en cuenta para poblar correctamente la tabla de obras. Para solucionarlo, en todas las relaciones entre entidades se han tenido que filtrar por nombre y por año de producción.

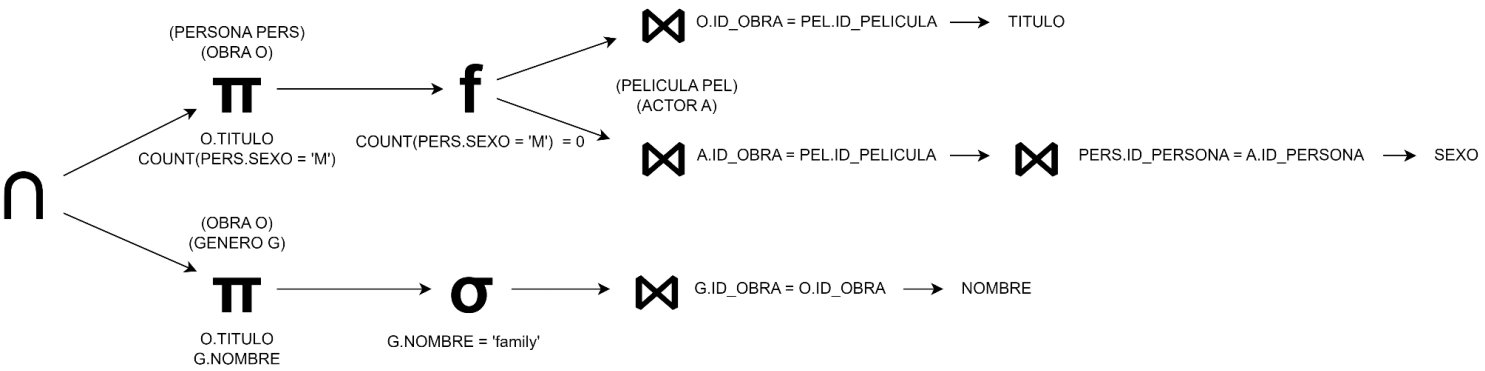
3. Nos encontramos también un caso en el cual se repetía el título de la obra y el año de producción, pero una obra era una película y la otra obra era una serie. Tan solo nos encontramos una caso por lo tanto fue muy sencillo corregirlo para las tablas “Serie” y “Obra” juntos con sus respectivas relaciones. Sin embargo, para poblar todas las tablas referentes al personal (“Director”, “Actor”, “Otro Rol”) se tuvo que filtrar por título de la obra, año de producción y tipo de obra (Serie o película)

En nuestro E/R se puede ver que hemos consideramos como obras a las series y a la películas únicamente (especialidad) . Los capítulos los hemos dejado apartados ya que consideramos que no tienen la suficiente importancia. Por este motivo, todos los actores, directores y el resto de personal que se encontraba dentro de cada capítulo de la serie, pasan a formar parte de la serie en general y cuya información se encontrará dentro de la tabla “Persona”.

Consultas SQL.

1ª consulta. Título de las películas de género familiar interpretadas por actrices

Árbol sintáctico.



Consulta SQL.

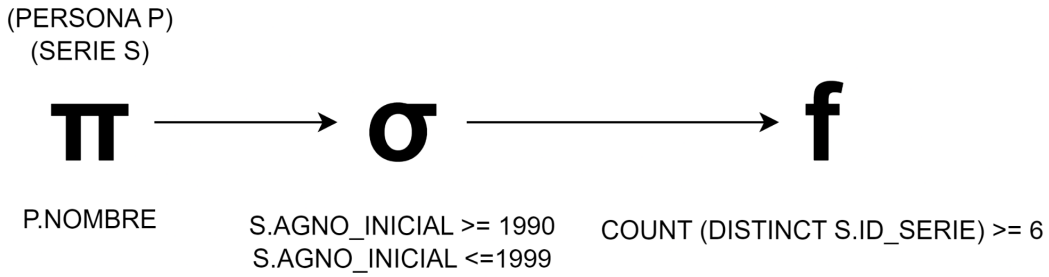
```
CREATE OR REPLACE VIEW v_n_actores AS
-- mostramos el titulo, genero y numero de actores de cada sexo
SELECT OBRA.TITULO, GENERO.NOMBRE,
   -- contamos el número de actores de cada sexo
   COUNT(CASE WHEN PERSONA.SEXO = 'm' THEN 1 END) AS NUM_ACT_MASC
FROM PELICULA
   -- seleccionamos los datos correspondientes al id para las tablas necesarias
   INNER JOIN OBRA ON PELICULA.ID_PELICULA = OBRA.ID_OBRA
   INNER JOIN ACTOR ON PELICULA.ID_PELICULA = ACTOR.ID_OBRA
   INNER JOIN PERSONA ON ACTOR.ID_PERSONA = PERSONA.ID_PERSONA
   INNER JOIN GENERO ON PELICULA.ID_PELICULA = GENERO.ID_OBRA
GROUP BY OBRA.TITULO, GENERO.NOMBRE;

SELECT TITULO FROM v_n_actores
-- ningún actor masculino y género familiar
WHERE NUM_ACT_MASC = 0 AND NOMBRE = 'family';
```

Resultado.

```
The Language You Cry In
```

2ª consulta. Directores que han dirigido 6 series distintas en los 90's

Árbol sintáctico.



Consulta SQL.

```
CREATE OR REPLACE VIEW v_directores_90 AS
-- mostramos
SELECT  PERSONA.NOMBRE, COUNT(DISTINCT SERIE.ID_SERIE) AS NUM_SERIES
FROM DIRECTOR
   INNER JOIN PERSONA ON DIRECTOR.ID_PERSONA = PERSONA.ID_PERSONA
   INNER JOIN SERIE ON DIRECTOR.ID_OBRA = SERIE.ID_SERIE
WHERE SERIE.AGNO_INICIAL >= 1990 AND SERIE.AGNO_INICIAL <= 1999
GROUP BY PERSONA.NOMBRE;

-- query final
SELECT NOMBRE FROM v_directores_90 WHERE NUM_SERIES >= 6;
```

Resultado.

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

3ª consulta. Nº personajes interpretados por +4 actores distintos en alguna saga.

Árbol sintáctico.

Seleccionamos el personaje y el id de la persona que interpreta el papel en la película. Agrupamos en una vista las películas que son precuelas o secuelas.
Sobre esta vista, seleccionamos el número de filas que obtenemos al agrupar los personajes y contando el número de actores que lo interpretan.

Consulta SQL.

```
CREATE OR REPLACE VIEW v_personajes AS
SELECT ACTOR.PERSONAJE, ACTOR.ID_PERSONA
FROM PELICULA
  INNER JOIN RELACION ON PELICULA.ID_PELICULA = RELACION.ID_PELICULA
  INNER JOIN ACTOR ON PELICULA.ID_PELICULA = ACTOR.ID_OBRA
WHERE (RELACION.TIPO = 'follows' OR RELACION.TIPO = 'followed by')
GROUP BY ACTOR.PERSONAJE, ACTOR.ID_PERSONA;

SELECT COUNT(COUNT(1)) AS NUM_ACTORES
FROM v_personajes
GROUP BY PERSONAJE
HAVING COUNT(DISTINCT ID_PERSONA) >= 4;
```

Resultado.

25

PARTE 3: Diseño Físico

Optimizaciones

Primera Consulta.

From.

```
| Id  | Operation               |Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |   147   (2)| 00:00:01 |
|*  1 |  FILTER                 |            |      |
|   2 |   HASH GROUP BY   |147   (2)| 00:00:01 |
|*  3 |    HASH JOIN            |146   (1)| 00:00:01 |
|*  4 |     HASH JOIN           |78   (2)| 00:00:01 |
|   5 |      NESTED LOOPS       |9   (0)| 00:00:01 |
|   6 |       NESTED LOOPS      |9   (0)| 00:00:01 |
|   7 |        TABLE ACCESS FULL| 9   (0)| 00:00:01 |
|*  8 |        INDEX UNIQUE SCAN|0   (0)| 00:00:01 |
|*  9 |       INDEX UNIQUE SCAN |0   (0)| 00:00:01 |
|  10 |      TABLE ACCESS FULL  |68   (0)| 00:00:01 |
|  11 |     TABLE ACCESS FULL   |68   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Inner Join.

```
| Id  | Operation               |Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT        |147   (2)| 00:00:01 |
|*  1 |  FILTER                 |          |      |
|   2 |   HASH GROUP BY         |147   (2)| 00:00:01 |
|*  3 |    HASH JOIN            |146   (1)| 00:00:01 |
|*  4 |     HASH JOIN           |78   (2)| 00:00:01 |
|   5 |      NESTED LOOPS       |9   (0)| 00:00:01 |
|   6 |       NESTED LOOPS      |9   (0)| 00:00:01 |
|   7 |        TABLE ACCESS FULL|9   (0)| 00:00:01 |
|*  8 |        INDEX UNIQUE SCAN|0   (0)| 00:00:01 |
|*  9 |       INDEX UNIQUE SCAN |0   (0)| 00:00:01 |
|  10 |      TABLE ACCESS FULL  |68   (0)| 00:00:01 |
|  11 |     TABLE ACCESS FULL   |68   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Segunda Consulta.

From.

```
| Id  | Operation                  |Cost (%CPU)|Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT           |423   (4)|00:00:01 |
|*  1 |  FILTER                    |              |         |
|   2 |   HASH GROUP BY            |423   (4)|00:00:01 |
|   3 |    VIEW                    |423   (4)|00:00:01 |
|   4 |     HASH GROUP BY          |423   (4)|00:00:01 |
|   5 |      MERGE JOIN CARTESIAN  |418   (3)|00:00:01 |
|*  6 |       HASH JOIN            |78   (0)|00:00:01 |
|*  7 |        HASH JOIN           |10   (0)|00:00:01 |
|*  8 |         TABLE ACCESS FULL  |3   (0)|00:00:01 |
|   9 |         TABLE ACCESS FULL  |7   (0)|00:00:01 |
|  10 |        TABLE ACCESS FULL   |68   (0)|00:00:01 |
|  11 |       BUFFER SORT          |355   (5)|00:00:01 |
|  12 |        INDEX FAST FULL SCAN|0   (0)|00:00:01 |
-------------------------------------------------------------------------------
```

Inner Join.

```
| Id  | Operation              |Cost (%CPU)| Time    |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |79   (2)| 00:00:01 |
|*  1 |  FILTER                |            |    |
|   2 |   HASH GROUP BY        |79   (2)| 00:00:01 |
|   3 |    VIEW                |79   (2)| 00:00:01 |
|   4 |     HASH GROUP BY      |79   (2)| 00:00:01 |
|*  5 |      HASH JOIN         |78   (0)| 00:00:01 |
|*  6 |       HASH JOIN        |10   (0)| 00:00:01 |
|*  7 |        TABLE ACCESS FULL    |3   (0)| 00:00:01 |
|   8 |        TABLE ACCESS FULL |7   (0)| 00:00:01 |
|   9 |       TABLE ACCESS FULL |68   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Tercera Consulta.

From.

```
| Id  | Operation                                    |Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT                             |71   (2)| 00:00:01 |
|   1 |  SORT AGGREGATE                              |71   (2)| 00:00:01 |
|*  2 |   FILTER                                     |                  |          |
|   3 |    HASH GROUP BY                             |71   (2)| 00:00:01 |
|   4 |     VIEW                                     |71   (2)| 00:00:01 |
|   5 |      HASH GROUP BY                           |71   (2)| 00:00:01 |
|   6 |       VIEW                                   |71   (2)| 00:00:01 |
```

```
|   7 |        HASH GROUP BY                          |71   (2)| 00:00:01 |
|*  8 |         HASH JOIN                             |70   (0)| 00:00:01 |
|   9 |          INLIST ITERATOR                      |          |          |
|  10 |           TABLE ACCESS BY INDEX ROWID BATCHED|2   (0)| 00:00:01 |
|* 11 |            INDEX RANGE SCAN                   |1   (0)| 00:00:01 |
|* 12 |          TABLE ACCESS FULL                    |68   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Inner Join.

```
| Id  | Operation                                     |Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
-----------------------------
|   0 | SELECT STATEMENT                              |71   (2)| 00:00:01 |
|   1 |  SORT AGGREGATE                               |71   (2)| 00:00:01 |
|*  2 |   FILTER                                      |                   |          |
|   3 |    HASH GROUP BY                              |71   (2)| 00:00:01 |
|   4 |     VIEW                                      |71   (2)| 00:00:01 |
|   5 |      HASH GROUP BY                            |71   (2)| 00:00:01 |
|   6 |       VIEW                                    |71   (2)| 00:00:01 |
|   7 |        HASH GROUP BY                          |71   (2)| 00:00:01 |
|*  8 |         HASH JOIN                             |70   (0)| 00:00:01 |
|   9 |          INLIST ITERATOR                      |          |          |
|  10 |           TABLE ACCESS BY INDEX ROWID BATCHED|2   (0)| 00:00:01 |
|* 11 |            INDEX RANGE SCAN                   |1   (0)| 00:00:01 |
|* 12 |          TABLE ACCESS FULL                    |68   (0)| 00:00:01 |
-------------------------------------------------------------------------------
```

Hemos sacado dos tablas de coste de ejecución de instrucción y coste de tiempo de cada instrucción, una con las consultas antiguas, las cuales se han realizado utilizando from y la otra que es la optimizada que se ha hecho mediante el uso de inner join. Para la primera y la tercera consulta no se ha notado ningún tipo de cambio con lo que concluimos que ambas maneras eran igual de óptimas en nuestro caso, sin embargo en la consulta número dos si que se ha apreciado un cambio sustancial, pues en los inner join hay menos coste y se ha ahorrado el uso de 3 operaciones las cuales estaban relacionadas a una ordenación por medio del merge y un sort, con lo que en este caso los inner join sí que han optimizado bastante las consultas, por lo que recomendamos siempre hacerlo de esta manera. Hemos comprobado también el coste usando views materializadas, pero al estar continuamente actualizando la tabla no hemos sacado ningún tipo de beneficio en ejecución con lo que hemos concluido que era mejor hacer uso de los inner join como ya se ha explicado.

Triggers.

En nuestra base de datos encontramos restricciones que Oracle no puede solucionar con "**CONSTRAINTS"** en la sentencia de creación de tablas. Algunas de estas son:

1. Comprobar que no hay 2 capitulos en la misma serie con el mismo número y misma temporada
2. Borrar persona si esta no actúa, dirige o no es personal (otro rol) de una obra
3. Comprueba si los datos que se quieren introducir en la tabla Actores ya existen

1. Comprobar que no hay 2 capitulos en la misma serie con el mismo número y misma temporada.

```
-- no puede haber un capítulo igual temporada y número
CREATE OR REPLACE TRIGGER TR_ADD_CAP
BEFORE INSERT ON CAPITULO
FOR EACH ROW
DECLARE
   existe NUMBER;
BEGIN
   -- Hace la comprobación
   SELECT COUNT(*) INTO existe FROM CAPITULO WHERE NUMERO = :NEW.NUMERO AND
TEMPORADA = :NEW.TEMPORADA AND ID_SERIE = :NEW.ID_SERIE;

  -- Si existe no se podrá insertar la tupla
   IF existe > 0 THEN
      RAISE_APPLICATION_ERROR(-20000, 'Ya existe un capítulo con ese número y temporada en la
obra');
   END IF;
END;
```

En la tabla "Capitulos" tenemos una clave primaria artificial para no tener una clave compuesta de 3 atributos (prácticamente todos los atributos de la tabla). Por ello no podemos comprobar que no haya 2 capítulos con el mismo número y misma temporada en una serie.
Para hacer esto basta con hacer un trigger que haga un conteo de todas las tuplas (filas) de la tabla CAPITULO comprobando que no se repita el numero del capitulo, la temporada y el <ID_SERIE>.
Si en alguna de las tuplas cumple la condición se puede decir que existe ya un capítulo y por lo tanto no se tiene que introducir esta nueva tupla con el capítulo.

2. Borrar persona si esta no actúa, dirige o no es personal (otro rol) de una obra.

```
CREATE OR REPLACE TRIGGER TR_ACTOR_DEL_PERSONA
AFTER DELETE ON ACTOR
FOR EACH ROW
DECLARE
  actor_count NUMBER;
  director_count NUMBER;
  otro_rol_count NUMBER;
BEGIN

   -- Comprueba si existen
   SELECT COUNT(*) INTO actor_count FROM ACTOR WHERE ID_PERSONA = :OLD.ID_PERSONA;
   SELECT COUNT(*) INTO director_count FROM DIRECTOR WHERE ID_PERSONA = :OLD.ID_PERSONA;
   SELECT COUNT(*) INTO otro_rol_count FROM OTRO_ROL WHERE ID_PERSONA = :OLD.ID_PERSONA;

   IF actor_count = 0 AND director_count = 0 AND otro_rol_count = 0 THEN
      DELETE FROM PERSONA WHERE ID_PERSONA = :OLD.ID_PERSONA;
   END IF;
END;
```

Tras eliminar la participación, en este caso tras eliminar un actor, se hace un conteo comparando el <ID_PERSONA> borrado con el <ID_PERSONA> del resto de las tuplas que se encuentran en las tablas “ACTOR”, “DIRECTOR” Y “OTRO_ROL”. Si el conteo de una de las tablas se mayor que 1 significa que esa persona aún participa en alguna obra y por lo tanto no se puede eliminar.

Este trigger se hace de igual forma con la tabla “DIRECTORES”y “OTRO_ROL”. De hecho si queremos eliminar por completo las personas que no participan en ninguna obra se tendría que implementar este mismo trigger con “DIRECTORES”y “OTRO_ROL”, como se ha comentado antes.

3. Comprueba si los datos que se quieren introducir en la tabla Actores ya existen

```
CREATE OR REPLACE TRIGGER triggerRepeticionPapel
BEFORE INSERT ON ACTOR
FOR EACH ROW
DECLARE
   existe NUMBER(1);
BEGIN
   -- Comprueba si existe
   SELECT COUNT(*) INTO existe
   FROM ACTOR
   WHERE ID_PERSONA = :NEW.ID_PERSONA AND ID_OBRA = :NEW.ID_OBRA AND PERSONAJE =
:NEW.PERSONAJE;
   IF ( existe > 0) THEN
      RAISE_APPLICATION_ERROR(-20000, 'Este actor ya desempeña este papel en esta obra');
   END IF;
END;
```

Este trigger comprueba si los datos que se van a introducir en la tabla actores, ya existen en la tabla. Para ello basta con contar el número de veces que una tupla vieja (fila que ya se encuentra introducida en la tabla) es igual a la nueva tupla que se quiere introducir, excepto el <id_actor> que no hay que tenerlo en cuenta. Si este conteo es mayor igual a 1 significa que los datos están repetidos y por lo tanto no se debe introducir esta nueva tupla.

Conclusiones

Horas dedicadas:

	ALEJANDRO BENEDI	ALVARO DE FRANCISCO	JAVIER JULVE
PARTE I	4 HORAS	5 HORAS Y 30 MINUTOS	4 HORAS
PARTE II	8 HORAS	4 HORAS	8 HORAS
PARTE III	2 HORAS	2 HORAS	2 HORAS Y 30 MINUTOS

División del trabajo:

Al dividirnos el trabajo, acordamos dividir equitativamente las tareas para hacer el mayor número de cosas en el mayor tiempo posible, es por ello que mientras unos hacían el esquema entidad relación, más adelante otro se encargaría más de la parte de poblar, o mientras uno creaba tablas, el otro iba haciendo sql de las consultas al igual que ayudarse a la hora de optimización. Si bien las tareas se han dividido equitativamente, todas han sido supervisadas por todos los miembros del equipo y además han habido casos donde todos hemos colaborado en una misma tarea ya sea aportando información o haciendo pruebas, con lo que hemos aprendido lo mismo pero destinando los tiempos a una mayor optimización del trabajo.

Dificultades:

Como tal en esta práctica solo hemos tenido una única dificultad, ya que al haber avanzado en teoría y con la experiencia de la anterior base de datos, esta se nos ha complicado menos, no obstante esa dificultad vino al principio del todo al no saber plantear la separación con herencia de series y películas en el modelo entidad relación, pues las especialidades fue algo que no llegamos a ver hasta una semana después de empezar, con lo que al no saber de su existencia ni su funcionamiento no sabíamos cómo plantear de forma precisa este esquema.