Universidad San Jorge

Escuela de Arquitectura y Tecnología

Grado en Ingeniería Informática

Base de datos de Películas

Autores de la práctica: Nicolas Bastida y Máximo Valenciano

Asignatura de la práctica: Sistemas de la información

Zaragoza, 17 de 04 de 2024

Índice

Elección de la base de datos

Se ha elegido una base de datos sobre elementos multimedia (Series y Películas). En esta base de datos se almacenará información sobre las películas (título de las películas, año de estreno, genero/s), así como la lista de personas involucradas en dicha película (lista de actores, director/es y demás personal implicado en la realización), guardando la información básica como nombre, apellidos y género.

Destacando el papel de cada persona y una breve descripción de los personajes que interpretan los actores.

De la misma manera se almacenará información sobra sagas de obras cinematográficas.

Estas sagas pueden ser remakes, secuelas o precuelas de las películas.

Además, se guardará información sobre las series de TV como el periodo de emisión.

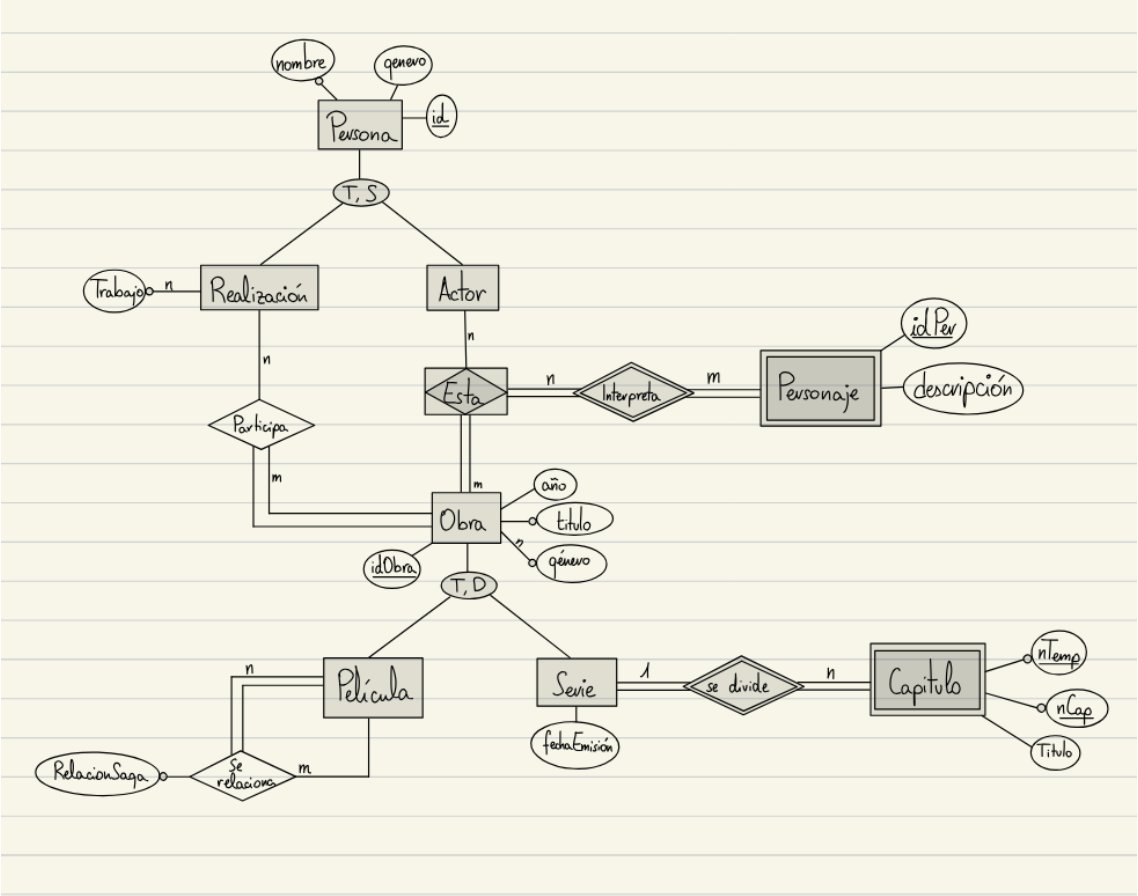
Sobre la información de las series, se pretende almacenar información sobre los capítulos de las series. Guardando la información del título del capítulo, la temporada a la que pertenece y el número de capitulo.

Esquema entidad relación E/R

Para la base de datos se ha pensado en el siguiente modelo entidad-relación. En este modelo se han puesto las entidades Realización, Actor, Obra, Personaje, Serie, Película, Capítulo y Persona.

Estas entidades se presentan de la siguiente manera: Una persona puede ser Realización, actor o ambas. Un actor está en una obra. Y un actor en una obra interpreta Personajes.

Por otra parte, una obra solo puede ser una película o una serie. Una serie tiene capítulos y una película puede o no estar en una saga.



En el modelo se han puesto dos generalizaciones, para Obra y para Persona. Ambas son totales, mientras que una es disjunta y la otra solapada.

Por otra parte, Personaje y Capítulo son entidades débiles, dado que sin la existencia de Un actor en una obra no puede existir un personaje y sin la existencia de una Serie no puede existir ningún capítulo.

También, se ha creado una entidad compuesta “Esta” para guardar la información de un actor que realiza una obra y así manejar la relación ternaria de una forma más esquematizada.

Además, una película se puede relacionar con otra(s) (relación reflexiva) siempre que estas tengan una relación de Saga. Por ejemplo, *Cazafantasmas* tiene una secuela y un remake.

Para identificar un capítulo se tendrá que obtener el id de la obra que se está buscando, el número de temporada y el número del capítulo de dicho capítulo.

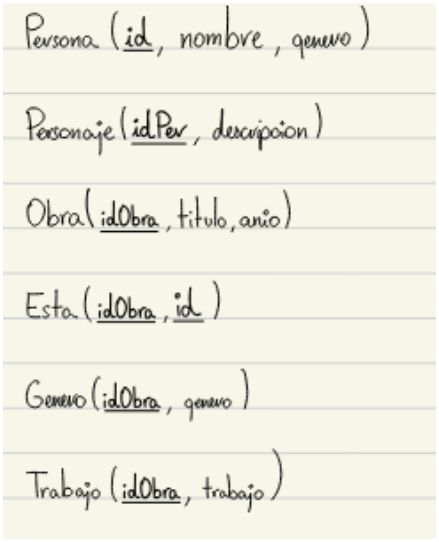
Restricciones

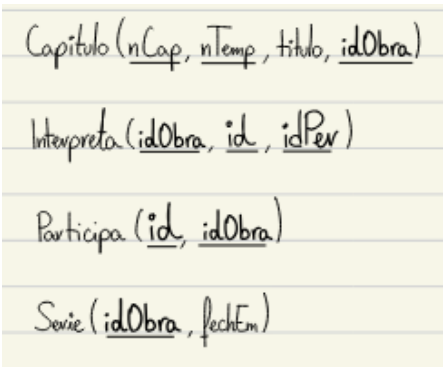
Este modelo presenta varias restricciones:

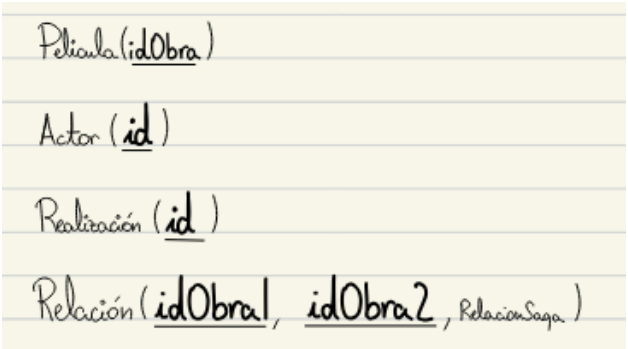
- Una obra no puede ser serie y película a la vez.
- Una obra no puede tener más de un género igual.
- Una película no puede tener capítulos, en su lugar pueden pertenecer a una saga (Remake, precuela, secuela).
- Una película no puede estar relacionada consigo mismo.
- Para una obra, una persona puede trabajar como realización, como actor o como ambas.
- Una persona no puede trabajar más de un trabajo igual para una obra.

Relacional

El modelo relacional resultante es el siguiente:







Para este modelo se han tenido en cuenta los atributos multievaluados. De forma que se han creado relaciones que gestionan los problemas de redundancia que generan los atributos multievaluados.

Dado que en la generalización “Obra” una obra puede ser una película o una serie, siendo que ambas guardan atributos distintos. Se han generado dos relaciones “Serie” y “Película” de forma que se gestionen los atributos pertinentes

Para una Serie se tratará el atributo fechEm, que representa la fecha de emisión de una serie. Mientras que para una Película, se puede o no gestionar el atributo RelacionSaga, siempre que haya una relación entre dos películas.

Normalización

La base de datos está en primera forma normal debido a que se han eliminado los atributos multievaluados (trabajo y género) en las tablas ‘Persona’ y ‘Obra’, haciendo una tabla independiente para cada uno.

Por otra parte, no hay atributos con dominios dispares en distintas tablas ni tampoco se encuentran atributos con nombres iguales en la misma relación para ninguna de las relaciones de la base de datos.

La base de datos se encuentra en segunda forma normal. Esto se debe a que no se encuentra ninguna dependencia parcial.

Por otra parte, la base de datos se encuentra en tercera forma normal debido a que no se encuentra ninguna dependencia transitiva.

Para finalizar la normalización, hay que verificar la forma normal de Boyce Codd.

Para ello se observan las relaciones creadas, verificando que se cumplan las tres primeras formas normales. Una vez se verifican, hay que observar que en las relaciones no haya ninguna superclave que aporte redundancia.

En este caso se cumple, ya que se han gestionado las relaciones N-M y las posibles redundancias generadas por superclaves.

Sentencias SQL para la creación de la base de datos

Las sentencias que se han usado para la base de datos son:

```sql
CREATE DATABASE multimedia;
```

Para crear la base de datos multimedia.

A partir de esta base de datos, las sentencias para crear todas y cada una de las tablas son las siguientes:

Tabla Persona:

```sql
CREATE TABLE IF NOT EXISTS Persona(
    idPersona INT PRIMARY KEY NOT NULL,
    nombrePersona VARCHAR(70) NOT NULL,
    genero VARCHAR(4)
);
```

Tabla Actor:

```sql
CREATE TABLE IF NOT EXISTS Actor(
    idActor INT PRIMARY KEY NOT NULL, FOREIGN KEY (idActor) REFERENCES Persona(idPersona)
);
```

Tabla Realización:

```sql
CREATE TABLE IF NOT EXISTS Realizacion(
    idRealizacion INT PRIMARY KEY NOT NULL, FOREIGN KEY (idRealizacion) REFERENCES Persona(idPersona)
);
```

Tabla Personaje:

```sql
CREATE TABLE IF NOT EXISTS Personaje(
    idPersonaje INT PRIMARY KEY NOT NULL,
    descripcion VARCHAR(70)
);
```

Tabla Obra:

```
CREATE TABLE IF NOT EXISTS Obra(
    idObra INT PRIMARY KEY NOT NULL,
    titulo VARCHAR(160) NOT NULL,
    fechaProduccion INT NOT NULL
);
```

Tabla Película:

```
CREATE TABLE IF NOT EXISTS Pelicula(
    idPelicula INT PRIMARY KEY NOT NULL, FOREIGN KEY (idPelicula) REFERENCES Obra(idObra)
);
```

Tabla Serie:

```
CREATE TABLE IF NOT EXISTS Serie(
    idSerie INT PRIMARY KEY NOT NULL,FOREIGN KEY (idSerie) REFERENCES Obra(idObra),
    fechaEmision VARCHAR(40) NOT NULL
);
```

Tabla Esta:

```
CREATE TABLE IF NOT EXISTS Esta(
    idObra INT NOT NULL, FOREIGN KEY(idObra) REFERENCES Obra(idObra),
    idActor INT NOT NULL, FOREIGN KEY(idActor) REFERENCES Actor(idActor),
    CONSTRAINT Pk_Esta PRIMARY KEY(idObra, idActor)
);
```

Tabla Genero:

```
CREATE TABLE IF NOT EXISTS Genero(
    idObra INT NOT NULL, FOREIGN KEY (idObra) REFERENCES Obra(idObra),
    genero VARCHAR(60) NOT NULL,
    CONSTRAINT Pk_Genero PRIMARY KEY(idObra, genero)
);
```

Tabla Trabajo:

```
CREATE TABLE IF NOT EXISTS Trabajo(
    idRealizacion INT NOT NULL, FOREIGN KEY(idRealizacion) REFERENCES Realizacion(idRealizacion),
    trabajo VARCHAR(60) NOT NULL,
    CONSTRAINT Pk_Trabajo PRIMARY KEY(idRealizacion, trabajo)
);
```

Tabla Capitulo:

```
CREATE TABLE IF NOT EXISTS Capitulo(
    numCap INT NOT NULL,
    numTemp INT NOT NULL,
    titulo VARCHAR(160),
    idSerie INT NOT NULL, FOREIGN KEY idSerie REFERENCES Serie(idSerie),
    CONSTRAINT Pk_Capitulo PRIMARY KEY(numCap, numTemp, idSerie)
);
```

Tabla Interpreta:

```
CREATE TABLE IF NOT EXISTS Interpreta(
    idActor INT NOT NULL, FOREIGN KEY(idActor) REFERENCES Actor(idActor),
    idObra INT NOT NULL, FOREIGN KEY (idObra) REFERENCES Obra(idObra),
    idPersonaje INT NOT NULL, FOREIGN KEY(idPersonaje) REFERENCES Personaje(idPersonaje),
    CONSTRAINT Pk_Interpreta PRIMARY KEY(idActor, idObra, idPersonaje)
);
```

Tabla Participa:

```
CREATE TABLE IF NOT EXISTS Participa(
    idRealizacion INT NOT NULL, FOREIGN KEY(idRealizacion) REFERENCES Realizacion(idRealizacion),
    idObra INT NOT NULL, FOREIGN KEY (idObra) REFERENCES Obra(idObra),
    CONSTRAINT Pk_Participa PRIMARY KEY(idRealizacion, idObra)
);
```

Tabla Saga:

```
CREATE TABLE IF NOT EXISTS Saga(
    idPelicula1 INT NOT NULL, FOREIGN KEY (idPelicula1) REFERENCES Pelicula(idPelicula),
    idPelicula2 INT NOT NULL, FOREIGN KEY (idPelicula2) REFERENCES Pelicula(idPelicula),
    nombreSaga VARCHAR(50),
    CONSTRAINT Pk_Saga PRIMARY KEY (idPelicula1, idPelicula2)
);
```

Insertar los datos en la base de datos

Para poblar la base de datos se han realizado sentencias SQL con comandos “INSERT INTO”.

Ejemplos de estas sentencias son:

```
79375   INSERT INTO Ejemplo2 (idPersona, nombrePersona, genero) VALUES ("4295", "Binev, Nikolai", );
79376   INSERT INTO Ejemplo2 (idPersona, nombrePersona, genero) VALUES ("4296", "Bing, Tracey", );
79377   INSERT INTO Ejemplo2 (idPersona, nombrePersona, genero) VALUES ("4297", "Bini Bustric, Sergio", );
79378   INSERT INTO Ejemplo2 (idPersona, nombrePersona, genero) VALUES ("4298", "Binoche, Juliette", "FEM");
79379   INSERT INTO Ejemplo2 (idPersona, nombrePersona, genero) VALUES ("4299", "Biondi, Lidia", "FEM");
79380   INSERT INTO Ejemplo2 (idPersona, nombrePersona, genero) VALUES ("4300", "Bios", );
79381   INSERT INTO Ejemplo2 (idPersona, nombrePersona, genero) VALUES ("4301", "Bioy Casares, Adolfo", );
79382   INSERT INTO Ejemplo2 (idPersona, nombrePersona, genero) VALUES ("4302", "Birabent, Antonio", "MASC");
```

En esta imagen se aprecian varios datos que van a ser insertados en la tabla Persona. Para cada sentencia se introducen los datos referentes a idPersona, nombre y género, siempre que haya un género.

El fichero que presenta todas las sentencias para poblar la base de datos recibe de nombre *Inserts.sql*.

Consultas

Consultas sobre una tabla

Primera Consulta

Se pretende obtener las Series que fueron emitidas entre los años 1994 y 2004.

$$\text{Series emitidas entre 1994 y 2004} = \sigma_{\text{fechEm} = \text{'1994-2004'}}(\text{Serie})$$

```
SELECT *
FROM Serie s
WHERE fechaEmision = '1994-2004'
```

Resultado:

	idSerie	fechaEmision
1	2	1994-2004

Segunda Consulta

Se pretende obtener el número de capítulos de series cuyas temporadas superen la quinta temporada.

$$\text{Número de capitulos} = \mathcal{G}_{\text{count(nCap)}} \text{ <- } \sigma_{\text{nTemp} > 5}(\text{Capitulo})$$

```
SELECT COUNT(numCap)
AS TempCount
FROM Capitulo c
WHERE numTemp > 5
```

Resultado:

	TempCount
1	2

Tercera Consulta

Se pretende buscar el nombre y apellido de los actores cuyo genero sea MASC (Masculino).

$$genMasc = \pi_{nombrePersona} (\sigma_{genero = \text{“MASC”}} (Personas))$$

```
SELECT nombrePersona FROM Persona WHERE genero='MASC'
```

Primeros 30 resultado:

1	Aaron, Jack
2	Abad de Santillán, Diego
3	Abad, Diego
4	Abad, Diego
5	Abad, Francisco
6	Abad, Kike
7	Abad, Manuel
8	Abadal, Francisco José
9	Abadal, Ignasi
10	Abades, César
11	Abades, Reyes
12	Abades, Reyes
13	Abadou, Youness
14	Abajo, David
15	Abascal, Miguel
16	Abatantuono, Diego
17	Abate, Massimo
18	Abati, Joaquín
19	Abbati, Stefano
20	Abbott, Thomas
21	Abbrescia, Dino
22	Abbu, Anwuar
23	Abdallah, Mouzahem
24	Abdelkrim, HMaia
25	Abdo, Joe
26	Abeal, Marcelo
27	Abeijón, Luis
28	Abel, Dominique
29	Abeledo, Danny
30	Abella, Rafael

Cuarta Consulta

Se pretende buscar el id de las películas que tienen 1 remake y mostrar únicamente el id de las originales.

$$remakeOf = \pi_{idPelicula1} (\sigma_{relacionSaga = \text{“remake of”}} (Saga))$$

```
SELECT idPelicula1 FROM relacionSaga WHERE (nombreSaga = "remake of");
```

Resultado:

	idPelicula1
1	2.956
2	3.469
3	5.426

Consultas sobre varias tablas

Primera Consulta

Se pretende buscar el nombre de las películas que tienen continuación “followed by”.

$$peliculasFollowedBy = \pi_{idPelicula1} (\sigma_{relacionSaga = \text{“followed by”}} (relacionSaga))$$

$$joinPeliculasFollowedBy = peliculasFollowedBy \bowtie idPelicula1 = idPelicula (Películas)$$

$$nombrePeliculasContinuacion = \pi_{nombre} (joinPeliculasFollowedBy)$$

```sql
SELECT DISTINCT titulo
FROM Obra
JOIN
(
    SELECT idPelicula
    FROM Pelicula
    JOIN Saga ON Pelicula.idPelicula = Saga.idPelicula1
    WHERE Saga.nombreSaga = "followed by"
)
AS peliculasFollowedBy
ON Obra.idObra = peliculasFollowedBy.idPelicula;
```

Resultado:

	titulo
10	El Lute (camina o revienta)
11	El pico
12	El retorno del Hombre-Lobo
13	El virgo de Visanteta
14	El último guateque
15	Eva man (Due sessi in uno)
16	Evilio.
17	Hundra
18	Katy, la oruga
19	La bestia y la espada mágica
20	La diosa salvaje
21	La escopeta nacional
22	La guerra de los niños
23	La leyenda del viento del Norte
24	La maldición de la bestia
25	La quinta del porro
26	La segunda guerra de los niños
27	Las cosas del querer
28	Las locuras de Parchís
29	Le lac des morts vivants
30	Le llamaban J.R.
31	Makinavaja, el último choriso
32	Patrimonio nacional
33	Perros callejeros
34	Perros callejeros II
35	Suspiros de España (y Portugal)
36	To er mundo é güeno
37	To er mundo é... mejo!
38	Valentina
39	Victòria! 2: La disbauxa del 17
40	Victòria! La gran aventura dun poble
41	Yo soy Fulana de Tal

Segunda Consulta

Se pretende buscar el id de los productores de las obras cuya fecha de producción sea anterior a 1980.

$$peliculasFechaMayor1980 = \sigma_{fechaProduccion>1980} (\pi_{idObra,\ fechaProduccion} (Obra))$$

$$\rho = \rho\ peliculasFechaMayor1980\ (peliculasFechaMayor1980.idObra,\ fechaProduccion)\ (peliculasFechaMayor1980)$$

$$\rho = \rho\ relacionParticipa\ (relacionParticipa.idObra,\ idRealizacion)\ (relacionParticipa)$$

$$realizacionObras = peliculasFechaMayor1980 \bowtie peliculasFechaMayor1980.idObra = relacionParticipa.idObra\ (relacionParticipa)$$

$$\rho = \rho\ realizacionObras\ (realizacionObras.idRealizacion,\ realizacionObras.idObra)\ (relacionParticipa)$$

$$pr = \sigma_{trabajo = \text{“producer”}}\ (multievaluadoTrabajo)$$

$$\rho = \rho\ pr\ (pr.idRealizacion,\ trabajo)\ (pr)$$

$$productores = realizacionObras \bowtie realizacionObras.idRealizacion = pr.idRealizacion\ (pr)$$

```sql
SELECT Trabajo.idRealizacion, trabajo
FROM Trabajo
JOIN
(
    SELECT idRealizacion
    FROM Participa
    JOIN
    (
        SELECT idObra, fechaProduccion
        FROM Obra
        WHERE (fechaProduccion > 1980)
    ) AS peliculasFechaMayor1980 ON Participa.idObra = peliculasFechaMayor1980.idObra
) AS participa
ON participa.idRealizacion = Trabajo.idRealizacion
WHERE trabajo = "producer";
```

Resultado:

	idRealizacion	trabajo
1	7.104	producer
2	19.188	producer
3	19.551	producer
4	29.447	producer
5	38.322	producer
6	30.402	producer
7	11.709	producer
8	12.384	producer
9	14.652	producer
10	18.166	producer
11	6.445	producer
12	14.652	producer
13	6.845	producer
14	15.736	producer
15	23.037	producer
16	24.420	producer
17	802	producer
18	15	producer
19	8.479	producer
20	31.179	producer
21	8.977	producer
22	20.868	producer
23	28.563	producer
24	28.563	producer
25	28.245	producer
26	29.942	producer
27	4.138	producer
28	7.851	producer

Consultas con subconsulta

Primera Consulta

Se pretende buscar el personaje de género femenino cuyo id sea mayor de 100.

$$idMenor100 = \sigma_{idPersona < 100}(Persona)$$

$$genFem = \sigma_{genero = \text{“FEM”}}(idMenor100)$$

```sql
SELECT *
FROM (
    SELECT *
    FROM Persona p
    WHERE p.idPersona < 100
) AS idMenor100
WHERE idMenor100.genero = 'FEM';
```

Resultado:

	idPersona	nombrePersona	genero
1	4	Aba, Dayrein	FEM
2	6	Abad, Adela	FEM
3	7	Abad, Asun	FEM
4	10	Abad, Esperanza	FEM
5	13	Abad, Isabel	FEM
6	16	Abad, Marisa	FEM
7	17	Abad, Mercedes	FEM
8	20	Abad, Sol	FEM
9	39	Abascal, Nati	FEM
10	40	Abascal, Silvia	FEM
11	46	Abbad, Montse	FEM
12	52	Abbruzzese, Alexandra	FEM
13	55	Abdelhafid, Ami	FEM
14	67	Abella, Nicole	FEM
15	81	Abia, Alba	FEM
16	87	Aboy, Isabel	FEM
17	88	Abradelo, María	FEM
18	92	Abraham, Paloma	FEM
19	96	Abramovitsch, Anna María	FEM

Segunda Consulta

Se pretende buscar las películas que tengan una relación de saga “references”, siempre que ambas dos películas relacionadas posean un id mayor a 1000.

$$\text{referenciaA} = \sigma_{\text{nombreSaga} = \text{“references”}} (\text{relacionSaga})$$

$$\text{idObraMayores1000} = \sigma_{\text{idObra1} > 1000 \text{ AND } \text{idObra2} > 1000} (\text{referenciaA})$$

```sql
SELECT *
FROM (
    SELECT *
    FROM Saga
    WHERE nombreSaga = 'references'
) AS referenciaA
WHERE idPelicula1 > 1000 AND idPelicula2 > 1000;
```

Resultado:

	idPelicula1	idPelicula2	nombreSaga
1	1.775	5.086	references
2	1.933	1.959	references
3	1.933	2.125	references
4	2.450	2.784	references
5	2.450	4.845	references
6	2.627	3.822	references
7	2.635	2.656	references
8	2.635	5.240	references
9	2.694	1.773	references
10	2.791	2.887	references
11	2.853	2.140	references
12	2.853	2.800	references
13	2.853	3.465	references
14	2.853	3.833	references
15	2.853	4.016	references
16	2.893	3.639	references
17	2.893	4.928	references
18	3.001	5.254	references
19	3.104	4.106	references
20	3.309	2.216	references
21	3.334	3.704	references
22	3.334	4.606	references
23	3.334	5.160	references
24	3.687	2.216	references
25	3.704	5.490	references
26	3.743	2.977	references
27	3.818	2.020	references
28	3.818	3.615	references
29	4.021	4.136	references
30	4.073	3.715	references
31	4.646	3.489	references
32	4.943	4.139	references
33	5.144	3.488	references
34	5.144	3.656	references
35	5.159	1.948	references
36	5.159	2.568	references
37	5.159	3.902	references
38	5.159	4.760	references
39	5.160	4.384	references
40	5.237	3.768	references
41	5.490	4.606	references

Integración de SQL en lenguajes de alto nivel

Para la integración de SQL en #C se tienen que incluir las bibliotecas studio.h, stdlib.h, string.h y mysql.h.

En la siguiente imagen se muestran las funciones que se van a usar.

```c
#include "dependencies/mysql.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define SIZE_VARS 200
#define INSERTS_FILE "Inserts.sql"

void init_connection(MYSQL **conn, const char *ddbb);
void close_connection(MYSQL *conn);
void send_query(MYSQL *conn, const char *query);
int count_elements(char *str, char c);
void take_out_char(char *str, char c);
int check_if_all_null(char **var, int elements);
void insert_all_from_documents(MYSQL *con, const char *dir);
void prepare_insert(char *var);

```

Inti_connection se encarga de iniciar la conexión con la base de datos.

```c
void init_connection(MYSQL **conn, const char *ddbb)
{
    const char *host = "127.0.0.1";
    const char *user = "root";
    const char *password = "";
    const char *database = ddbb;
    const int port = 3306;
    if ((*conn = mysql_init(NULL)) == NULL)
    {
        printf("Error: %s", mysql_error(*conn));
        exit(1);
    }

    if (!mysql_real_connect(*conn, host, user, password, database, port, NULL, 0))
    {
        printf("%s\n", mysql_error(*conn));
        exit(1);
    }
}

```

Close_connection se encarga de finalizar la conexión con la base de datos.

```c
void close_connection(MYSQL *conn)
{
    mysql_close(conn);
}

```

Send_query y *count_elements* se encargan de mandar una solicitud a la base de datos una vez se ha conectado y la otra se encarga de contar los elementos de un fichero.

```
69  void send_query(MYSQL *conn, const char *query)
70  {
71      if (mysql_query(conn, query))
72      {
73          printf("%s\n", mysql_error(conn));
74          exit(1);
75      }
76  }
77
78  int count_elements(char *str, char c)
79  {
80      int count = 1;
81      for (int i = 0; str[i] != '\0'; i++)
82      {
83          if (str[i] == c)
84              count++;
85      }
86      return count;
87  }
```

Check_if_all_null se encarga de comprobar si los caracteres que recibe son todos nulos. *Take_out_char* por otra parte, sustituye un elemento por otro dado.

```
89  int check_if_all_null(char **var, int elements)
90  {
91      int aux = 0;
92      for (int i = 0; i < elements; i++)
93      {
94          if (strcmp(var[i], ""))
95              aux = 1;
96      }
97      return aux;
98  }
99
100 void take_out_char(char *str, char c)
101 {
102     char *l = str;
103     while (*l != '\0')
104     {
105         if (*l == c)
106             *l = '\0';
107         l++;
108     }
109 }
```

Prepare_insert transforma datos para que MySQL detecte los datos en su formato correcto. Por ejemplo, si una serie ha sido emitida durante un periodo de tiempo que comprende dos años distintos, en los datos se mostraba como Año1–Año2.

De esta forma, MySQL interpretaba que fechaEmision era una resta entre ambos valores.

```
111  void prepare_insert(char *var)
112  {
113      char *aux = malloc(SIZE_VARS);
114      strcpy(aux, "\"");
115      strcat(aux, var);
116      strcat(aux, "\"");
117      strcpy(var, aux);
118      if (aux != NULL)
119      {
120          free(aux);
121          aux = NULL;
122      }
123  }
```

La función *insert_all_from_documents* que se encarga de leer del fichero que se pasa como parámetro y hace todos los inserts en la base de datos.

```c
void insert_all_from_documents(MYSQL *conn, const char *dir)
{
    FILE *lectura = fopen(dir, "r");
    char insert[300] = "INSERT INTO ";
    char insert_copy[300];
    int elements = 0;
    char linea[300];
    int aux = 0;
    char *table = malloc(40);
    fgets(table, 40, lectura);
    take_out_char(table, '\n');
    strcat(insert, table);
    strcat(insert, " (");
    FILE *escritura = fopen(INSERTS_FILE, "a");
    fgets(linea, 300, lectura);
    elements = count_elements(linea, ';');
    take_out_char(linea, '\n');
    char *data = strtok(linea, ";");
    while (aux < elements)
    {
        strcat(insert, data);
        if (aux != elements - 1)
            strcat(insert, ", ");
        data = strtok(NULL, ";");
        aux++;
    }
    char *var[elements];
    for (int i = 0; i < elements; i++)
        var[i] = malloc(SIZE_VARS);
    strcat(insert, ") VALUES (");
    strcpy(insert_copy, insert);
    if (lectura == NULL)
        return;
    while (!feof(lectura))
    {
        strcpy(insert, insert_copy);
        for (int i = 0; i < elements; i++)
        {
            var[i][0] = '\0';
            if (elements == 1)
                fscanf(lectura, "\r\n%[^\n]s", var[i]);
            else if (i == 0)
                fscanf(lectura, "\r\n%[^;]s;", var[i]);
            else if (i == elements - 1)
                fscanf(lectura, ";%[^\n]s\r\n", var[i]);
            else
                fscanf(lectura, ";%[^;]s;", var[i]);
        }

        if (check_if_all_null(var, elements))
        {
            for (int i = 0; i < elements; i++)
            {
                if (strcmp(var[i], ""))
                    prepare_insert(var[i]);
                strcat(insert, (!strcmp(var[i], "")) ? "NULL" : var[i]);
                if (i != elements - 1)
                    strcat(insert, ", ");
            }
            strcat(insert, ");");
            fprintf(escritura, "%s\n", insert);
            send_query(conn, insert);
        }
    }
    for (int i = 0; i < elements; i++)
    {
        if (var[i] != NULL)
        {
            free(var[i]);
            var[i] = NULL;
        }
    }
    if (data != NULL)
    {
        free(data);
        data = NULL;
    }
    if (table != NULL)
    {
        free(table);
        table = NULL;
    }
    if (lectura != NULL)
        fclose(lectura);
    if (escritura != NULL)
        fclose(escritura);
}
```

Por último, el main es la función donde se realiza la conexión con la base de datos, se inicia la conexión y se insertan los valores propios de nuestra base de datos.

```c
int main(void)
{
    MYSQL *conn;
    init_connection(&conn, "ejemplo");
    FILE *f = fopen(INSERTS_FILE, "w");
    fprintf(f, "");
    if (f != NULL)
        fclose(f);
    insert_all_from_documents(conn, "Tablas/Actor.csv");
    insert_all_from_documents(conn, "Tablas/Capitulo.csv");
    insert_all_from_documents(conn, "Tablas/Multivaluado_genero.csv");
    insert_all_from_documents(conn, "Tablas/Multivaluado_trabajo.csv");
    insert_all_from_documents(conn, "Tablas/Obra.csv");
    insert_all_from_documents(conn, "Tablas/Pelicula.csv");
    insert_all_from_documents(conn, "Tablas/Persona.csv");
    insert_all_from_documents(conn, "Tablas/Personaje.csv");
    insert_all_from_documents(conn, "Tablas/Realizacion.csv");
    insert_all_from_documents(conn, "Tablas/Relacion_esta.csv");
    insert_all_from_documents(conn, "Tablas/Relacion_interpreta.csv");
    insert_all_from_documents(conn, "Tablas/Relacion_participa.csv");
    insert_all_from_documents(conn, "Tablas/Relacion_saga.csv");
    insert_all_from_documents(conn, "Tablas/Serie.csv");
    close_connection(conn);
    return 0;
}
```