



UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA  
*La Universidad Católica de Loja*

Informe Preliminar  
Fundamentos de Bases de Datos

**Autor:** Hermin Leonardo Chuquimarca Jaramillo

Octubre 2022 – Febrero 2023

## Tabla de contenido

1.	Introducción.....	3
2.	Normalización .....	4
3.	Modelos .....	6
4.	Tabla Uno a Muchos (1 : N) .....	9
5.	Tabla Muchos a Muchos (N : N).....	10
6.	Pasos para Realizar el Proyecto .....	12
7.	LIMPIEZA COLUMNA CREW .....	41
8.	Consultas.....	62
9.	Conclusiones .....	64

## **1. Introducción**

Este informe actual tiene como objetivo presentar un archivo CSV que se ha importado a una base de datos mediante un sistema de administración de bases de datos MySQL, que muestra diferentes fases, incluida la inserción de datos, la limpieza de datos, la carga y, especialmente, el uso de datos incluidos. Del mismo modo, podemos distinguir los métodos utilizados para llevar a cabo todas las fases propuestas en este trabajo.

Este informe y los proyectos propuestos tienen como objetivo lograr resultados hacia la gestión y el uso eficiente de información relevante e interesante basada en conjuntos de datos de trabajo.

## 2. Normalización

### First Normal Form (1NF)

Se debe seguir una serie de pasos para normalizar, en otras palabras, para decir que nuestra tabla está en primera forma normal. Estos son:

- Eliminar los grupos repetitivos de las tablas individuales.
- Crear una tabla separada por cada grupo de datos relacionados.
- Identificar cada grupo de datos relacionados con una clave primaria.

Ejemplo:

Una tabla llamada cast con una tabla llamada movie se relacionan en varios por varios (m:n), ya que una película puede tener varios actores, así también como un mismo actor puede participar en diferentes películas; con esta relación se los identifica con una clave primaria idCast y el idMovie respectivamente.

### La segunda Forma Normal (2FN)

Se debe seguir los siguientes pasos:

- Tener la 1º forma normal
- Crear tablas separadas para aquellos grupos de datos que se aplican a varios registros
- Relacionar estas tablas mediante una clave externa

Ejemplo:

La tabla llamada original title que se relaciona con la tabla keywords, IdOriginalTitle y el idKeywords respectivamente, cuentan con una clave externa que es idMovies, debido a que es la clave principal de la tabla universal que es Movies.

### La tercera Forma Normal (3FN)

Se debe considerar los siguientes puntos:

- Tener la 2º forma normal
- Eliminar aquellos campos que no dependan de la clave
- Ninguna columna puede depender de una columna que no tenga una clave
- No puede haber datos derivados

Ejemplo:

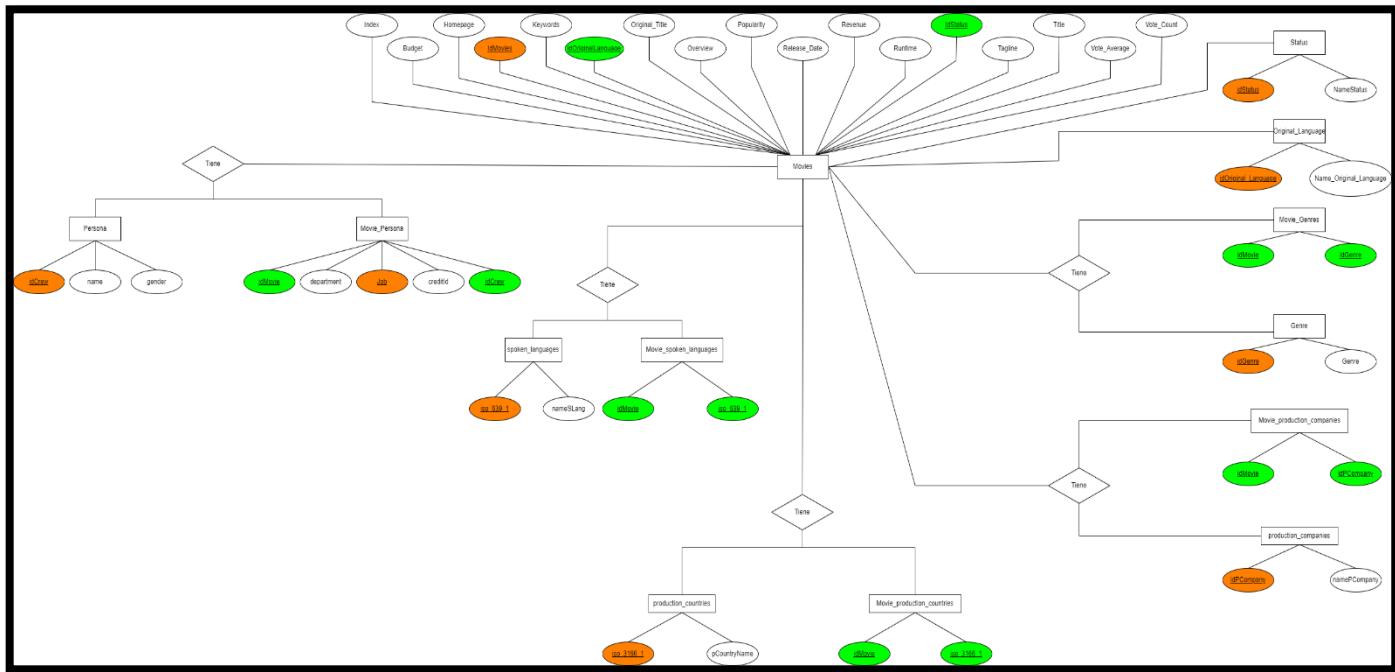
El más claro ejemplo para esta tercera forma normal es la tabla universal “Movies”, ya que es la única tabla que no depende de otras tablas.

Teniendo en base estos conceptos, procedimos a realizar los respectivos modelos Entidad/Relación, Lógico y Físico. Ya con esta conceptualización, generamos un pequeño script con la creación de tablas, determinando ya el tipo de dato que serían de los determinados atributos, establecer los tipos de llaves primarias y foráneas, entre muchas más características necesarias. Esto ya implementado en el motor de bases de datos a utilizar.

### 3. Modelos

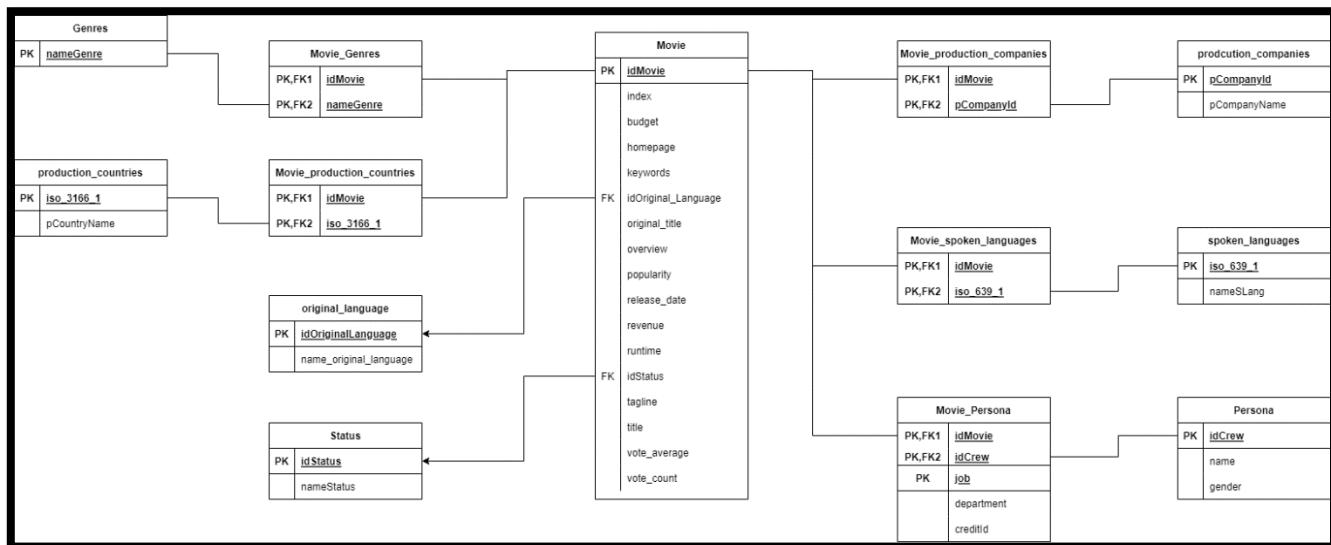
#### Modelo Conceptual

Después de haber establecido los datos mediante la tabla universal, normalizamos los mismos utilizando la metodología Entidad-Relación, identificando los atributos correspondientes a cada relación. Para ello, utilizamos la herramienta draw.io, lo que nos permitió crear diagramas de flujo efectivamente.



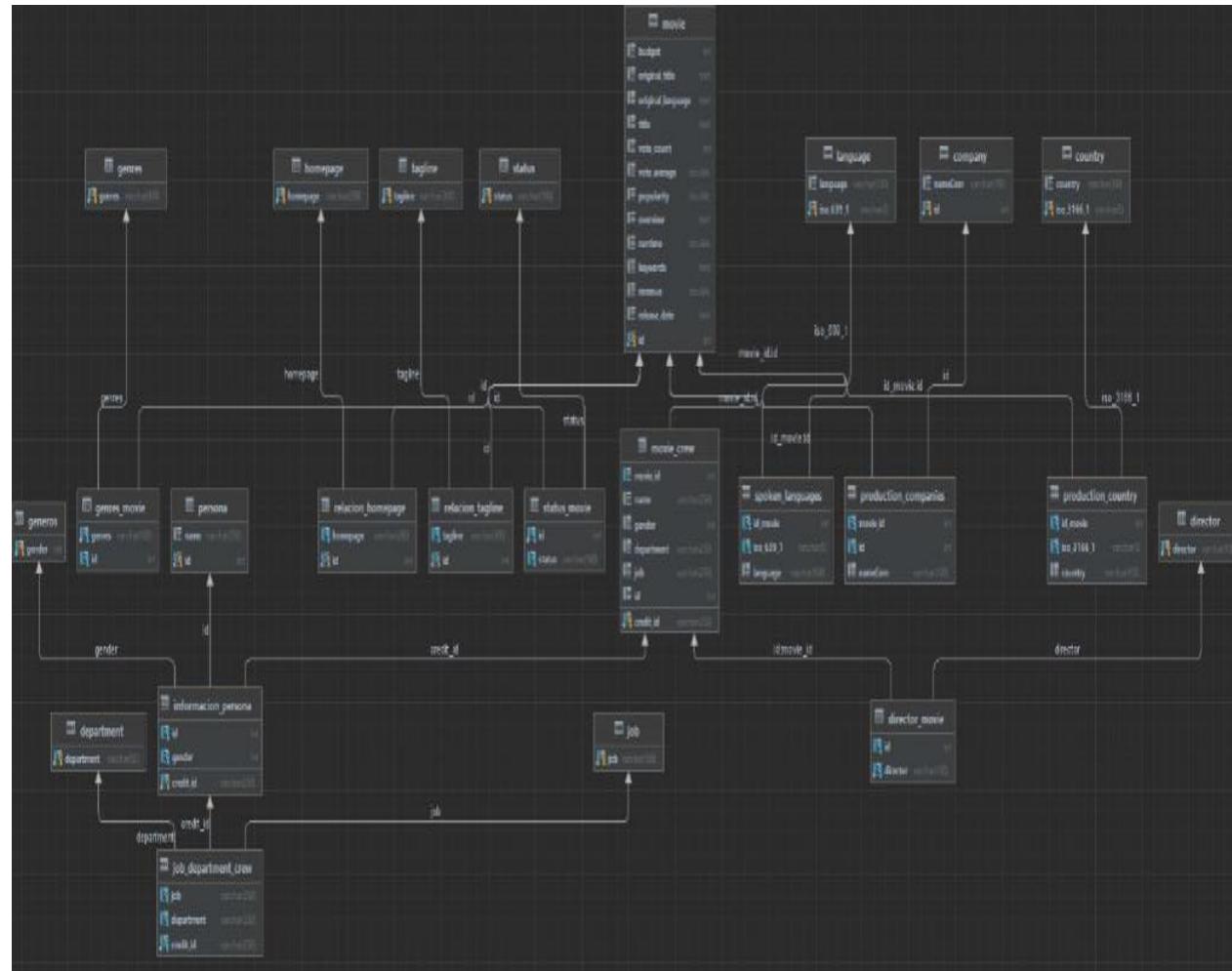
## Modelo Lógico

Con el modelo conceptual ya creado, los atributos apropiados para cada tabla se seleccionan y crean prototipos en función de las entidades y relaciones identificadas. Se declara una clave principal y, opcionalmente, una clave externa en cada tabla para garantizar una base sólida para desarrollar el esquema de la base de datos.



## Modelo Físico

Una vez que el modelo lógico está completo, procedemos a la fase final de construcción del modelo físico. Aquí se especifican las tablas, columnas, claves primarias y sus claves foráneas o claves foráneas y sus relaciones. Este modelo puede generar el correspondiente registro DDL.



Una vez establecidos los modelos se identificó y eliminó grupos repetitivos.

## 4. Tabla Uno a Muchos (1 : N)

Status es una columna con tres posibles valores (released, rumored, post-production)

- Una película puede tener un Status, pero un Status puede asociarse a muchas películas.
- Para solucionar, se crea una tabla separada que se llame Status.
- La llave primaria es nameStatus.
- Como solución, utilizamos el nameStatus como llave foránea en Director que es una columna con el nombre de un único director.
- Una película puede ser dirigida por un Director, pero un Director puede dirigir muchas películas.
- Para solucionar, se crea una tabla separada que se llame Director.
- La llave primaria es directorName.

Como solución, utilizamos el directorName como llave foránea en Original\_language es una columna con el nombre del lenguaje original de la Movie.

Una película tiene un lenguaje original, pero un lenguaje original puede estar en muchas películas, para solucionar, se crea una tabla separada que se llame original\_language, la llave primaria es original\_languageName.

Como solución, utilizamos el original\_languageName como llave foránea en la tabla original\_language.

## 5. Tabla Muchos a Muchos (N : N)

En ninguna de las 4803 entradas se repite, cada película es diferente (única), Cada Movie tiene un index y un id diferente. Como llave primaria utilizamos id el cual nombramos idMovie para no confundirlo con otros id que existan en otras columnas.

Genres contiene un String de géneros que puede contener 0 a n géneros.

- Para solucionar, cada columna debe contener un solo valor.
- En este caso tenemos una lista de géneros. Existen múltiples valores.
- La solución es crear una tabla separada que se llame Genres.
- La llave primaria es genreName
- Una Movie puede tener muchos géneros, y un género puede estar en

muchas Movie.

- La relación (muchos a muchos) se soluciona con una tabla llamada Movie\_Genres utilizando la llave primaria de cada uno.
- Keywords contiene un String de keywords que puede contener (0 a N)
- Para solucionar, cada columna debe contener un solo valor.
- En este caso tenemos una lista de keywords. Existen múltiples valores.
- La solución es crear una tabla separada que se llame Keywords.

- La llave primaria es keywordName
- Una Movie puede tener muchos keywords, y un keyword puede aparecer

en muchas Movie.

- La relación (muchos a muchos) se soluciona con una tabla llamada
- Movie\_Keywords utilizando la llave primaria de cada uno.

Production\_companies contiene un String de JSON que contiene un name y un id que puede contener 0 a n production\_companies.

Para solucionar, cada columna debe contener un solo valor.

- En este caso tenemos una lista de production\_companies. Existen múltiples valores.
- La solución es crear una tabla separada que se llame Production\_companies.
- Tiene dos atributos, name e id los cuales los nombramos

prodCompName y prodCompId el cual es la primary key.

- Una Movie puede tener muchos production\_companies, y un production\_companies puede aparecer en muchas Movie.
- La relación (muchos a muchos) se soluciona con una tabla llamada
- Movie\_Production\_companies utilizando la llave primaria de cada uno.

- Production\_countries contiene un String de JSON que contiene un iso\_3166\_1 y un name que puede contener 0 a n production\_countries.
- Para solucionar, cada columna debe contener un solo valor.

- En este caso tenemos una lista de production\_countries. Existen múltiples valores.
- La solución es crear una tabla separada que se llame Production\_countries. Tiene dos atributos, iso\_3166\_1 y name los cuales los nombramos, iso\_3166\_1 el cual es la primary key y prodCompName.
- Una Movie puede tener muchos production\_countries, y un production\_countries puede aparecer en muchas Movie.
- La relación (muchos a muchos) se soluciona con una tabla llamada
- Movie\_Production\_countries utilizando la llave primaria de cada uno.

Cada Movie tiene uno de los siguientes:

- Production\_countries
- Production\_companies
- Spoken\_languages
- Crew

Para no tener una fila con una clave principal de Película y todos los atributos de cada atributo anterior, use solo la clave principal de cada atributo anterior con película y tenga una tabla con dos claves principales. Luego puede identificar la tabla a la que pertenece y asociar el atributo específico asociado con esa clave de esta manera.

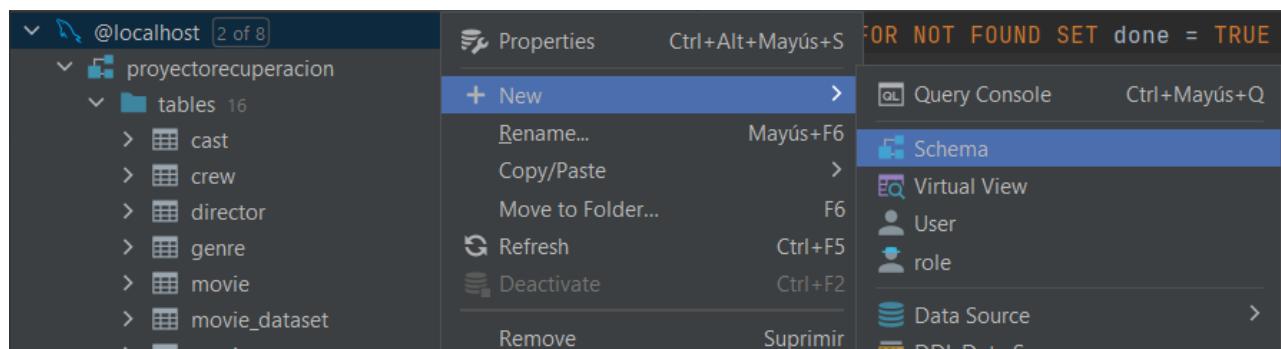
## 6. Pasos para Realizar el Proyecto

Creación de un “Schema”.

Para la creación del “Schema” se podía realizar de dos distintas maneras, la creación automática o por sentencias SQL.

## OPCIÓN 1 – MEDIANTE LA CREACIÓN AUTOMÁTICA

Para la creación automática del Schema dentro del lenguaje de Base de Datos MySql en DataGrip, primero se debe ubicar en la parte superior izquierda, donde se encuentra el localhost que se encuentra conectado, al señalarlo, le aparece un nuevo cuadro donde se tendrá que ir a “New” y luego en el último cuadro se va donde dice “Schema” y automáticamente se crea el schema como se puede ver en el siguiente Anexo:



## OPCIÓN 2 – MEDIANTE SENTENCIAS SQL

Para la creación del “Schema” o base de datos usando sentencias SQL, es necesario hacer lo siguiente:

```
DROP DATABASE IF EXISTS ProyectoBaseRecuperacion;  
CREATE DATABASE ProyectoRecuperacion CHARACTER SET utf8mb4;
```

Es necesario especificarle la codificación de caracteres utf8 ya que en el archivo CSV se usa caracteres especiales y esta misma codificación los transforma.

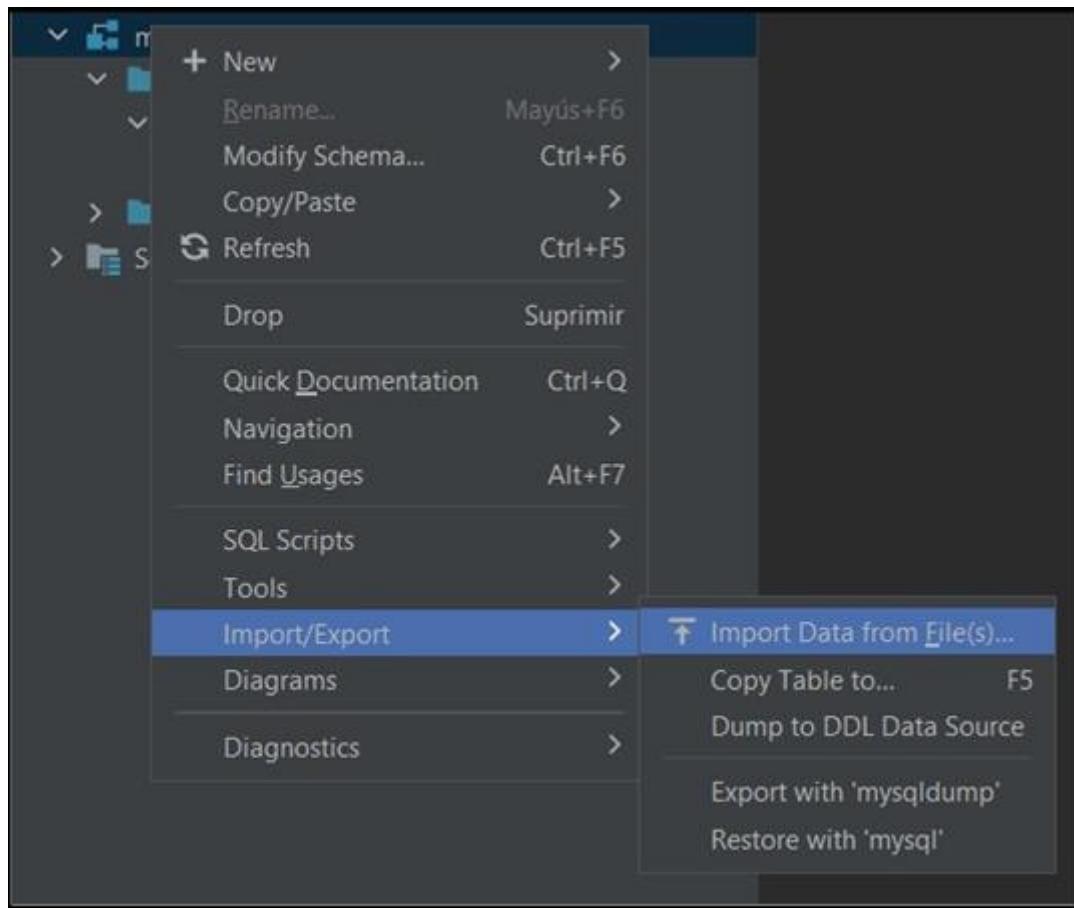
Conexión del “Schema” a la Base De Datos.

Para poderse conectar, por así decirlo, es necesario hacerlo mediante sentencias SQL, la sentencia es la siguiente:

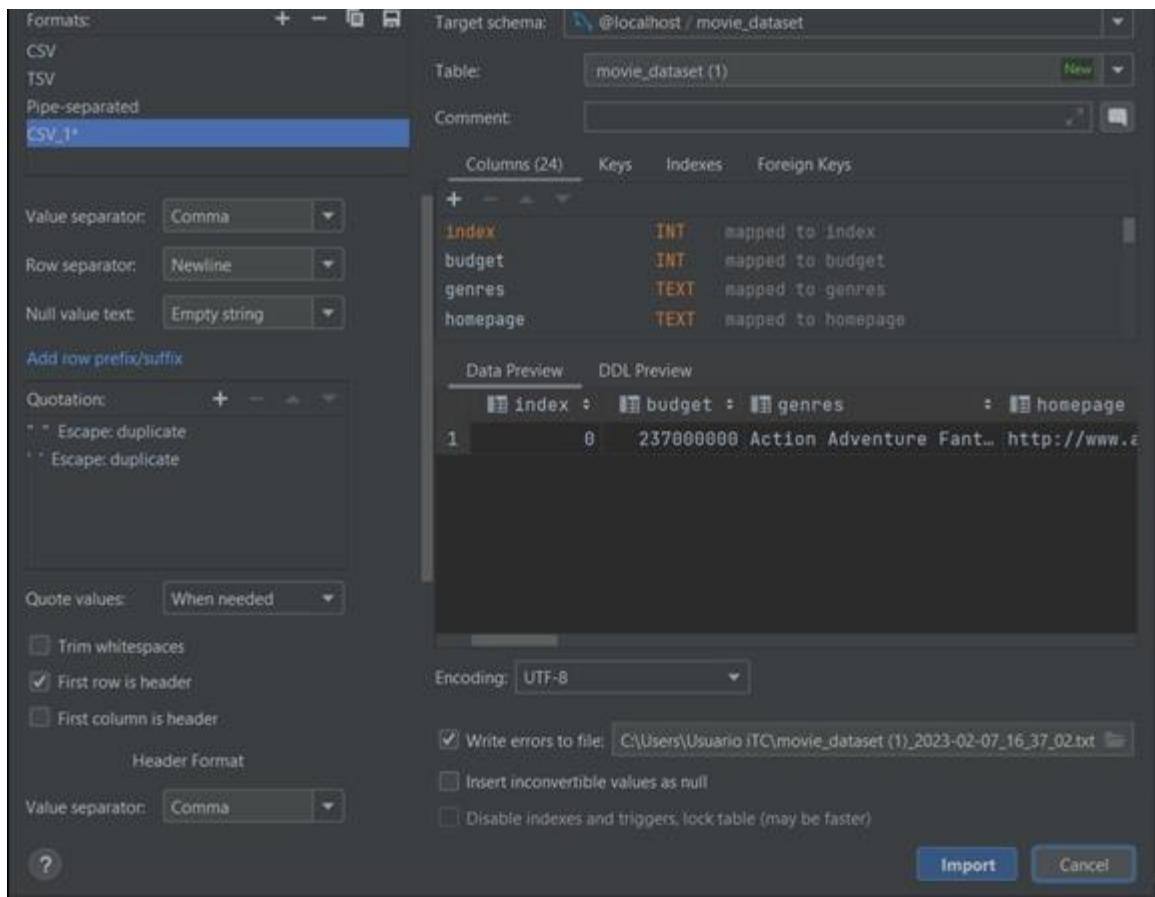
```
USE ProyectoRecuperacion;
```

## Importación del CSV.

DataGrip es un IDE de JetBrains para el manejo de base de datos. Dentro de este existe una herramienta facilitadora de importación de varios tipos de archivos a tablas MySQL



Una vez entrado a este apartado, se habilita una interfaz de importación en donde especificamos que el separador es la coma, y que la primera fila son los títulos de las columnas



## Creación de tablas en DATAGRIP

```
DROP TABLE IF EXISTS original_language;
CREATE TABLE original_language(
    idOriginalLang INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
    name_original_language VARCHAR(2) NOT NULL
);

DROP TABLE IF EXISTS Status;
CREATE TABLE Status(
    idStatus INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
    nameStatus VARCHAR(15) NOT NULL
);
```

```
DROP TABLE IF EXISTS Movie;
CREATE TABLE Movie(
    idMovie INT PRIMARY KEY NOT NULL,
    `index` INT NOT NULL ,
    budget BIGINT NOT NULL,
    homepage VARCHAR(255),
    keywords VARCHAR(255),
    idOrigLang INT NOT NULL,
    original_title VARCHAR(255) NOT NULL,
    overview TEXT,
    popularity DOUBLE NOT NULL,
    release_date DATE,
    revenue BIGINT NOT NULL,
    runtime DOUBLE,
    idStatus INT NOT NULL,
    tagline VARCHAR(255),
    title VARCHAR(255) NOT NULL,
    vote_average DOUBLE NOT NULL,
    vote_count INT NOT NULL,
    FOREIGN KEY (idOrigLang) REFERENCES original_language(idOrigLang),
    FOREIGN KEY (idStatus) REFERENCES status(idStatus)
);
DROP TABLE IF EXISTS Genre;
CREATE TABLE Genre(
    idGenre int PRIMARY KEY AUTO_INCREMENT NOT NULL,
    nameGenre VARCHAR(100)
);
```

```
DROP TABLE IF EXISTS production_countries;
CREATE TABLE production_countries(
    iso_3166_1 VARCHAR(10) PRIMARY KEY NOT NULL,
    pCountryName VARCHAR(255) NOT NULL
);

DROP TABLE IF EXISTS production_companies;
CREATE TABLE production_companies(
    pCompanyId INT PRIMARY KEY NOT NULL,
    pCompanyName VARCHAR(255) NOT NULL
);

DROP TABLE IF EXISTS spoken_language;
CREATE TABLE spoken_language(
    iso_639_1 VARCHAR(2) PRIMARY KEY NOT NULL,
    nameSLang VARCHAR(255) NOT NULL
);

DROP TABLE IF EXISTS Persona;
CREATE TABLE Persona(
    idPerson INT PRIMARY KEY NOT NULL,
    name VARCHAR(255) NOT NULL,
    gender INT
);
```

```
DROP TABLE IF EXISTS Crew;
CREATE TABLE Crew (
    idMovie INT NOT NULL,
    idPerson INT NOT NULL,
    job VARCHAR(255) NOT NULL,
    department VARCHAR(255),
    credit_id VARCHAR(255),
    PRIMARY KEY (idMovie, idPerson, job),
    FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
    FOREIGN KEY (idPerson) REFERENCES Persona(idPerson)
);

DROP TABLE IF EXISTS Cast;
CREATE TABLE Cast (
    idMovie INT NOT NULL,
    idPerson INT NOT NULL,
    FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
    FOREIGN KEY (idPerson) REFERENCES Persona(idPerson)
);

DROP TABLE IF EXISTS Director;
CREATE TABLE Director (
    idMovie INT NOT NULL,
    idPerson INT NOT NULL,
    FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
    FOREIGN KEY (idPerson) REFERENCES Persona(idPerson)
);
```

```

# MUCHOS A MUCHOS

DROP TABLE IF EXISTS Movie_genres;
CREATE TABLE Movie_genres(
    idMovie INT NOT NULL,
    idGenre INT NOT NULL,
    PRIMARY KEY (idMovie, idGenre),
    FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
    FOREIGN KEY (idGenre) REFERENCES Genre(idGenre)
);

DROP TABLE IF EXISTS Movie_production_countries;
CREATE TABLE Movie_production_countries(
    idMovie INT NOT NULL,
    iso_3166_1 VARCHAR(255) NOT NULL,
    PRIMARY KEY (idMovie, iso_3166_1),
    FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
    FOREIGN KEY (iso_3166_1) REFERENCES production_countries(iso_3166_1)
);

DROP TABLE IF EXISTS Movie_production_companies;
CREATE TABLE Movie_production_companies(
    idMovie INT NOT NULL,
    pCompanyId INT NOT NULL,
    PRIMARY KEY (idMovie, pCompanyId),
    FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
    FOREIGN KEY (pCompanyId) REFERENCES production_companies(pCompanyId)
);

DROP TABLE IF EXISTS Movie_spoken_languages;
CREATE TABLE Movie_spoken_languages(
    idMovie INT NOT NULL,
    iso_639_1 VARCHAR(2) NOT NULL,
    PRIMARY KEY (idMovie, iso_639_1),
    FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
    FOREIGN KEY (iso_639_1) REFERENCES spoken_language(iso_639_1)
);

```

## Creación de funciones que Permitan Extraer y Limpiar los Datos.

Descripción de las tablas Director, Status y original\_langauge

(Tablas uno a muchas)

Lo primero que se hace en el procedimiento almacenado en SQL es que este comienza verificando si existe un procedimiento con el nombre "TablaDirector". Si existe, se elimina. Luego, se declaran dos variables: "done" con un valor predeterminado de "FALSE" y "nameDirector" de tipo VARCHAR con una longitud máxima de 100 caracteres.

Después, se declara un cursor "CursorDirector" que selecciona los nombres únicos de los directores en la tabla "movie\_dataset". Se establece un manejador de continuación "CONTINUE HANDLER" para detectar cuando se ha alcanzado el final del cursor. Se abre el cursor y se inicia un ciclo repetitivo que recorre cada uno de los nombres de los directores en el cursor.

En cada iteración del ciclo, se asigna el nombre del director a la variable "nameDirector". Si se ha alcanzado el final del cursor, se sale del ciclo. Si el nombre de director es nulo, se asigna un valor vacío. Además, se tiene casos especiales en los nombres que con el Replace se pueden solucionar en la misma sentencia de código.

Luego, se crea una consulta dinámica que inserta el nombre del director en la tabla "director". Se prepara y ejecuta la consulta dinámica y se libera la memoria de la consulta preparada. Se repite este proceso para cada nombre de director en el cursor.

Finalmente, se cierra el cursor y se finaliza el procedimiento almacenado. En resumen, este procedimiento permite crear y llenar una tabla con los nombres únicos de los directores en una tabla de origen.

Este código nos servirá para poblar estas tres tablas.

### Población Tabla “director”

```
DROP PROCEDURE IF EXISTS TablaDirector;
DELIMITER $$

CREATE PROCEDURE TablaDirector()
BEGIN
DECLARE done INT DEFAULT FALSE ;
DECLARE idPersonas INT;
DECLARE Movid INT;
DECLARE MovDirector VARCHAR(100);
-- Declarar el cursor
DECLARE CursorDirector CURSOR FOR
    SELECT id,director FROM movie_dataset;
-- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
-- Abrir el cursor
OPEN CursorDirector;
drop table if exists directorTemp;
SET @sql_text = 'CREATE TABLE directorTemp ( idPer int,
idMov int)';
PREPARE stmt FROM @sql_text;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
CursorMovie_loop: LOOP
    FETCH CursorDirector INTO Movid,MovDirector;
    -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE CursorMovie_loop;
    END IF;
    SELECT MAX(idPerson) INTO idPersonas FROM Persona WHERE
        SELECT MAX(idPerson) INTO idPersonas FROM Persona WHERE
            Persona.name=MovDirector;
            If idPersonas IS NOT NULL THEN
                INSERT INTO directorTemp VALUES (idPersonas,Movid);
            END IF;
        END LOOP;
    CLOSE CursorDirector;
    select distinct * from directorTemp;
    INSERT INTO Director
        SELECT DISTINCT idMov, idPer
        FROM directorTemp;
    drop table if exists directorTemp;
END $$
DELIMITER ;
CALL TablaDirector();
```

## ROUTINE “TablaDirector”

```
create
  definer = root@localhost procedure TablaDirector()
BEGIN
DECLARE done INT DEFAULT FALSE ;
DECLARE idPersonas INT;
DECLARE Movid INT;
DECLARE MovDirector VARCHAR(100);
-- Declarar el cursor
DECLARE CursorDirector CURSOR FOR
    SELECT id,director FROM movie_dataset;
-- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
-- Abrir el cursor
OPEN CursorDirector;
drop table if exists directorTemp;
SET @sql_text = 'CREATE TABLE directorTemp ( idPer int, idMov int);';
PREPARE stmt FROM @sql_text;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
CursorMovie_loop: LOOP
    FETCH CursorDirector INTO Movid,MovDirector;
    -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE CursorMovie_loop;
    END IF;
    SELECT MAX(idPerson) INTO idPersonas FROM Persona WHERE Persona.name=MovDirector;
    If idPersonas IS NOT NULL THEN
        INSERT INTO directorTemp VALUES (idPersonas,Movid);
    END IF;
    END IF;
    END LOOP;
CLOSE CursorDirector;
select distinct * from directorTemp;
INSERT INTO Director
    SELECT DISTINCT idMov, idPer
    FROM directorTemp;
drop table if exists directorTemp;
END;
```

## Población Tabla “Status”

```
DROP PROCEDURE IF EXISTS TablaStatus;
DELIMITER $$

CREATE PROCEDURE TablaStatus()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE nameStatus VARCHAR(15);
    -- Declarar el cursor
    DECLARE CursorStatus CURSOR FOR
        SELECT DISTINCT CONVERT(status USING UTF8MB4) AS names from movie_dataset;
    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    -- Abrir el cursor
    OPEN CursorStatus;
    CursorStatus_loop: LOOP
        FETCH CursorStatus INTO nameStatus;
        -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
        IF done THEN
            LEAVE CursorStatus_loop;
        END IF;
        SET @_oStatement = CONCAT('INSERT INTO Status (nameStatus) VALUES (\'' ,
        nameStatus,'\'');');
        PREPARE sent1 FROM @_oStatement;
        EXECUTE sent1;
        DEALLOCATE PREPARE sent1;
    END LOOP;
    CLOSE CursorStatus;
END $$

DELIMITER ;
CALL TablaStatus();
```

ROUTINE “TablaStatus”

```
create
  definer = root@localhost procedure |TablaStatus()
BEGIN
DECLARE done INT DEFAULT FALSE;
DECLARE nameStatus VARCHAR(15);
-- Declarar el cursor
DECLARE CursorStatus CURSOR FOR
    SELECT DISTINCT CONVERT(status USING UTF8MB4) AS names from movie_dataset;
-- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
-- Abrir el cursor
OPEN CursorStatus;
CursorStatus_loop: LOOP
    FETCH CursorStatus INTO nameStatus;
-- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE CursorStatus_loop;
    END IF;
    SET @_oStatement = CONCAT('INSERT INTO Status (nameStatus) VALUES (\'',nameStatus,'\'');');
    PREPARE sent1 FROM @_oStatement;
    EXECUTE sent1;
    DEALLOCATE PREPARE sent1;
END LOOP;
CLOSE CursorStatus;
END;
```

## Población Tabla “Original\_language”

```
DROP PROCEDURE IF EXISTS TablaOriginalLanguage;
DELIMITER $$

CREATE PROCEDURE TablaOriginalLanguage()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE nameOL VARCHAR(2);
    -- Declarar el cursor
    DECLARE CursorOL CURSOR FOR
        SELECT DISTINCT original_language AS names from movie_dataset;
    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    -- Abrir el cursor
    OPEN CursorOL;
    CursorOL_loop: LOOP
        FETCH CursorOL INTO nameOL;
        -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
        IF done THEN
            LEAVE CursorOL_loop;
        END IF;
        SET @_oStatement = CONCAT('INSERT INTO original_language
(name_original_language) VALUES (\',
nameOL, \');');
        PREPARE sent1 FROM @_oStatement;
        EXECUTE sent1;
        DEALLOCATE PREPARE sent1;
    END LOOP;
    CLOSE CursorOL;
END $$
```

```
DELIMITER ;
CALL TablaOriginalLanguage();
```

ROUTINE “TablaOriginalLanguage”

```
create
  definer = root@localhost procedure TablaOriginalLanguage()
BEGIN
DECLARE done INT DEFAULT FALSE;
DECLARE nameOL VARCHAR(2);
-- Declarar el cursor
DECLARE CursorOL CURSOR FOR
  SELECT DISTINCT original_language AS names from movie_dataset;
-- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
-- Abrir el cursor
OPEN CursorOL;
CursorOL_loop: LOOP
  FETCH CursorOL INTO nameOL;
-- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
  IF done THEN
    LEAVE CursorOL_loop;
  END IF;
  SET @_oStatement = CONCAT('INSERT INTO original_language
(name_original_language) VALUES (\'' ,
nameOL,''\');");
  PREPARE sent1 FROM @_oStatement;
  EXECUTE sent1;
  DEALLOCATE PREPARE sent1;
END LOOP;
CLOSE CursorOL;
END;
```

Debido a que la tabla Movie utiliza llaves foráneas, era necesario crear las tablas director, Status y original\_langauge para utilizar sus llaves primarias como foráneas en Movie.  
(REFERENCES)

### Población Tabla “Movie”

El siguiente código SQL es un procedimiento almacenado que crea una tabla denominada "películas" en la base de datos. La tabla se crea a partir de los datos de un conjunto de datos denominado "movie\_dataset".

Un procedimiento almacenado primero elimina todos los procedimientos existentes con el mismo nombre. Luego se declaran varias variables para almacenar los valores de cada columna en la tabla "movie\_dataset"

A continuación, cree un cursor llamado "CursorMovie" que seleccione datos de la tabla "movie\_dataset". Los cursores se utilizan para recorrer cada fila de una tabla y extraer valores de cada columna. Se Declara un "manejador" para manejar el caso cuando se alcance el final del puntero.

El cursor se abre y "CursorMovie\_loop" comienza a recorrer cada fila de la tabla "movie\_dataset". Dentro del ciclo, la declaración FETCH se usa para recuperar valores de la fila actual y almacenarlos en variables previamente declaradas.

Si se alcanza el final del puntero, el ciclo termina. De lo contrario, comprueba si el nombre del administrador está vacío. Si es así, se le asigna un valor de cero. Ejecuta una consulta para obtener la identificación del director a partir de su nombre y guárdela en la variable "Director\_idDirector".

Finalmente, se inserta una nueva fila en la tabla "movies" con el valor obtenido de la tabla "movie\_dataset" y el ID del director. Cuando se completa el ciclo, el cursor se cierra y el procedimiento almacenado finaliza.

```
DROP PROCEDURE IF EXISTS TablaMovie;
DELIMITER $$

CREATE PROCEDURE TablaMovie()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE Mov_idMovie INT;
    DECLARE Mov_index INT;
    DECLARE Mov_budget BIGINT;
    DECLARE Mov_homepage VARCHAR(255);
    DECLARE Mov_keywords VARCHAR(255);
    DECLARE Mov_name_original_language VARCHAR(2);
    DECLARE Mov_original_title VARCHAR(255);
    DECLARE Mov_overview TEXT;
    DECLARE Mov_popularity DOUBLE;
    DECLARE Mov_release_date DATE;
    DECLARE Mov_revenue BIGINT;
    DECLARE Mov_runtime DOUBLE;
    DECLARE Mov_nameStatus VARCHAR(15);
    DECLARE Mov_tagline VARCHAR(255);
    DECLARE Mov_title VARCHAR(255);
    DECLARE Mov_vote_average DOUBLE;
    DECLARE Mov_vote_count INT;
    DECLARE Status_idStatus int;
    DECLARE OL_idOriginal_language int;
    -- Declarar el cursor
    DECLARE CursorMovie CURSOR FOR
```

```


DECLARE CursorMovie CURSOR FOR
    SELECT id,`index`,budget,hompage, keywords, original_language, original_title,
overview,
        popularity, release_date, revenue, runtime, `status`, tagline, title,
        vote_average, vote_count FROM movie_dataset;
-- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
-- Abrir el cursor
OPEN CursorMovie;
CursorMovie_loop: LOOP
    FETCH CursorMovie INTO Mov_idMovie,Mov_index,Mov_budget, Mov_homepage,
Mov_keywords,Mov_name_original_language,
        Mov_original_title, Mov_overview, Mov_popularity, Mov_release_date,
Mov_revenue, Mov_runtime,
        Mov_nameStatus, Mov_tagline, Mov_title, Mov_vote_average, Mov_vote_count;
    -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE CursorMovie_loop;
    END IF;
    SELECT `idStatus` INTO Status_idStatus FROM Status WHERE nameStatus =
Mov_nameStatus;
    SELECT `idOriginalLang` INTO OL_idOriginal_language FROM original_language WHERE
name_original_language = Mov_name_original_language;
    INSERT INTO Movie
(`idMovie`,`index`,budget,hompage,keywords,idOriginalLang,original_title,overview,popularity,release_date,rev
    idStatus, tagline,title,vote_average,vote_count)
    VALUES (Mov_idMovie,Mov_index,Mov_budget, Mov_homepage,
    Mov_keywords,OL_idOriginal_language,
        Mov_original_title, Mov_overview, Mov_popularity, Mov_release_date,
Mov_revenue, Mov_runtime,
        Status_idStatus, Mov_tagline, Mov_title, Mov_vote_average, Mov_vote_count);
END LOOP;
CLOSE CursorMovie;
END $$
DELIMITER ;
CALL TablaMovie ();

```

## ROUTINE “TablaMovie”

```
CREATE PROCEDURE TablaMovie()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE Mov_idMovie INT;
    DECLARE Mov_index INT;
    DECLARE Mov_budget BIGINT;
    DECLARE Mov_homepage VARCHAR(255);
    DECLARE Mov_keywords VARCHAR(255);
    DECLARE Mov_name_original_language VARCHAR(2);
    DECLARE Mov_original_title VARCHAR(255);
    DECLARE Mov_overview TEXT;
    DECLARE Mov_popularity DOUBLE;
    DECLARE Mov_release_date DATE;
    DECLARE Mov_revenue BIGINT;
    DECLARE Mov_runtime DOUBLE;
    DECLARE Mov_nameStatus VARCHAR(15);
    DECLARE Mov_tagline VARCHAR(255);
    DECLARE Mov_title VARCHAR(255);
    DECLARE Mov_vote_average DOUBLE;
    DECLARE Mov_vote_count INT;
    DECLARE Status_idStatus int;
    DECLARE OL_idOriginal_language int;
    -- Declarar el cursor
    DECLARE CursorMovie CURSOR FOR
        SELECT id, `index`, budget, homepage, keywords, original_language, original_title, overview, popularity,
               revenue, runtime, `status`, tagline, title, vote_average, vote_count FROM movie_dataset;
    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    -- Abrir el cursor
    OPEN CursorMovie;
    CursorMovie_loop: LOOP
        FETCH CursorMovie INTO Mov_idMovie, Mov_index, Mov_budget, Mov_homepage, Mov_keywords, Mov_name_original_language,
                           Mov_original_title, Mov_overview, Mov_popularity, Mov_release_date, Mov_revenue, Mov_runtime, Mov_tagline,
                           Mov_title, Mov_vote_average, Mov_vote_count;
        -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
        IF done THEN
            LEAVE CursorMovie_loop;
        END IF;
        SELECT `idStatus` INTO Status_idStatus FROM Status WHERE nameStatus = Mov_nameStatus;
        SELECT `idOrigLang` INTO OL_idOriginal_language FROM original_language WHERE name_original_language = Mov_name_original_language;
        INSERT INTO Movie(`idMovie`, `index`, budget, homepage, keywords, idOrigLang, original_title, overview, popularity,
                         release_date, revenue, runtime, idStatus, tagline, title, vote_average, vote_count)
        VALUES (Mov_idMovie, Mov_index, Mov_budget, Mov_homepage, Mov_keywords, OL_idOriginal_language, Mov_original_title,
                Mov_overview, Mov_popularity, Mov_release_date, Mov_revenue, Mov_runtime, Status_idStatus, Mov_tagline,
                Mov_title, Mov_vote_average, Mov_vote_count);
    END LOOP;
    CLOSE CursorMovie;
END;
```

Tenemos tres columnas con valores JSON, para las cuales seguiremos el mismo procedimiento para las tres que son spoken\_languages, production\_companies y production\_countries

Este código es un procedimiento que se encarga de extraer información de una tabla llamada "movie\_dataset", en particular la columna "production\_companies", que contiene información en formato JSON sobre las compañías productoras de películas.

El procedimiento comienza declarando variables, un cursor que se encarga de recorrer las filas de la tabla "movie\_dataset" y un controlador de eventos "CONTINUE HANDLER" para determinar cuándo se ha alcanzado el final de los datos. Luego, se crea una tabla temporal llamada "production\_companieTem" para almacenar los datos extraídos del formato JSON.

El cursor se usa para recorrer las filas de la tabla "movie\_dataset" y, para cada fila, se extrae la información de las compañías productoras de las películas utilizando la función "JSON\_EXTRACT". Los datos extraídos se insertan en la tabla "production\_companieTem".

Después de que se han recorrido todas las filas, se seleccionan los datos únicos de la tabla "production\_companieTem" y se insertan en la tabla "production\_companie". Finalmente, se cierra el cursor y se elimina la tabla temporal "production\_companieTem".

## Población Tabla “Production\_companies”

```
DROP PROCEDURE IF EXISTS TablaProduction_companies;
DELIMITER $$

CREATE PROCEDURE TablaProduction_companies ()
BEGIN
    DECLARE done INT DEFAULT FALSE ;
    DECLARE jsonData json ;
    DECLARE jsonId varchar(250) ;
    DECLARE jsonLabel varchar(250) ;
    DECLARE i INT;
    -- Declarar el cursor
    DECLARE myCursor
    CURSOR FOR
        SELECT JSON_EXTRACT(CONVERT(production_companies USING UTF8MB4), '$[*]') FROM
movie_dataset ;
    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER
        FOR NOT FOUND SET done = TRUE ;
    -- Abrir el cursor
    OPEN myCursor ;
    drop table if exists production_companietem;
    SET @sql_text = 'CREATE TABLE production_companietem ( id int, nameCom
VARCHAR(255));';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
cursorLoop: LOOP
    FETCH myCursor INTO jsonData;
```

```

cursorLoop: LOOP
    FETCH myCursor INTO jsonData;
    -- Controlador para buscar cada uno de los arrays
    SET i = 0;
    -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE cursorLoop ;
    END IF ;
    WHILE(JSON_EXTRACT(jsonData, CONCAT('$[', i, ']')) IS NOT NULL) DO
        SET jsonId = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].id')), '') ;
        SET jsonLabel = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].name')), '') ;
        SET i = i + 1;
        SET @sql_text = CONCAT('INSERT INTO production_companieTem VALUES (
REPLACE(jsonId,'\'',''), ', ', jsonLabel, '); ');
        PREPARE stmt FROM @sql_text;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;
    END WHILE;
    END LOOP ;
    select distinct * from production_companieTem;
    INSERT INTO production_companies
    SELECT DISTINCT id, nameCom
    FROM production_companieTem;
    drop table if exists production_companieTem;
    CLOSE myCursor ;
END$$
DELIMITER ;
call TablaProduction_companies();

```

## ROUTINE “TablaProduction\_companies”

```
create
  definer = root@localhost procedure TablaProduction_companies()
BEGIN
    DECLARE done INT DEFAULT FALSE ;
    DECLARE jsonData json ;
    DECLARE jsonId varchar(250) ;
    DECLARE jsonLabel varchar(250) ;
    DECLARE i INT;
    -- Declarar el cursor
    DECLARE myCursor
        CURSOR FOR
            SELECT JSON_EXTRACT(CONVERT(production_companies USING UTF8MB4), '$[*]') FROM movie_dataset ;
    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE ;
    -- Abrir el cursor
    OPEN myCursor ;
    drop table if exists production_companietem;
    SET @sql_text = 'CREATE TABLE production_companietem ( id int, nameCom VARCHAR(255));';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
cursorLoop: LOOP
    FETCH myCursor INTO jsonData;
    -- Controlador para buscar cada uno de los arrays
    SET i = 0;
    -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE cursorLoop ;
    END IF ;
    --
    END IF ;
    WHILE(JSON_EXTRACT(jsonData, CONCAT('[$', i, ']')) IS NOT NULL) DO
        SET jsonId = IFNULL(JSON_EXTRACT(jsonData, CONCAT('[$', i, '].id')), '') ;
        SET jsonLabel = IFNULL(JSON_EXTRACT(jsonData, CONCAT('[$', i, '].name')), '') ;
        SET i = i + 1;
        SET @sql_text = CONCAT('INSERT INTO production_companietem VALUES (', REPLACE(jsonId,'\'',''), ', ', jsonLabel, ')');
        PREPARE stmt FROM @sql_text;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;
    END WHILE;
END LOOP ;
select distinct * from production_companietem;
    INSERT INTO production_companies
    SELECT DISTINCT id, nameCom
    FROM production_companietem;
    drop table if exists production_companietem;
CLOSE myCursor ;
END;
```

## Población Tabla “Production\_countries”

```
DROP PROCEDURE IF EXISTS TablaProduction_countries;
DELIMITER $$

CREATE PROCEDURE TablaProduction_countries ()
BEGIN
    DECLARE done INT DEFAULT FALSE ;
    DECLARE jsonData json ;
    DECLARE jsonId varchar(250) ;
    DECLARE jsonLabel varchar(250) ;
    DECLARE i INT;
    -- Declarar el cursor
    DECLARE myCursor
        CURSOR FOR
            SELECT JSON_EXTRACT(CONVERT(production_countries USING UTF8MB4), '$[*]') FROM
movie_dataset ;
    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER
        FOR NOT FOUND SET done = TRUE ;
    -- Abrir el cursor
    OPEN myCursor ;
    drop table if exists production_countriesTem;
    SET @sql_text = 'CREATE TABLE production_countriesTem ( iso_3166_1 varchar(2),
nameCountry VARCHAR(255));';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
cursorLoop: LOOP
    FETCH myCursor INTO jsonData;
```

```

    FETCH myCursor INTO jsonData;
    -- Controlador para buscar cada uno de los arrays
    SET i = 0;
    -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE cursorLoop ;
    END IF ;
    WHILE(JSON_EXTRACT(jsonData, CONCAT('$[', i, ']')) IS NOT NULL) DO
        SET jsonId = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].iso_3166_1')), '');
        ;
        SET jsonLabel = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].name')), '' );
        SET i = i + 1;
        SET @sql_text = CONCAT('INSERT INTO production_countriesTem VALUES (',
        REPLACE(jsonId,'\\'', ''), ', ', jsonLabel, '); ');
        PREPARE stmt FROM @sql_text;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;
    END WHILE;
    END LOOP ;
    select distinct * from production_countriesTem;
    INSERT INTO production_countries
    SELECT DISTINCT iso_3166_1, nameCountry
    FROM production_countriesTem;
    drop table if exists production_countriesTem;
    CLOSE myCursor ;
END$$
DELIMITER ;
call TablaProduction_countries();

```

ROUTINE “TablaProduction\_countries”

```
create
    definer = root@localhost procedure |TablaMovie_production_countries()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE idMovie int;
    DECLARE idProdCoun text;
    DECLARE idJSON text;
    DECLARE i INT;
    -- Declarar el cursor
    DECLARE myCursor
        CURSOR FOR
            SELECT id, production_countries FROM movie_dataset;
    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE ;
    -- Abrir el cursor
    OPEN myCursor ;
    drop table if exists MovieProdCompTemp;
    SET @sql_text = 'CREATE TABLE MovieProdCompTemp ( id int, idGenre varchar(255) )';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
cursorLoop: LOOP
    FETCH myCursor INTO idMovie, idProdCoun;
    -- Controlador para buscar cada uno de los arrays
    SET i = 0;
    -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE cursorLoop ;
    END IF ;
    LEAVE cursorLoop ;
END IF ;
WHILE(JSON_EXTRACT(idProdCoun, CONCAT('$[', i,'].iso_3166_1')) IS NOT NULL) DO
    SET idJSON = JSON_EXTRACT(idProdCoun, CONCAT('$[', i,'].iso_3166_1')) ;
    SET i = i + 1;
    SET @sql_text = CONCAT('INSERT INTO MovieProdCompTemp VALUES (' , idMovie, ', ', REPLACE(idJSON,'\\',''))');
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
END WHILE;
END LOOP ;
select distinct * from MovieProdCompTemp;
    INSERT INTO Movie_production_countries
        SELECT DISTINCT id, idGenre
        FROM MovieProdCompTemp;
    drop table if exists MovieProdCompTemp;
    CLOSE myCursor ;
END;
```

## Población Tabla “Spoken\_languages”

```
DROP PROCEDURE IF EXISTS TablaSpokenLanguages;
DELIMITER $$

CREATE PROCEDURE TablaSpokenLanguages ()
BEGIN
    DECLARE done INT DEFAULT FALSE ;
    DECLARE jsonData json ;
    DECLARE jsonId varchar(250) ;
    DECLARE jsonLabel varchar(250) ;
    DECLARE i INT;
    -- Declarar el cursor
    DECLARE myCursor
    CURSOR FOR
        SELECT JSON_EXTRACT(CONVERT(spoken_languages USING UTF8MB4), '$[*]') FROM
    movie_dataset ;
    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER
    FOR NOT FOUND SET done = TRUE ;
    -- Abrir el cursor
    OPEN myCursor ;
    drop table if exists spokenLanguagesTem;
    SET @sql_text = 'CREATE TABLE spokenLanguagesTem ( iso_639_1 varchar(2),
    nameLang VARCHAR(255));';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
cursorLoop: LOOP
    FETCH myCursor INTO jsonData;
```

```
FETCH myCursor INTO jsonData;
-- Controlador para buscar cada uno de los arrays
SET i = 0;
-- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
IF done THEN
    LEAVE cursorLoop ;
END IF ;
WHILE(JSON_EXTRACT(jsonData, CONCAT('$[', i, ']')) IS NOT NULL) DO
    SET jsonId = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].iso_639_1')), '');
    SET jsonLabel = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].name')), '') ;
    SET i = i + 1;
    SET @sql_text = CONCAT('INSERT INTO spokenLanguagesTem VALUES (',
REPLACE(jsonId,'\\',''), ', ', jsonLabel, ')'); ''
PREPARE stmt FROM @sql_text;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
END WHILE;
END LOOP ;
select distinct * from spokenLanguagesTem;
    INSERT INTO spoken_language
    SELECT DISTINCT iso_639_1, nameLang
    FROM spokenLanguagesTem;
    drop table if exists spokenLanguagesTem;
CLOSE myCursor ;
END$$
DELIMITER ;
call TablaSpokenLanguages();
```

ROUTINE “TablaSpokenLanguages”

```
create
  definer = root@localhost procedure |TablaMovie_spoken_languages()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE idMovie int;
  DECLARE idSpokLang text;
  DECLARE idJSON text;
  DECLARE i INT;
  -- Declarar el cursor
  DECLARE myCursor
    CURSOR FOR
  SELECT id, spoken_languages FROM movie_dataset;
  -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE ;
  -- Abrir el cursor
  OPEN myCursor ;
cursorLoop: LOOP
  FETCH myCursor INTO idMovie, idSpokLang;
  -- Controlador para buscar cada uno de los arrays
  SET i = 0;
  -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
  IF done THEN
    LEAVE cursorLoop ;
  END IF ;
  WHILE(JSON_EXTRACT(idSpokLang, CONCAT('$[', i, '].iso_639_1')) IS NOT NULL) DO
    SET idJSON = JSON_EXTRACT(idSpokLang, CONCAT('$[', i, '].iso_639_1')) ;
    SET i = i + 1;
    SET @sql_text = CONCAT('INSERT INTO Movie_spoken_languages VALUES (', idMovie, ',', REPLACE(idJSON, '\'', PREPARE stmt FROM @sql_text;

PREPARE stmt FROM @sql_text;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
  END WHILE;
END LOOP ;
CLOSE myCursor ;
END;
```

## 7. LIMPIEZA COLUMNA CREW

1. Primero, usemos IntelliJ y el lenguaje de programación Scala para leer el archivo CSV y comenzar a analizar las columnas de Crew. Encontré algunos casos en los que el valor de la clave "Nombre" contenía comillas simples.
2. Analice diferentes patrones en los datos para encontrar cuándo se usan comillas simples, como en O'Brian, o cuándo se usan comillas dobles para encerrar apodos como "D.J." I was. Las comillas son hasta '\', por lo que se consideran parte de la cadena de nombre, no JSON.
3. El análisis de los casos muestra que hay casos como e', t' o d' y estos patrones coinciden con las claves 'nombre', 'departamento', 'id\_crédito' o 'id'. Por lo tanto, no era posible reemplazar el valor e', t' o d' con e\\', t\\' o d\\', ya que provocaría un error en la clave.
4. Para las intercalaciones 0, 1 y 2, tuve que convertir según la posición y el carácter al lado. Como tal, simplemente ponlo entre comillas.
5. El valor clave de "id" también tenía que estar entre comillas simples, así que lo puse de acuerdo con el valor antes y después de "\id\:", pero se convirtió en "\id\'\:" puse ' entre comillas antes del valor, reemplazó '}',' con '\'},' y colocó una comilla simple al final del valor.
6. En el caso del último valor de “id” reemplazamos '}]' al colocar '\"}]' en su lugar. A diferencia del último reemplazo en el punto 5, el} no está seguido por un, debido a que es el último valor del arreglo. Termina con una llave] así que por eso tomamos este caso en consideración para que todos los últimos valores de key tengan comillas cerradas.

7. Tras realizar todos los REPLACE, realizamos un JSON\_EXTRACT JSON\_EXTRACT  
(CONVERT(\*SENTENCIA\_DE\_TODOS\_LOS\_REPLACE\* USING UTF8MB4), '\$[\*]')  
para extraer todos los valores JSON.

8. Hemos realizado exitosamente la limpieza de la columna “crew”. El siguiente paso es  
realizar le carga de estos datos a su respectiva tabla/s.

```
DROP PROCEDURE IF EXISTS TablaCrew;
DELIMITER $$

CREATE PROCEDURE TablaCrew ()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE idMovie int;
    DECLARE idCrew text;
    DECLARE idJSON text;
    DECLARE jobJSON text;
    DECLARE departmentJSON text;
    DECLARE credit_idJSON text;
    DECLARE i INT;
    -- Declarar el cursor
    DECLARE myCursor
    CURSOR FOR
        SELECT id, CONVERT(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE
            (REPLACE(crew, '\"', '\\"'), '{\"', '\"'),
            '\"': '\"', '\"': '\"'), '\"', '\"', '\"'), '\"': '\"', '\"': '\"'), '\"', '\"', '\"')
        USING UTF8mb4 ) FROM movie_dataset;
    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER
    FOR NOT FOUND SET done = TRUE ;
    -- Abrir el cursor
    OPEN myCursor ;
    cursorLoop: LOOP
        FETCH myCursor INTO idMovie, idCrew;
        -- Controlador para buscar cada uno de los arrays
        SET i = 0;
```

```
SET i = 0;
-- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
IF done THEN
    LEAVE cursorLoop ;
END IF ;
WHILE(JSON_EXTRACT(idCrew, CONCAT('$[', i, '].id')) IS NOT NULL) DO
    SET idJSON = JSON_EXTRACT(idCrew, CONCAT('$[', i, '].id')) ;
    SET jobJSON = JSON_EXTRACT(idCrew, CONCAT('$[', i, '].job')) ;
    SET departmentJSON = JSON_EXTRACT(idCrew, CONCAT('$[', i, '].department')) ;
    SET credit_idJSON = JSON_EXTRACT(idCrew, CONCAT('$[', i, '].credit_id')) ;
    SET i = i + 1;
    SET @sql_text = CONCAT('INSERT INTO Crew VALUES (' , idMovie, ', ', ,
REPLACE(idJSON, '\'', '') , ', ', REPLACE(jobJSON, '\'', '') , ', ',
REPLACE(departmentJSON, '\'', '') , ', ', REPLACE(credit_idJSON, '\'', '') , ') ; ');
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
END WHILE;
END LOOP ;
CLOSE myCursor ;
END$$
DELIMITER ;
call TablaCrew();
```

## ROUTINE “TablaCrew”

```
create
  definer = root@localhost procedure |TablaCrew()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE idMovie int;
    DECLARE idCrew text;
    DECLARE idJSON text;
    DECLARE jobJSON text;
    DECLARE departmentJSON text;
    DECLARE credit_idJSON text;
    DECLARE i INT;
    -- Declarar el cursor
    DECLARE myCursor
        CURSOR FOR
            SELECT id, CONVERT(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(crew, "", "\\"), '{\"}, '\"),
            '\"', '\"', '\"'), '\"', '\"'), '\"', '\"') USING UTF8mb4 ) FROM movie_dataset;
    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE ;
    -- Abrir el cursor
    OPEN myCursor ;
cursorLoop: LOOP
    FETCH myCursor INTO idMovie, idCrew;
    -- Controlador para buscar cada uno de los arrays
    SET i = 0;
    -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE cursorLoop ;
    END IF ;
    WHILE(JSON_EXTRACT(idCrew, CONCAT('$[', i, ']'.id)) IS NOT NULL) DO
        WHILE(JSON_EXTRACT(idCrew, CONCAT('$[', i, ']'.id)) IS NOT NULL) DO
            SET idJSON = JSON_EXTRACT(idCrew, CONCAT('$[', i, ']'.id));
            SET jobJSON = JSON_EXTRACT(idCrew, CONCAT('$[', i, ']'.job));
            SET departmentJSON = JSON_EXTRACT(idCrew, CONCAT('$[', i, ']'.department));
            SET credit_idJSON = JSON_EXTRACT(idCrew, CONCAT('$[', i, ']'.credit_id));
            SET i = i + 1;
            SET @sql_text = CONCAT('INSERT INTO Crew VALUES (' , idMovie, ', ', REPLACE(idJSON,'\"',''), ', ',',
            REPLACE(jobJSON,'\"',''), ', ', REPLACE(departmentJSON,'\"',''), ', ', REPLACE(credit_idJSON,'\"','')
PREPARE stmt FROM @sql_text;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
    END WHILE;
END LOOP ;
CLOSE myCursor ;
END;
```

## Población “Genre”

```
DROP PROCEDURE IF EXISTS TablaGenre;
DELIMITER $$

CREATE PROCEDURE TablaGenre()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE nameGenre VARCHAR(100);
    -- Declarar el cursor
    DECLARE CursorGenre CURSOR FOR
        SELECT DISTINCT CONVERT(REPLACE(REPLACE(genres, 'Science Fiction', 'Science-
Fiction'),
        'TV Movie', 'TV-Movie') USING UTF8MB4) from movie_dataset;
    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    -- Abrir el cursor
    OPEN CursorGenre;
    drop table if exists temperolgenre;
    SET @sql_text = 'CREATE TABLE temperolgenre (name VARCHAR(100));';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
CursorDirector_loop: LOOP
    FETCH CursorGenre INTO nameGenre;
    -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE CursorDirector_loop;
    END IF;
    -- Separar los géneros en una tabla temporal
    DROP TEMPORARY TABLE IF EXISTS temp_genres;

```

```
DROP TEMPORARY TABLE IF EXISTS temp_genres;
CREATE TEMPORARY TABLE temp_genres (genre VARCHAR(50));
SET @_genres = nameGenre;
WHILE (LENGTH(@_genres) > 0) DO
    SET @_genre = TRIM(SUBSTRING_INDEX(@_genres, ' ', 1));
    INSERT INTO temp_genres (genre) VALUES (@_genre);
    SET @_genres = SUBSTRING(@_genres, LENGTH(@_genre) + 2);
END WHILE;
-- Insertar los géneros separados en filas individuales
INSERT INTO temperolgenre (name)
SELECT genre FROM temp_genres;
END LOOP CursorDirector_loop;
select distinct * from temperolgenre;
INSERT INTO Genre (nameGenre)
SELECT DISTINCT name
FROM temperolgenre;
drop table if exists temperolgenre;
CLOSE CursorGenre;
END $$
DELIMITER ;
CALL TablaGenre();
```

## ROUTINE “TablaGenre”

```
create
  definer = root@localhost procedure TablaGenre()
BEGIN
DECLARE done INT DEFAULT FALSE;
DECLARE nameGenre VARCHAR(100);
-- Declarar el cursor
DECLARE CursorGenre CURSOR FOR
  SELECT DISTINCT CONVERT(REPLACE(REPLACE(genres, 'Science Fiction', 'Science-
Fiction'),
  'TV Movie', 'TV-Movie') USING UTF8MB4) from movie_dataset;
-- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
-- Abrir el cursor
OPEN CursorGenre;
drop table if exists temperolgenre;
  SET @sql_text = 'CREATE TABLE temperolgenre (name VARCHAR(100));';
  PREPARE stmt FROM @sql_text;
  EXECUTE stmt;
  DEALLOCATE PREPARE stmt;
CursorDirector_loop: LOOP
  FETCH CursorGenre INTO nameGenre;
  -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
  IF done THEN
    LEAVE CursorDirector_loop;
  END IF;
  -- Separar los géneros en una tabla temporal
  DROP TEMPORARY TABLE IF EXISTS temp_genres;
  CREATE TEMPORARY TABLE temp_genres (genre VARCHAR(50));
  SET @_genres = nameGenre;

  SET @_genres = nameGenre;
  WHILE (LENGTH(@_genres) > 0) DO
    SET @_genre = TRIM(SUBSTRING_INDEX(@_genres, ' ', 1));
    INSERT INTO temp_genres (genre) VALUES (@_genre);
    SET @_genres = SUBSTRING(@_genres, LENGTH(@_genre) + 2);
  END WHILE;
  -- Insertar los géneros separados en filas individuales
  INSERT INTO temperolgenre (name)
  SELECT genre FROM temp_genres;
END LOOP CursorDirector_loop;
select distinct * from temperolgenre;
  INSERT INTO Genre (nameGenre)
  SELECT DISTINCT name
  FROM temperolgenre;
  drop table if exists temperolgenre;
CLOSE CursorGenre;
END;
```

## Población “persona”

```
DROP PROCEDURE IF EXISTS TablaPersona;
DELIMITER $$

CREATE PROCEDURE TablaPersona ()
BEGIN
    DECLARE done INT DEFAULT FALSE ;
    DECLARE jsonData json ;
    DECLARE jsonId INT ;
    DECLARE jsonLabel varchar(250) ;
    DECLARE jsongenre VARCHAR(250);
    DECLARE i INT;
    -- Declarar el cursor
    DECLARE CursorPerson
    CURSOR FOR
        SELECT JSON_EXTRACT(CONVERT(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE
            (REPLACE(crew, "'", "\''), '{\'', '{"'),
            '\': \'', '\"'), '\', \"', '\"'), '\': ', '\"', ')', '\', \"', '\"')
        USING UTF8mb4 ), '$[*]') FROM movie_dataset;
    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER
        FOR NOT FOUND SET done = TRUE ;
    -- Abrir el cursor
    OPEN CursorPerson ;
    drop table if exists personTem;
    SET @sql_text = 'CREATE TABLE personTem ( idcrew int, name VARCHAR(255), gender
int)';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
cursorLoop: LOOP
```

```

FETCH CursorPerson INTO jsonData;
-- Controlador para buscar cada uno de los arrays
SET i = 0;
-- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
IF done THEN
    LEAVE cursorLoop ;
END IF ;
WHILE(JSON_EXTRACT(jsonData, CONCAT('[$', i, ']')) IS NOT NULL) DO
    SET jsonId = IFNULL(JSON_EXTRACT(jsonData, CONCAT('[$', i, '].id')), '') ;
    SET jsonLabel = IFNULL(JSON_EXTRACT(jsonData, CONCAT('[$', i, '].name')), '') ;
    SET jsongenre = IFNULL(JSON_EXTRACT(jsonData, CONCAT('[$', i, '].gender')), '') ;
    SET i = i + 1;
    SET @sql_text = CONCAT('INSERT INTO personTem VALUES (',
REPLACE(jsonId,'\\'', ''), ', ',
jsonLabel, ', ', REPLACE(jsongenre, '\\'', ''), ');');
PREPARE stmt FROM @sql_text;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
END WHILE;
END LOOP ;
select distinct * from personTem;
INSERT IGNORE INTO Persona
SELECT DISTINCT idcrew, name, gender
FROM personTem;
drop table if exists personTem;
CLOSE CursorPerson ;
END$$
DELIMITER ;
CALL TablaPersona();

```

## ROUTINE “TablaPersona”

```
create
  definer = root@localhost procedure TablaPersona()
BEGIN
    DECLARE done INT DEFAULT FALSE ;
    DECLARE jsonData json ;
    DECLARE jsonId INT ;
    DECLARE jsonLabel varchar(250) ;
    DECLARE jsongenre VARCHAR(250);
    DECLARE i INT;
    -- Declarar el cursor
    DECLARE CursorPerson
    CURSOR FOR
        SELECT JSON_EXTRACT(CONVERT(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(crew, '\"', '\\"'), '\"\\', '\"\\',
        '\"': '\"'), '\\", \\", \"', '\"'), '\"': '\"', '\"', '\"', '\"')
        USING UTF8mb4 ), '$[*]') FROM movie_dataset;
    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER
        FOR NOT FOUND SET done = TRUE ;
    -- Abrir el cursor
    OPEN CursorPerson ;
    drop table if exists personTem;
    SET @sql_text = 'CREATE TABLE personTem ( idcrew int, name VARCHAR(255), gender int);';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
cursorLoop: LOOP
    FETCH CursorPerson INTO jsonData;
    -- Controlador para buscar cada uno de los arrays
    SET i = 0;
```

```
    SET i = 0;
    -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE cursorLoop ;
    END IF ;
    WHILE(JSON_EXTRACT(jsonData, CONCAT('$[', i, ']')) IS NOT NULL) DO
        SET jsonId = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].id')), '' );
        SET jsonLabel = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].name')), '' );
        SET jsongenre = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].gender')), '' );
        SET i = i + 1;
        SET @sql_text = CONCAT('INSERT INTO personTem VALUES (' , REPLACE(jsonId,'\"',''), ', ', jsonLabel,',',
        REPLACE(jsongenre,'\"',''), ');');
        PREPARE stmt FROM @sql_text;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;
    END WHILE;
END LOOP ;
select distinct * from personTem;
    INSERT IGNORE INTO Persona
    SELECT DISTINCT idcrew, name, gender
    FROM personTem;
    drop table if exists personTem;
    CLOSE CursorPerson ;
END;
```

## MUCHOS A MUCHOS

### Población “Movie – Genres”

```
DROP PROCEDURE IF EXISTS populate_movie_genres;
-- Crear un procedimiento almacenado para insertar los registros en la tabla Movie_genres
DELIMITER $$

CREATE PROCEDURE populate_movie_genres()
BEGIN
    -- Declarar variables locales
    DECLARE i INT DEFAULT 1;
    DECLARE n INT;
    DECLARE current_genre VARCHAR(255);
    DECLARE current_genres_array VARCHAR(255);

    -- Obtener la cantidad de registros en la tabla movie_dataset
    SELECT COUNT(*) FROM movie_dataset INTO n;

    -- Bucle a través de cada registro en la tabla movie_dataset
    WHILE i <= n DO
        -- Obtener los géneros del registro actual
        SELECT genres FROM movie_dataset WHERE `index` = i INTO current_genres_array;

        -- Reemplazar 'Science Fiction' con 'Sciencie Fiction'
        SET current_genres_array = REPLACE(current_genres_array, 'Science Fiction', 'Sciencie Fiction');
        -- Reemplazar 'TV Movie' con 'TV-Movie'
        SET current_genres_array = REPLACE(current_genres_array, 'TV Movie', 'TV-Movie');

        -- Separar los géneros en un array
        SET current_genres_array = CONCAT(current_genres_array, ',');
        WHILE LENGTH(current_genres_array) > 0 DO
            SET current_genre = TRIM(SUBSTR(current_genres_array, 1, INSTR(current_genres_array, ',') - 1));
            SET current_genres_array = SUBSTR(current_genres_array, INSTR(current_genres_array, ',') + 1);

            -- Buscar el idGenre correspondiente a current_genre en la tabla Genre
            SELECT idGenre FROM Genre WHERE nameGenre = current_genre INTO @id;

            -- Agregar el género a la tabla Movie_genres si se encontró un idGenre correspondiente
            IF @id IS NOT NULL THEN
                INSERT IGNORE INTO Movie_genres (idMovie, idGenre) VALUES (i, @id);
            END IF;
        END WHILE;

        -- Incrementar el índice
        SET i = i + 1;
    END WHILE;
END $$

DELIMITER ;
call populate_movie_genres();
```

## ROUTINE “populate\_movie\_genres”

```
create
  definer = root@localhost procedure |populate_movie_genres()
BEGIN
    -- Declarar variables locales
    DECLARE i INT DEFAULT 1;
    DECLARE n INT;
    DECLARE current_genre VARCHAR(255);
    DECLARE current_genres_array VARCHAR(255);

    -- Obtener la cantidad de registros en la tabla movie_dataset
    SELECT COUNT(*) FROM movie_dataset INTO n;

    -- Bucle a través de cada registro en la tabla movie_dataset
    WHILE i <= n DO
        -- Obtener los géneros del registro actual
        SELECT genres FROM movie_dataset WHERE `index` = i INTO current_genres_array;

        -- Reemplazar 'Science Fiction' con 'Sciencie Fiction'
        SET current_genres_array = REPLACE(current_genres_array, 'Science Fiction', 'Sciencie Fiction');
        -- Reemplazar 'TV Movie' con 'TV-Movie'
        SET current_genres_array = REPLACE(current_genres_array, 'TV Movie', 'TV-Movie');

        -- Separar los géneros en un array
        SET current_genres_array = CONCAT(current_genres_array, ',');
        WHILE LENGTH(current_genres_array) > 0 DO
            SET current_genre = TRIM(SUBSTR(current_genres_array, 1, INSTR(current_genres_array, ',') - 1));
            SET current_genres_array = SUBSTR(current_genres_array, INSTR(current_genres_array, ',') + 1);

            SET current_genre = TRIM(SUBSTR(current_genres_array, 1, INSTR(current_genres_array, ',') - 1));
            SET current_genres_array = SUBSTR(current_genres_array, INSTR(current_genres_array, ',') + 1);

            -- Buscar el idGenre correspondiente a current_genre en la tabla Genre
            SELECT idGenre FROM Genre WHERE nameGenre = current_genre INTO @id;

            -- Agregar el género a la tabla Movie_genres si se encontró un idGenre correspondiente
            IF @id IS NOT NULL THEN
                INSERT IGNORE INTO Movie_genres (idMovie, idGenre) VALUES (i, @id);
            END IF;
        END WHILE;

        -- Incrementar el índice
        SET i = i + 1;
    END WHILE;
END;
```

## Población “Movie – Production Companies”

```
DROP PROCEDURE IF EXISTS TablaMovie_production_companies;
DELIMITER $$

CREATE PROCEDURE TablaMovie_production_companies ()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE idMovie int;
    DECLARE idProdComp JSON;
    DECLARE idJSON text;
    DECLARE i INT;
    -- Declarar el cursor
    DECLARE myCursor
    CURSOR FOR
        SELECT id, production_companies FROM movie_dataset;
    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER
        FOR NOT FOUND SET done = TRUE ;
    -- Abrir el cursor
    OPEN myCursor ;
    drop table if exists MovieProdCompTemp;
    SET @sql_text = 'CREATE TABLE MovieProdCompTemp ( id int, idGenre int );';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
cursorLoop: LOOP
    FETCH myCursor INTO idMovie, idProdComp;
    -- Controlador para buscar cada uno de los arrays
    SET i = 0;
```

```
controlador para buscar cada uno de los arrays
SET i = 0;
-- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
IF done THEN
    LEAVE cursorLoop ;
END IF ;
WHILE(JSON_EXTRACT(idProdComp, CONCAT('$[', i, '].id')) IS NOT NULL) DO
    SET idJSON = JSON_EXTRACT(idProdComp, CONCAT('$[', i, '].id')) ;
    SET i = i + 1;
    SET @sql_text = CONCAT('INSERT INTO MovieProdCompTemp VALUES (' , idMovie, ', ', ,
REPLACE(idJSON,'\\'', ''), ') ; ');
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
END WHILE;
END LOOP ;
select distinct * from MovieProdCompTemp;
INSERT INTO Movie_production_companies
SELECT DISTINCT id, idGenre
FROM MovieProdCompTemp;
drop table if exists MovieProdCompTemp;
CLOSE myCursor ;
END$$
DELIMITER ;
call TablaMovie_production_companies();
```

## ROUTINE “TablaMovie\_production\_companies”

```
create
  definer = root@localhost procedure TablaMovie_production_companies()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE idMovie int;
    DECLARE idProdComp JSON;
    DECLARE idJSON text;
    DECLARE i INT;
    -- Declarar el cursor
    DECLARE myCursor
        CURSOR FOR
            SELECT id, production_companies FROM movie_dataset;
    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE ;
    -- Abrir el cursor
    OPEN myCursor ;
    drop table if exists MovieProdCompTemp;
    SET @sql_text = 'CREATE TABLE MovieProdCompTemp ( id int, idGenre int );';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
cursorLoop: LOOP
    FETCH myCursor INTO idMovie, idProdComp;
    -- Controlador para buscar cada uno de los arrays
    SET i = 0;
    -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE cursorLoop ;
    END IF ;
    -- Si no se cumple la condición de salir, se ejecuta el bucle
    WHILE(JSON_EXTRACT(idProdComp, CONCAT('[$', i, '].id')) IS NOT NULL) DO
        SET idJSON = JSON_EXTRACT(idProdComp, CONCAT('[$', i, '].id')) ;
        SET i = i + 1;
        SET @sql_text = CONCAT('INSERT INTO MovieProdCompTemp VALUES (' , idMovie, ', ', REPLACE(idJSON, '\'', ''))');
        PREPARE stmt FROM @sql_text;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;
    END WHILE;
END LOOP ;
select distinct * from MovieProdCompTemp;
INSERT INTO Movie_production_companies
SELECT DISTINCT id, idGenre
FROM MovieProdCompTemp;
drop table if exists MovieProdCompTemp;
CLOSE myCursor ;
END;
```

## Población “Movie – Production Countries”

```
DROP PROCEDURE IF EXISTS TablaMovie_production_countries;
DELIMITER $$

CREATE PROCEDURE TablaMovie_production_countries ()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE idMovie int;
    DECLARE idProdCoun text;
    DECLARE idJSON text;
    DECLARE i INT;
    -- Declarar el cursor
    DECLARE myCursor
    CURSOR FOR
        SELECT id, production_countries FROM movie_dataset;
    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER
        FOR NOT FOUND SET done = TRUE ;
    -- Abrir el cursor
    OPEN myCursor ;
    drop table if exists MovieProdCompTemp;
    SET @sql_text = 'CREATE TABLE MovieProdCompTemp ( id int, idGenre
varchar(255) )';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
cursorLoop: LOOP
    FETCH myCursor INTO idMovie, idProdCoun;
    -- Controlador para buscar cada uno de los arrays
    SET i = 0;
```

```
SET i = 0;
-- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
IF done THEN
    LEAVE cursorLoop ;
END IF ;
WHILE(JSON_EXTRACT(idProdCoun, CONCAT('$[', i, '].iso_3166_1')) IS NOT NULL) DO
    SET idJSON = JSON_EXTRACT(idProdCoun, CONCAT('$[', i, '].iso_3166_1')) ;
    SET i = i + 1;
    SET @sql_text = CONCAT('INSERT INTO MovieProdCompTemp VALUES (' , idMovie, ', ', ,
REPLACE(idJSON,'\\'', ''), '); ');
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
END WHILE;
END LOOP ;
select distinct * from MovieProdCompTemp;
INSERT INTO Movie_production_countries
SELECT DISTINCT id, idGenre
FROM MovieProdCompTemp;
drop table if exists MovieProdCompTemp;
CLOSE myCursor ;
END$$
DELIMITER ;
call TablaMovie_production_countries();
```

## ROUTINE “TablaMovie\_production\_countries”

```
create
  definer = root@localhost procedure |TablaMovie_production_countries()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE idMovie int;
    DECLARE idProdCoun text;
    DECLARE idJSON text;
    DECLARE i INT;
    -- Declarar el cursor
    DECLARE myCursor
        CURSOR FOR
            SELECT id, production_countries FROM movie_dataset;
    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE ;
    -- Abrir el cursor
    OPEN myCursor ;
    drop table if exists MovieProdCompTemp;
    SET @sql_text = 'CREATE TABLE MovieProdCompTemp ( id int, idGenre varchar(255) )';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
cursorLoop: LOOP
    FETCH myCursor INTO idMovie, idProdCoun;
    -- Controlador para buscar cada uno de los arrays
    SET i = 0;
    -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE cursorLoop ;
    END IF ;

    END IF ;
    WHILE(JSON_EXTRACT(idProdCoun, CONCAT('$[', i, '].iso_3166_1')) IS NOT NULL) DO
        SET idJSON = JSON_EXTRACT(idProdCoun, CONCAT('$[', i, '].iso_3166_1')) ;
        SET i = i + 1;
        SET @sql_text = CONCAT('INSERT INTO MovieProdCompTemp VALUES (', idMovie, ', ', REPLACE(idJSON,'\'',''), ',');
        PREPARE stmt FROM @sql_text;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;
    END WHILE;
END LOOP ;
select distinct * from MovieProdCompTemp;
INSERT INTO Movie_production_countries
SELECT DISTINCT id, idGenre
FROM MovieProdCompTemp;
drop table if exists MovieProdCompTemp;
CLOSE myCursor ;
END;
```

## Población “Movie – Spoken Languages

```
DROP PROCEDURE IF EXISTS TablaMovie_spoken_languages;
DELIMITER $$

CREATE PROCEDURE TablaMovie_spoken_languages ()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE idMovie int;
    DECLARE idSpokLang text;
    DECLARE idJSON text;
    DECLARE i INT;
    -- Declarar el cursor
    DECLARE myCursor
    CURSOR FOR
        SELECT id, spoken_languages FROM movie_dataset;
    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER
        FOR NOT FOUND SET done = TRUE ;
    -- Abrir el cursor
    OPEN myCursor ;
    cursorLoop: LOOP
        FETCH myCursor INTO idMovie, idSpokLang;
        -- Controlador para buscar cada uno de los arrays
        SET i = 0;
        -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
        IF done THEN
            LEAVE cursorLoop ;
        END IF ;
        WHILE(JSON_EXTRACT(idSpokLang, CONCAT('$[', i, '].iso_639_1')) IS NOT NULL) DO
            SET idJSON = JSON_EXTRACT(idSpokLang, CONCAT('$[', i, '].iso_639_1')) ;
            SET i = i + 1;
```

```
        SET i = i + 1;
        SET @sql_text = CONCAT('INSERT INTO Movie_spoken_languages VALUES (', idMovie, ', ', REPLACE(idJSON,'\'',''), ')');
        PREPARE stmt FROM @sql_text;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;
    END WHILE;
    END LOOP ;
    CLOSE myCursor ;
END$$
DELIMITER ;
call TablaMovie_spoken_languages();
```

## ROUTINE “TablaMovie\_spoken\_languages”

```
create
  definer = root@localhost procedure TablaMovie_spoken_languages()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE idMovie int;
    DECLARE idSpokLang text;
    DECLARE idJSON text;
    DECLARE i INT;
    -- Declarar el cursor
    DECLARE myCursor
        CURSOR FOR
            SELECT id, spoken_languages FROM movie_dataset;
    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE ;
    -- Abrir el cursor
    OPEN myCursor ;
    cursorLoop: LOOP
        FETCH myCursor INTO idMovie, idSpokLang;
        -- Controlador para buscar cada uno de los arrays
        SET i = 0;
        -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
        IF done THEN
            LEAVE cursorLoop ;
        END IF ;
        WHILE(JSON_EXTRACT(idSpokLang, CONCAT('$[', i, '].iso_639_1')) IS NOT NULL) DO
            SET idJSON = JSON_EXTRACT(idSpokLang, CONCAT('$[', i, '].iso_639_1')) ;
            SET i = i + 1;
            SET @sql_text = CONCAT('INSERT INTO Movie_spoken_languages VALUES (' , idMovie, ', ', REPLACE(idJSON, '\'', '
PREPARE stmt FROM @sql_text;

                PREPARE stmt FROM @sql_text;
                EXECUTE stmt;
                DEALLOCATE PREPARE stmt;
            END WHILE;
        END LOOP ;
        CLOSE myCursor ;
    END;
```

## Población “Cast”

```
DROP PROCEDURE IF EXISTS tabla_cast;
DELIMITER $$

CREATE PROCEDURE tabla_cast()
BEGIN
    DECLARE done INT DEFAULT 0;
    DECLARE movie_id INT;
    DECLARE movie_cast VARCHAR(1000);
    DECLARE actor_name VARCHAR(255);
    DECLARE actor_list CURSOR FOR SELECT id, cast FROM movie_dataset;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
    OPEN actor_list;
    read_loop: LOOP
        FETCH actor_list INTO movie_id, movie_cast;
        IF done THEN
            LEAVE read_loop;
        END IF;
        WHILE LENGTH(movie_cast) > 0 DO
            SET actor_name = TRIM(SUBSTRING_INDEX(movie_cast, ' ', 1));
            SET movie_cast = TRIM(SUBSTR(movie_cast, LENGTH(actor_name) + 1));
            IF LENGTH(actor_name) > 0 THEN
                INSERT IGNORE INTO Cast (idMovie, idPerson)
                    SELECT movie_id, idPerson FROM Persona WHERE name = actor_name;
            END IF;
        END WHILE;
    END LOOP;
    CLOSE actor_list;
END $$

DELIMITER ;
call tabla_cast();
```

## ROUTINE “tabla\_cast”

```
create
  definer = root@localhost procedure |tabla_cast()
BEGIN DECLARE done INT DEFAULT 0;
    DECLARE movie_id INT;
    DECLARE movie_cast VARCHAR(1000);
    DECLARE actor_name VARCHAR(255);
    DECLARE actor_list CURSOR FOR SELECT id, cast FROM movie_dataset;

    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

    OPEN actor_list;

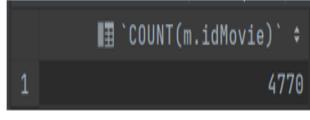
    read_loop: LOOP
        FETCH actor_list INTO movie_id, movie_cast;
        IF done THEN
            LEAVE read_loop;
        END IF;
        WHILE LENGTH(movie_cast) > 0 DO
            SET actor_name = TRIM(SUBSTRING_INDEX(movie_cast, ' ', 1));
            SET movie_cast = TRIM(SUBSTR(movie_cast, LENGTH(actor_name) + 1));
            IF LENGTH(actor_name) > 0 THEN
                INSERT IGNORE INTO Cast (idMovie, idPerson)
                    SELECT movie_id, idPerson FROM Persona WHERE name = actor_name;
            END IF;
        END WHILE;
    END LOOP;
    CLOSE actor_list;
END;
```

## 8. Consultas.

Ejemplo 1: ¿Cuántas películas por idioma tienen su tabla de películas (original language)?

Consulta SQL	Resultado																																																				
<pre>SELECT ol.name_original_language, COUNT(m.idOrigLang) FROM Movie m, original_language ol WHERE m.idOrigLang = ol.idOrigLang GROUP BY m.idOrigLang;</pre>	<table border="1"> <thead> <tr> <th>name_original_language</th> <th>COUNT(m.idOrigLang)</th> </tr> </thead> <tbody> <tr><td>en</td><td>4582</td></tr> <tr><td>ja</td><td>16</td></tr> <tr><td>fr</td><td>79</td></tr> <tr><td>zh</td><td>27</td></tr> <tr><td>es</td><td>32</td></tr> <tr><td>de</td><td>27</td></tr> <tr><td>hi</td><td>19</td></tr> <tr><td>ru</td><td>11</td></tr> <tr><td>ko</td><td>11</td></tr> <tr><td>te</td><td>1</td></tr> <tr><td>cn</td><td>12</td></tr> <tr><td>it</td><td>14</td></tr> <tr><td>nl</td><td>4</td></tr> <tr><td>ta</td><td>2</td></tr> <tr><td>sv</td><td>9</td></tr> <tr><td>th</td><td>3</td></tr> <tr><td>da</td><td>7</td></tr> <tr><td>xx</td><td>1</td></tr> <tr><td>hu</td><td>1</td></tr> <tr><td>cs</td><td>2</td></tr> <tr><td>pt</td><td>9</td></tr> <tr><td>is</td><td>1</td></tr> <tr><td>tr</td><td>1</td></tr> <tr><td>nb</td><td>1</td></tr> <tr><td>af</td><td>1</td></tr> </tbody> </table>	name_original_language	COUNT(m.idOrigLang)	en	4582	ja	16	fr	79	zh	27	es	32	de	27	hi	19	ru	11	ko	11	te	1	cn	12	it	14	nl	4	ta	2	sv	9	th	3	da	7	xx	1	hu	1	cs	2	pt	9	is	1	tr	1	nb	1	af	1
name_original_language	COUNT(m.idOrigLang)																																																				
en	4582																																																				
ja	16																																																				
fr	79																																																				
zh	27																																																				
es	32																																																				
de	27																																																				
hi	19																																																				
ru	11																																																				
ko	11																																																				
te	1																																																				
cn	12																																																				
it	14																																																				
nl	4																																																				
ta	2																																																				
sv	9																																																				
th	3																																																				
da	7																																																				
xx	1																																																				
hu	1																																																				
cs	2																																																				
pt	9																																																				
is	1																																																				
tr	1																																																				
nb	1																																																				
af	1																																																				

Ejemplo 2: ¿Cuántas películas tienen director y cuántas no?

# Tienen director	Tienen director	No tienen director
<pre>SELECT COUNT(m.idMovie) FROM Director m;</pre>		
<pre># No tienen director</pre>		
<pre>SELECT COUNT(id) FROM movie_dataset WHERE director IS NULL;</pre>		

Ejemplo 3: ¿Cuáles son los idiomas en los que están las películas, y cuántas hay en cada lengua (spoken\_languages)?

# Idiomas en los que están las películas	# 1	# 2																																																																																				
<pre>SELECT DISTINCT a.idMovie, b.nameSLang FROM Movie_spoken_languages a, spoken_language b WHERE a.iso_639_1 = b.iso_639_1; # ¿Cuántas películas hay en cada idioma? SELECT msl.iso_639_1, COUNT(msl.iso_639_1) FROM Movie_spoken_languages msl GROUP BY msl.iso_639_1 ORDER BY 2 DESC;</pre>	<table border="1"> <thead> <tr> <th>idMovie</th> <th>nameSLang</th> </tr> </thead> <tbody> <tr><td>1</td><td>868 Afrikaans</td></tr> <tr><td>2</td><td>1123 Afrikaans</td></tr> <tr><td>3</td><td>1372 Afrikaans</td></tr> <tr><td>4</td><td>17654 Afrikaans</td></tr> <tr><td>5</td><td>22600 Afrikaans</td></tr> <tr><td>6</td><td>59961 Afrikaans</td></tr> <tr><td>7</td><td>192136 Afrikaans</td></tr> <tr><td>8</td><td>9839</td></tr> <tr><td>9</td><td>25 العربية</td></tr> <tr><td>10</td><td>85 العربية</td></tr> <tr><td>11</td><td>231 العربية</td></tr> <tr><td>12</td><td>409 العربية</td></tr> <tr><td>13</td><td>435 العربية</td></tr> <tr><td>14</td><td>612 العربية</td></tr> <tr><td>15</td><td>691 العربية</td></tr> <tr><td>16</td><td>708 العربية</td></tr> <tr><td>17</td><td>947 العربية</td></tr> <tr><td>18</td><td>1164 العربية</td></tr> <tr><td>19</td><td>1495 العربية</td></tr> <tr><td>20</td><td>1535 العربية</td></tr> </tbody> </table>	idMovie	nameSLang	1	868 Afrikaans	2	1123 Afrikaans	3	1372 Afrikaans	4	17654 Afrikaans	5	22600 Afrikaans	6	59961 Afrikaans	7	192136 Afrikaans	8	9839	9	25 العربية	10	85 العربية	11	231 العربية	12	409 العربية	13	435 العربية	14	612 العربية	15	691 العربية	16	708 العربية	17	947 العربية	18	1164 العربية	19	1495 العربية	20	1535 العربية	<table border="1"> <thead> <tr> <th>iso_639_1</th> <th>COUNT(msl.iso_639_1)</th> </tr> </thead> <tbody> <tr><td>en</td><td>4482</td></tr> <tr><td>fr</td><td>436</td></tr> <tr><td>es</td><td>351</td></tr> <tr><td>de</td><td>262</td></tr> <tr><td>it</td><td>188</td></tr> <tr><td>ru</td><td>185</td></tr> <tr><td>zh</td><td>107</td></tr> <tr><td>ja</td><td>97</td></tr> <tr><td>pt</td><td>68</td></tr> <tr><td>ar</td><td>67</td></tr> <tr><td>pl</td><td>53</td></tr> <tr><td>la</td><td>52</td></tr> <tr><td>cn</td><td>48</td></tr> <tr><td>hi</td><td>48</td></tr> <tr><td>hu</td><td>41</td></tr> <tr><td>th</td><td>40</td></tr> <tr><td>cs</td><td>38</td></tr> <tr><td>he</td><td>33</td></tr> <tr><td>ko</td><td>31</td></tr> <tr><td>sv</td><td>22</td></tr> </tbody> </table>	iso_639_1	COUNT(msl.iso_639_1)	en	4482	fr	436	es	351	de	262	it	188	ru	185	zh	107	ja	97	pt	68	ar	67	pl	53	la	52	cn	48	hi	48	hu	41	th	40	cs	38	he	33	ko	31	sv	22
idMovie	nameSLang																																																																																					
1	868 Afrikaans																																																																																					
2	1123 Afrikaans																																																																																					
3	1372 Afrikaans																																																																																					
4	17654 Afrikaans																																																																																					
5	22600 Afrikaans																																																																																					
6	59961 Afrikaans																																																																																					
7	192136 Afrikaans																																																																																					
8	9839																																																																																					
9	25 العربية																																																																																					
10	85 العربية																																																																																					
11	231 العربية																																																																																					
12	409 العربية																																																																																					
13	435 العربية																																																																																					
14	612 العربية																																																																																					
15	691 العربية																																																																																					
16	708 العربية																																																																																					
17	947 العربية																																																																																					
18	1164 العربية																																																																																					
19	1495 العربية																																																																																					
20	1535 العربية																																																																																					
iso_639_1	COUNT(msl.iso_639_1)																																																																																					
en	4482																																																																																					
fr	436																																																																																					
es	351																																																																																					
de	262																																																																																					
it	188																																																																																					
ru	185																																																																																					
zh	107																																																																																					
ja	97																																																																																					
pt	68																																																																																					
ar	67																																																																																					
pl	53																																																																																					
la	52																																																																																					
cn	48																																																																																					
hi	48																																																																																					
hu	41																																																																																					
th	40																																																																																					
cs	38																																																																																					
he	33																																																																																					
ko	31																																																																																					
sv	22																																																																																					

Ejemplo 4: ¿Cuáles son las compañías de producción que tienen el mayor número de películas (production companies)?

```
SELECT a.pCompanyName, COUNT(b.pCompanyId)
FROM production_companies a, Movie_production_companies b
WHERE a.pCompanyId = b.pCompanyId
GROUP BY a.pCompanyId
ORDER BY 2 DESC
LIMIT 10;
```

pCompanyName	'COUNT(b.pCompanyId)'
1 Warner Bros.	319
2 Universal Pictures	311
3 Paramount Pictures	285
4 Twentieth Century Fox Film Corporation	222
5 Columbia Pictures	201
6 New Line Cinema	165
7 Metro-Goldwyn-Mayer (MGM)	122
8 Touchstone Pictures	118
9 Walt Disney Pictures	114
10 Relativity Media	102

Ejemplo 5: ¿Cuáles son las personas que tienen el mayor número de Jobs (Crew)?

```
SELECT a.name, COUNT(b.job)
FROM Persona a, crew b
WHERE a.idPerson = b.idPerson
GROUP BY a.idPerson
ORDER BY 2 DESC
LIMIT 10;
```

name	'COUNT(b.job)'
1 Robert Rodriguez	104
2 Steven Spielberg	84
3 Avy Kaufman	83
4 Mary Vernieu	82
5 Deborah Aquila	75
6 Hans Zimmer	71
7 James Newton Howard	69
8 Harvey Weinstein	68
9 Bob Weinstein	67
10 Tricia Wood	67

Ejemplo 6: ¿Cuáles son los directores que no constan como director en la Crew?

```
SELECT DISTINCT a.name, b.job, b.department
FROM Persona a, crew b
WHERE a.idPerson = b.idPerson AND b.department = 'Directing' AND b.job != 'Director';
```

name	job	department
1 Douglas Aarniokoski	First Assistant Director	Directing
2 Fernando Altschul	First Assistant Director	Directing
3 Jayne-Ann Tenggren	Script Supervisor	Directing
4 Lee Unkrich	Co-Director	Directing
5 Cherylanne Martin	Assistant Director	Directing
6 Bruce Moriarty	Assistant Director	Directing
7 Susan Malerstein	Script Supervisor	Directing
8 Tony Adler	Assistant Director	Directing
9 Ana Maria Quintana	Script Supervisor	Directing
10 Marilyn Giardino	Script Supervisor	Directing
11 Rosemary C. Cremona	Assistant Director	Directing
12 Anders Refn	Assistant Director	Directing
13 Carolina Sascha Cogez	Assistant Director	Directing
14 Jean Bourne	Script Supervisor	Directing
15 Franck Lebreton	Assistant Director	Directing
16 Olivier Horlait	Assistant Director	Directing
17 Aim\u00e9e Peyronnet	Assistant Director	Directing
18 Fr\u00e9d\u00e9ric Garson	Second Assistant Director	Directing
19 Kelly L'Estrange	First Assistant Director	Directing
20 Camille Lipmann	Assistant Director	Directing

Ejemplo 7: ¿Cuáles son las jobs qué tienen el mayor número de personas (crew)?

```
SELECT a.job, COUNT(b.idPerson)
FROM crew a, Persona b
WHERE a.idPerson = b.idPerson
GROUP BY a.job
ORDER BY 2 DESC
LIMIT 10;
```

job	COUNT(b.idPerson)
1 Producer	10195
2 Executive Producer	6177
3 Director	5162
4 Screenplay	5008
5 Editor	4696
6 Casting	4444
7 Director of Photography	3673
8 Art Direction	3336
9 Original Music Composer	3152
10 Production Design	2835

## 9. Conclusiones

Este proyecto se realizó bajo estándares de modelado de bases de datos como normalización, diseño conceptual y lógica. Y tras el arduo trabajo de todos los integrantes del grupo, pudimos lograr todos los objetivos planteados al inicio del desarrollo. Tuvimos que afrontar varios retos a lo largo del proyecto. Sin embargo, la guía de los profesores acompañantes y nuestra creatividad en La resolución de problemas nos permitió cumplir con los requisitos establecidos.

Todos los pasos apropiados para completar el proyecto Como resultado, hemos logrado un progreso significativo en el modelado y la toma de decisiones para las entradas de la base de datos, y hemos aprendido nuevas estrategias de programación que se pueden aplicar a situaciones reales. - casos del mundo. Desarrollar este proyecto definitivamente lo acercará un paso más a convertirse en un profesional seguro cuando trabaje en proyectos futuros.

He aprendido que los datos deben procesarse correctamente antes de cada operación para mantener la coherencia y evitar errores. Además, entendemos la importancia de las herramientas a la hora de trabajar con bases de datos. Y puedo concluir que todo lo que he visto en este ciclo es muy importante ya que se utilizó en este proyecto de integración, fíjate que fue un reto para nuestros estudiantes.