



**UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA**

*La Universidad Católica de Loja*

**Informe de Proyecto Final**  
**Fundamentos de Bases de Datos**

Carlos Iván Montero Torres

Octubre 2022 – Febrero 2023

## TABLA DE CONTENIDO

|  |    |
|--|----|
| Introducción .....                                   | 3  |
| Modelos .....  | 4  |
| Dependencias Funcionales de la Tabla Universal ..... | 5  |
| Modelo Conceptual .....                              | 7  |
| Modelo Lógico .....                                  | 8  |
| Modelos Físico .....                                 | 9  |
| Normalización .....                                  | 10 |
| First Normal Form (1NF) .....                        | 10 |
| Second Normal Form (2NF) .....                       | 11 |
| Third Normal Form (3NF) .....                        | 11 |
| Proceso .....  | 11 |
| Procedimiento .....                                  | 16 |
| Limpieza .....                                       | 20 |
| Consultas .....                                      | 21 |
| SQL .....  | 23 |
| Conclusiones .....                                   | 52 |

## Introducción

Vivimos en una era digital donde existen grandes cantidades de flujos de datos y la mejor manera para almacenar esta información, de una manera estructurada es mediante bases de datos, las cuales nos permiten ingresar, manipular, almacenar y/o eliminar grandes cantidades de información. Los datos no deben ser almacenados de cualquier manera, se debe seguir las diferentes reglas y realizar los procesos correspondientes para la verificación de un proceso de modelado y carga exitoso. En este informe, se detallará como han sido aplicado los diferentes conocimientos adquiridos durante este ciclo académico de la materia de Fundamentos de Base de Datos. Se realizará la importación de un archivo CSV llamado *movie\_dataset* utilizando el IDE (del inglés integrated development environment) llamado DataGrip de JetBrains. Utilizaremos el sistema de gestión de base de datos MySQL y aplicaremos las distintas fases de modelado, además de la inserción, limpieza, y carga, y explotación de datos. Entre los conceptos utilizados se encuentran los temas estudiados anteriormente en la materia como diagramas de Entidad-Relación, modelos conceptuales, lógicos y físicos, sentencias de creación, inserción y búsqueda en SQL.

## Contextualización

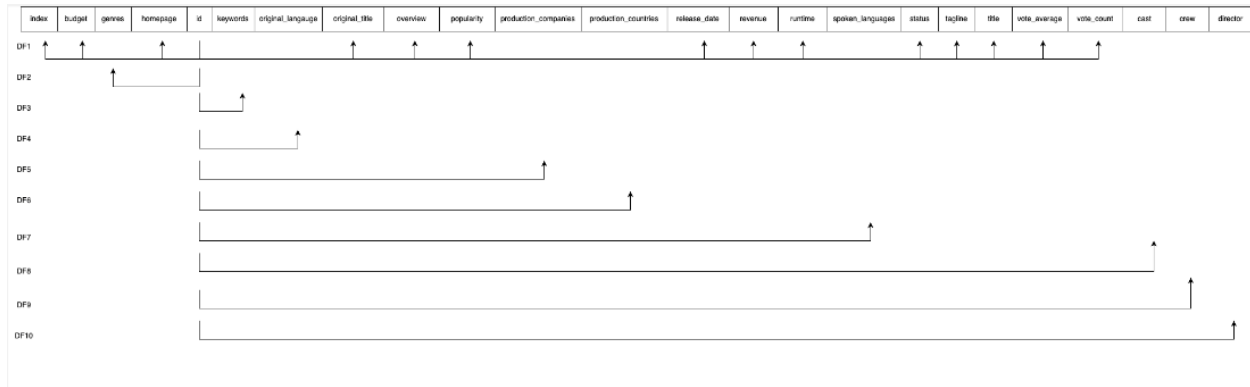
El problema por resolver está basado en un dataset de películas el cual es obtenido del repositorio publico Datasets del usuario rashida048 en Github, el cual tiene la dirección URL <https://github.com/rashida048/Datasets>. En este repositorio podemos encontrar varios archivos CSV de diferentes campos para realizar prácticas con ellos. Cinco archivos fueron subidos en el último año, pero la gran mayoría de los archivos han sido subido hace dos años, pero aun así siguen siendo relevantes.

En el CSV que nosotros vamos a trabajar, movie\_dataset.csv, llamado Movie Dataset, tenemos una tabla de 24 columnas en donde se almacenan datos importantes y relevantes de películas que han sido lanzadas hasta la creación del CSV.

Para facilitar la visualización de movie\_dataset, decidimos utilizar la herramienta de Excel en Windows que permite consultar y visualizar todos los datos recibidos, al igual que la aplicación Numbers de Apple que permite realizar cálculos de las diferentes columnas.

## Modelos

### Dependencias Funcionales de la Tabla Universal



*Figura 1: Descripción gráfica de las dependencias existentes en las columnas del dataset*

Las dependencias funcionales son de mi primer intento de realizar relaciones entre las columnas de las diferentes columnas de la tabla universal. Arrancamos desde la clave primaria que decidí que sería el id debido a que este era un identificador que no se repetía, y a diferencia del index, contenía un valor con algún sentido, ya sea como por ejemplo que haya sido sacado de una tabla más grande. Se notó un patrón en las columnas que contenían valores JSON, los cuales contenían arreglos de objetos, los cuales formarían parte de relaciones muchos a muchos. Además, pude encontrar tres columnas que contenían valores que se repetían, como `original_language`, `status` y `director`. Se notaría que un valor de estas columnas podía aparecer en muchas películas, pero películas únicamente podían tener un valor de estos. Se empezó a dar forma a la tabla universal y a las dependencias funcionales.

Pero al continuar y seguir analizando, existía un caso especial en una de las columnas de tipo JSON. En el caso de `crew`, teníamos una tabla que tenía más dependencias dentro de sus columnas. Es por esto que se realizó un nuevo análisis, comparando los valores de la columna con valores fuera de ella y se pudo evidenciar que existían nuevas dependencias que

debíamos tomar en cuenta. Por ejemplo, tenemos la columna de cast, en el cual tenemos nombres de actores, y aunque no están limpios (no separados por una manera continua, como por comas o comillas), guardan información de personas. Y en caso de los directores, ellos también son personas. Es por esto que de aquí nace la creación de la entidad Persona, la cual busca darle solución a estas dependencias y relaciones, en busca de juntar valores similares para lograr un almacenamiento estructurado de la data.

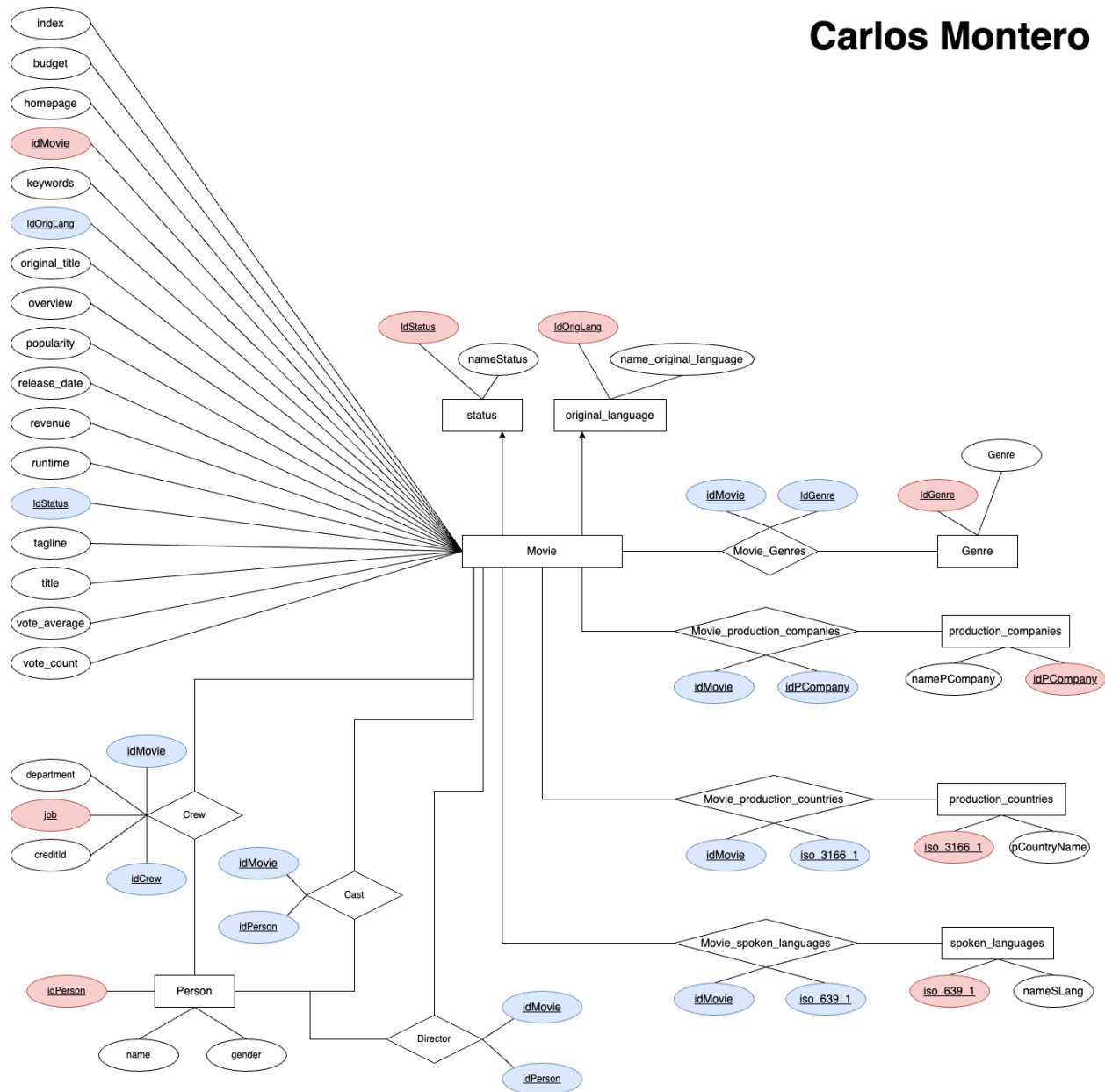
Determine que la llave primaria id nos daría las siguientes dependencias, sin incluir las del caso Persona que he comentado anteriormente.

1. Id -> {index, budget, homepage, original\_title, overview, popularity, release\_date, revenue, runtime, status, tagline, title, vote\_average, vote\_count}
2. Id -> genres
3. Id -> keywords
  - a. Se tomó en cuenta que debido a que existen varios keywords dentro de una String en cada fila (también existen filas con valores nulos), existía la relación de muchos a muchos pero debido a las grandes cantidades de nombres y diferentes tipos de casos de separación de nombres, sería un trabajo muy largo para completar sin la ayuda de un servicio web como el de reconocimiento de entidades de MeaningCloud. Es por esto que se lo incluyó junto a las demás columnas en la primera dependencia.
4. Id -> original\_language
5. Id -> production\_companies
6. Id -> production\_countries
7. Id -> spoken\_languages
8. Id -> cast
9. Id -> crew

## Modelo Conceptual

Después de haber establecido las dependencias funcionales 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.

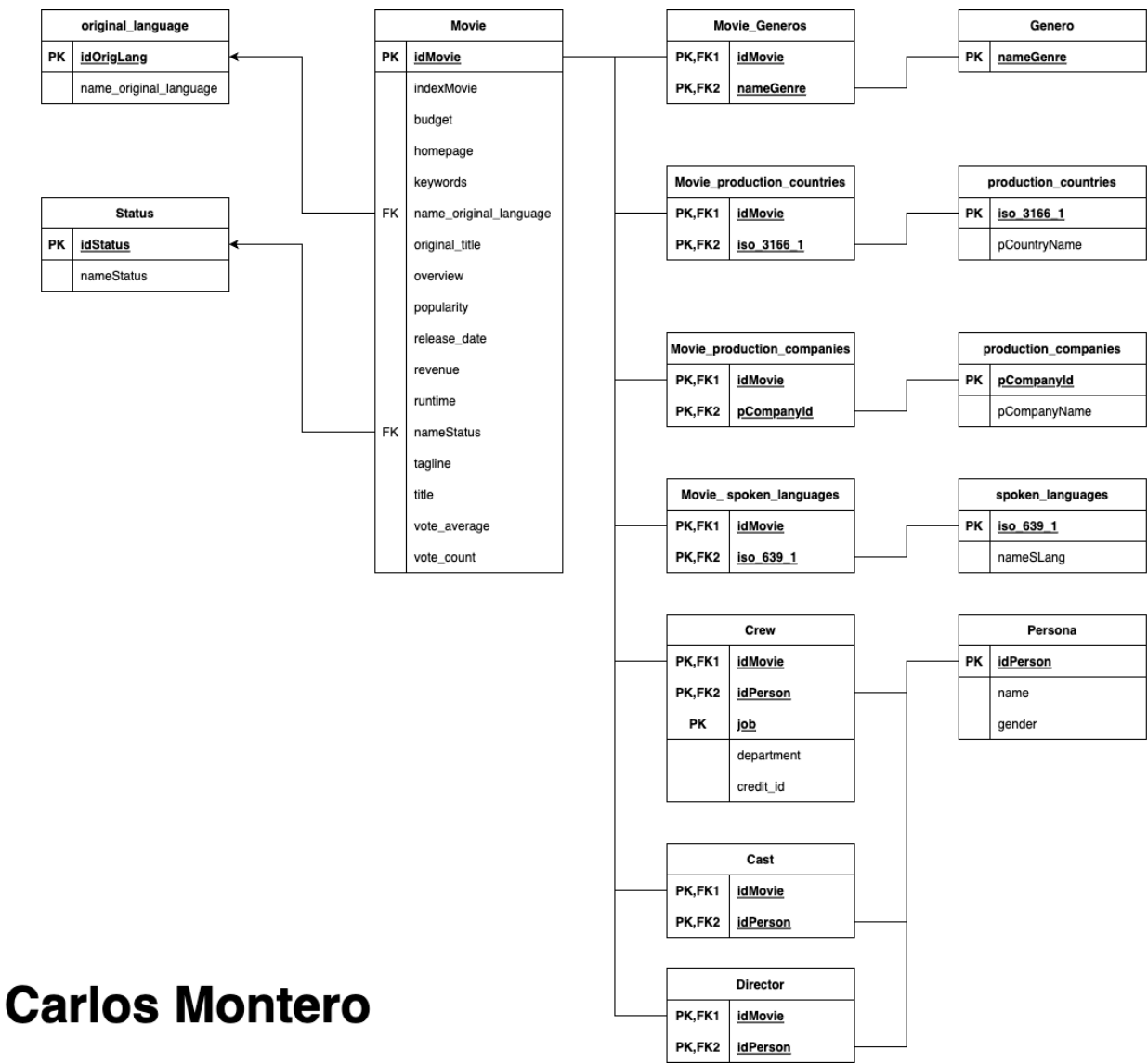
**Carlos Montero**



*Figura 2: Representación del Modelo Conceptual*

Modelo Lógico

Con el modelo conceptual ya realizado, se seleccionan los atributos apropiados para cada tabla y se crea un prototipo de estas basado en las entidades y relaciones identificadas. En cada tabla, se declara una clave primaria y, en su caso, una clave foránea, esto con el objetivo de tener una base sólida para desarrollar el esquema de la base de datos.



Carlos Montero

Figura 3: Representación gráfica del Modelo Lógico



Modelos Físico

Una vez terminado el modelo lógico pasamos a la fase final que es la creación del modelo físico donde se especificó las tablas, columnas, claves principales y a su vez sus claves externas o foráneas con sus respectivas relaciones. Con este modelo podemos pasar a generar las respectivas sentencias DDL.

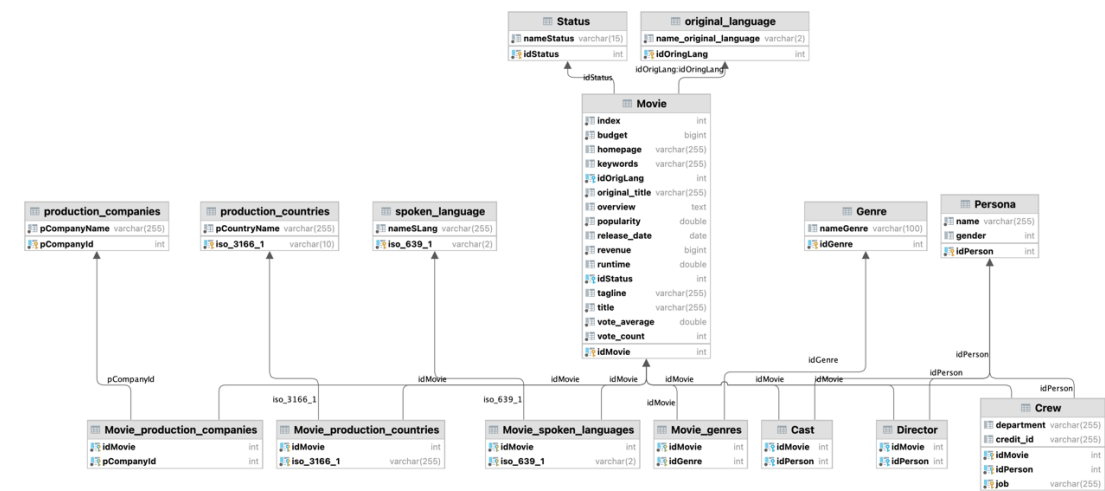
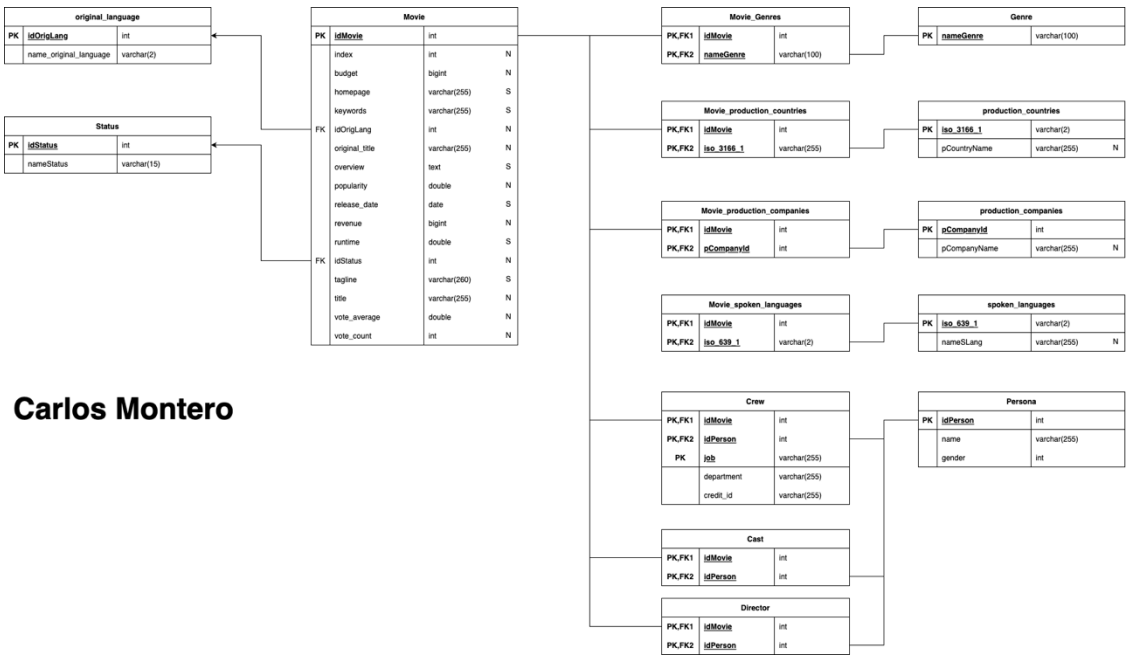


Figura 4: Representación gráfica del Modelo Físico



Carlos Montero

Figura 5: Representación gráfica del Modelo Físico 2

## Normalización

La normalización es un proceso en el diseño de bases de datos que tiene como objetivo minimizar la redundancia y la dependencia funcional dentro de las tablas de una base de datos. Se logra a través de la descomposición de tablas grandes en unas más pequeñas y específicas, y mediante la estructuración de estas tablas utilizando relaciones. Tiene como fin asegurar de que la información se almacene de manera eficiente, sin duplicidad y de forma que sea fácilmente accesible. Este proceso se realiza en la estructura de las bases de datos para mejorar la eficiencia, organización e integridad. En otras palabras, este proceso consiste en la división de información en diferentes tablas y las relaciones entre sí. Seguimos los siguientes pasos para normalizar los datos del archivo CSV.

### First Normal Form (1NF)

1. Una relación en donde la intersección de cada fila y columna contiene un y solo un valor.
2. La fase antes de 1NF es Unnormalized Form (UNF) la cual es una tabla que contiene uno más grupos repetidos.
3. Para transformar la tabla no normalizada a la primera forma normal, identificamos y eliminamos los grupos que se repiten dentro de la tabla.
4. Un grupo repetitivo es un atributo, o grupo de atributos, dentro de una tabla que ocurre con múltiples valores para una sola ocurrencia de los atributos clave designados para esa tabla.

## Second Normal Form (2NF)

1. Una relación que está en la primera forma normal y cada atributo que no es de clave principal depende funcionalmente de la clave principal.
2. La normalización de las relaciones 1NF a 2NF implica la eliminación de dependencias parciales. Si existe una dependencia parcial, eliminamos los atributos parcialmente dependientes de la relación colocándolos en una nueva relación junto con una copia de su determinante.

## Third Normal Form (3NF)

1. Una relación que está en primera y segunda forma normal y en la que ningún atributo que no sea de clave principal depende transitivamente de la clave principal.
2. La normalización de las relaciones 2NF a 3NF implica la eliminación de las dependencias transitivas.
3. Si existe una dependencia transitiva, eliminamos los atributos transitivamente dependientes de la relación colocando los atributos en una nueva relación junto con una copia del determinante.

## Proceso

Primero identificamos y eliminamos grupos repetitivos dentro de la tabla. Un grupo repetitivo es un atributo, o grupo de atributos, dentro de una tabla que ocurre con múltiples valores para una sola ocurrencia de los atributos clave designados para esa tabla.

### **Tabla Uno a Muchos (1 : N)**

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

3. Una película puede tener un Status, pero un Status puede representar a muchas películas.

- Para solucionar, se crea una tabla separada que se llame Status.
- La llave primaria es statusName.

Como solución, utilizamos el statusName como llave foránea en Movie. Director es una columna con el nombre de un único director.

- Una película puede tener un Director, pero un Director puede tener 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 Movie.

Original\_language es una columna con el nombre del lenguaje original de la Movie.

- Una película tiene un Original\_language, pero un Original\_language puede tener muchas películas.
- Para solucionar, se crea una tabla separada que se llame original\_language.
- La llave primaria es origLanName.
- Como solución, utilizamos el origLanName como llave foránea en original\_language.

### **Tabla Muchos a Muchos (N : N)**

En nuestro caso, 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) keywords.
  - 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. Existe 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.
- spoken\_languages contiene un String de JSON que contiene un iso\_639\_1 y un name que puede contener 0 a n spoken\_languages.
    - Para solucionar, cada columna debe contener un solo valor.
    - En este caso tenemos una lista de spoken\_languages. Existen múltiples valores.
    - La solución es crear una tabla separada que se llame spoken\_languages.

- Tiene dos atributos, `iso_639_1` y `name` los cuales los nombramos, `iso_639_1` el cual es la primary key y `spokLangName`.
  - Una Movie puede tener muchos `spoken_languages`, y un `spoken_languages` puede aparecer en muchas Movie.
  - La relación (muchos a muchos) se soluciona con una tabla llamada `Movie_Spoken_languages` utilizando la llave primaria de cada uno.
- 
- **'Cast', 'Crew' y 'Director'** contienen los nombres de integrantes que participan de alguna forma en la película. Estos necesitan ser normalizados en una tabla con datos comunes, en este caso llamada **Persona**.
  - Se crea una tabla Persona que incluya los atributos comunes a todas las entidades, entre los cuales están: `gender`, `idCrew` y `name`.
- 
1. Ya que la relacion Persona-Movie es muchos a muchos, por consecuencia se necesita otra tabla que contenga el `idMovie` e `idCrew`.
  2. Es momento de crear tablas temporales de `'cast'`, `'crew'` y `'director'`, para la migración de datos hacia la tabla persona.
  3. Antes de migrar los datos a la tabla Persona, es importante realizar una limpieza de datos para garantizar que los datos sean precisos y coherentes.

## Procedimiento

### 1. 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, primero se debe entrar se debe ubicar en la parte superior izquierda, donde se encuentra los íconos, señalar la que es para crear un Schema como se puede ver en el siguiente Anexo:

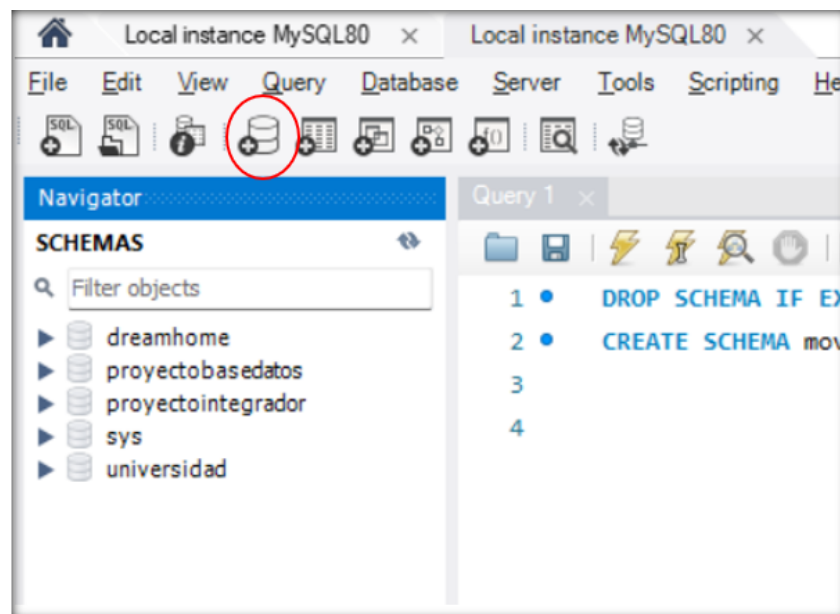
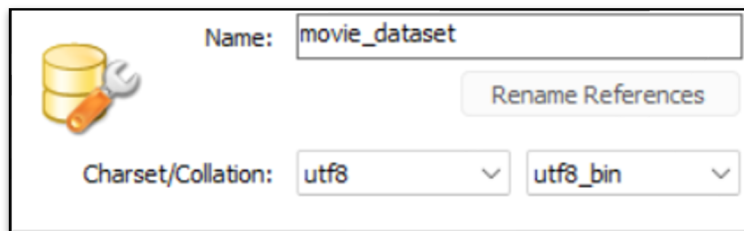


Figura 6: Representación de la barra de opciones perteneciente a Workbench



Subsecuentemente se le daría nombre al Schema en este caso “movie\_dataset”, donde se podría modificar varios aspectos, en nuestro Schema utilizamos un Charset (codificación de datos) de “utf8”:



*Figura 7: Representación de la funcionalidad de creación de Schema automáticamente*

## OPCIÓN 2 – MEDIANTE SENTENCIAS SQL

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

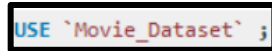
```
CREATE DATABASE IF NOT EXISTS `movie_dataset` DEFAULT CHARACTER SET utf8 ;
```

*Figura 8: Sentencia de CREATE DATABASE*

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

## 2. 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:



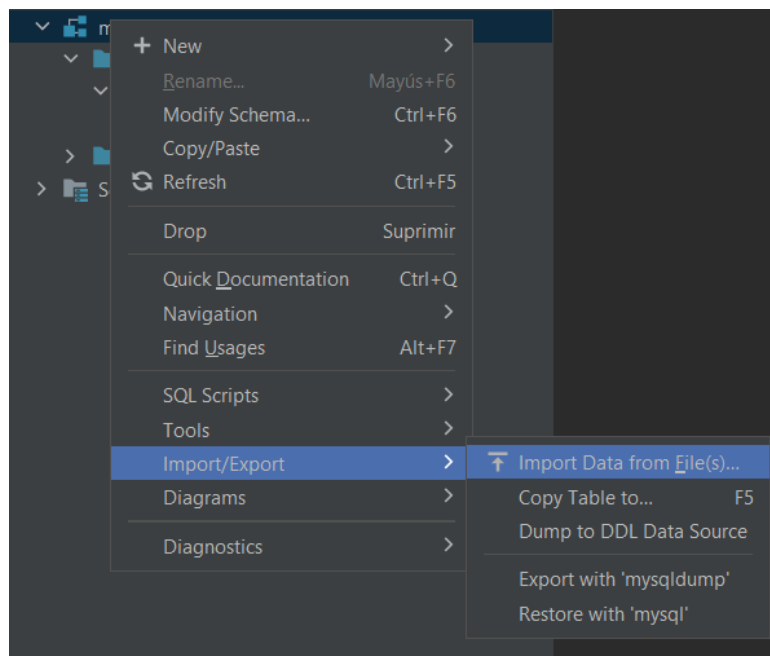
```
USE `Movie_Dataset` ;
```

*Figura 9: Utilizar el DATABASE*

### 3. 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.

A continuación, se muestran los pasos seguidos en la siguiente imagen:



*Figura 10: Importacion de archivos en DataGrip*

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

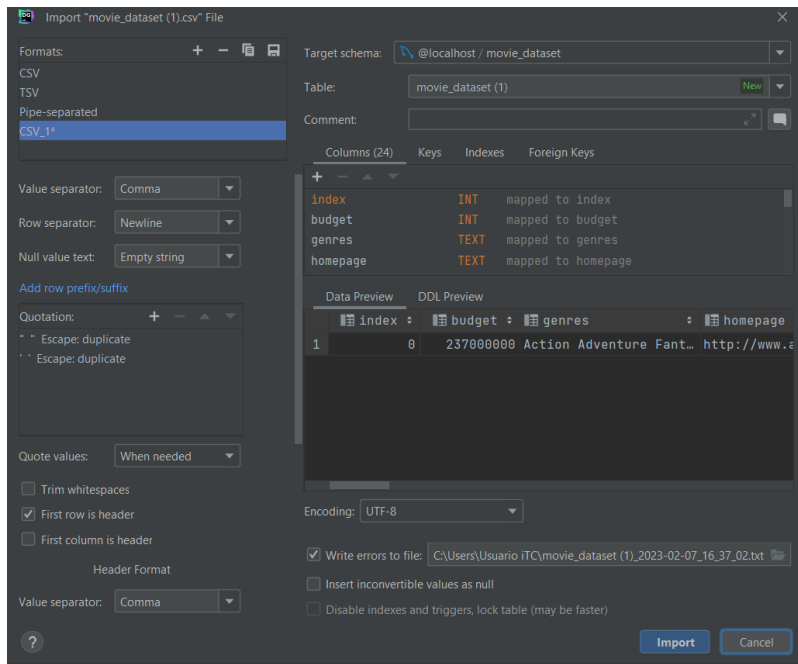


Figura 11: Importación de archivos en DataGrip

#### 4. 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.

## Limpieza

En la columna crew, tenemos un JSON mal formado en el cual, era necesario convertir comillas dobles en comillas simples. Debido a casos especiales en los que se tenía que cambiar caracteres o agregar unos, era necesario incluir el carácter junto a posiciones que no cambian, para que, de esta manera, se logre realizar un reemplazo (REPLACE) a todos los objetos y filas. Era también necesario tomar en cuenta el caso especial cuando se acababa un arreglo y en tal caso no podíamos utilizar el mismo caso cuando se cerraba una llave, seguido por una coma ya que ahora era necesario especificar que le seguía un corchete.

```
SELECT id,
JSON_VALID(CONVERT (
REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(crew,
''', '\'),
'{\ ', '{\"'),
'\': \ ', ': '),
'\ ', \ ', ', '),
'\': ', ': '),
', \ ', ', ')
USING UTF8mb4 )) AS Valid_YN,
CONVERT (
REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(crew,
''', '\'),
'{\ ', '{\"'),
'\': \ ', ': '),
'\ ', \ ', ', '),
'\': ', ': '),
', \ ', ', ')
USING UTF8mb4 ) AS crew_new,
crew AS crew_old
FROM movie_dataset ;
```

*Figura 12: Código de limpieza de la columna crew*

## Consultas

```
784 ✓ SELECT * FROM Movie;

Output #####

1-500 of 501+ | Tx: Auto | DDL

idMovie | index | budget | homepage | keywords
1 | 5 | 3766 | 4000000 | <null> | hotel new year's eve witch bet hotel ...
2 | 11 | 2912 | 11000000 | http://www.starwars.com/films/star-wa... | android galaxy hermit death star ligh...
3 | 12 | 328 | 94000000 | http://movies.disney.com/finding-nemo | father son relationship harbor underw...
4 | 13 | 809 | 55000000 | <null> | vietnam veteran hippie mentally disab...
5 | 14 | 2516 | 15000000 | http://www.dreamworks.com/ab/ | male nudity female nudity adultery mi...
6 | 16 | 2799 | 12800000 | <null> | individual dancing usa robbery factor...
7 | 18 | 322 | 90000000 | <null> | clone taxi cyborg egypt future
8 | 19 | 2638 | 92620000 | <null> | man vs machine underground world inve...
9 | 20 | 4023 | 0 | http://www.clubcultura.com/clubcine/c... | farewell responsibility dying and dea...
10 | 22 | 199 | 140000000 | http://disney.go.com/disneyvideos/liv... | exotic island blacksmith east india t...
11 | 24 | 828 | 30000000 | http://www.miramax.com/movie/kill-bil... | japan coma martial arts kung fu under...
12 | 25 | 557 | 72000000 | <null> | sniper marine corps saudi arabia petr...
13 | 28 | 1525 | 31500000 | http://www.apocalypsenow.com | guerrilla river vietnam vietcong camb...
14 | 33 | 2444 | 16000000 | <null> | prostitute sheriff haunted planet air...
```

*Figura 13: Consulta 1*

788 ✓ `SELECT title, vote_average FROM Movie WHERE budget > 100000000 AND runtime > 100;`

Output MoviesProyectoBaseDeDatos.Movie

220 rows

|    | title  | vote_average |
|----|--|--------------|
| 1  | Pirates of the Caribbean: The Curse of the Black Pearl | 7.5          |
| 2  | Pirates of the Caribbean: Dead Man's Chest             | 7            |
| 3  | War of the Worlds                                      | 6.2          |
| 4  | Armageddon   | 6.4          |
| 5  | Gladiator  | 7.9          |
| 6  | Charlie and the Chocolate Factory                      | 6.7          |
| 7  | The Dark Knight  | 8.2          |
| 8  | Ocean's Twelve   | 6.4          |
| 9  | Minority Report  | 7.1          |
| 10 | Indiana Jones and the Kingdom of the Crystal Skull     | 5.7          |
| 11 | King Kong  | 6.6          |
| 12 | Batman Begins  | 7.5          |
| 13 | Pirates of the Caribbean: At World's End               | 6.9          |

Figura 14: Consulta 2

790 ✓ `SELECT * FROM Movie_spoken_languages WHERE idMovie = 19995;`

Output MoviesProyectoBaseDeDatos.Movie\_spoken\_languages

2 rows

|   | idMovie | iso_639_1 |
|---|---------|-----------|
| 1 | 19995   | en        |
| 2 | 19995   | es        |

Figura 15: Consulta 3

792 ✓ `SELECT *`  
793 `FROM Movie_production_companies mpc`  
794 `WHERE idMovie = 19995;`

Output MoviesProyectoBaseDeDatos.Movie\_production\_companies

4 rows

|   | idMovie | mpcCompanyId |
|---|---------|--------------|
| 1 | 19995   | 289          |
| 2 | 19995   | 306          |
| 3 | 19995   | 444          |
| 4 | 19995   | 574          |

Figura 16: Consulta 4

The screenshot shows a SQL query editor with the following query:

```

792 SELECT mpc.idMovie, pd.pCompanyName
793 FROM Movie_production_companies mpc, production_companies pd
794 WHERE mpc.pCompanyId = pd.pCompanyId AND idMovie = 19995;

```

Below the query, the 'Output' tab shows 'Result 326' with 4 rows. The results are displayed in a table with two columns: 'idMovie' and 'pCompanyName'.

|   | idMovie | pCompanyName                           |
|---|---------|--|
| 1 | 19995   | Ingenious Film Partners                |
| 2 | 19995   | Twentieth Century Fox Film Corporation |
| 3 | 19995   | Dune Entertainment                     |
| 4 | 19995   | Lightstorm Entertainment               |

Figura 17: Consulta 5

## SQL

```

# Carlos Montero
#

DROP DATABASE IF EXISTS MoviesProyectoBaseDeDatos;
CREATE DATABASE MoviesProyectoBaseDeDatos;
USE MoviesProyectoBaseDeDatos;

# CREAR TABLAS

CREATE TABLE original_language(
    idOrigLang INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
    name_original_language VARCHAR(2) NOT NULL
);

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

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)
);

CREATE TABLE Genre(
    idGenre int PRIMARY KEY AUTO_INCREMENT NOT NULL,
    nameGenre VARCHAR(100)
);

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

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

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

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

# MUCHOS A MUCHOS

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)
);

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)
);

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)
);

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)
);

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)
);

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

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

# DROP TABLES
DROP TABLE IF EXISTS original_language;
DROP TABLE IF EXISTS Status;
DROP TABLE IF EXISTS Movie;
DROP TABLE IF EXISTS Genre;
DROP TABLE IF EXISTS production_countries;
DROP TABLE IF EXISTS production_companies;
DROP TABLE IF EXISTS spoken_language;
DROP TABLE IF EXISTS Persona;
#
DROP TABLE IF EXISTS Crew;
DROP TABLE IF EXISTS Cast;
DROP TABLE IF EXISTS Director;
# MUCHOS A MUCHOS
DROP TABLE IF EXISTS Movie_genres;
DROP TABLE IF EXISTS Movie_production_countries;
DROP TABLE IF EXISTS Movie_production_companies;
DROP TABLE IF EXISTS Movie_spoken_languages;

```

```
#####

# POBLACION

#####

# 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();

SELECT * FROM original_language;

#####

# 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();

SELECT * FROM Status;

#####

# Movie

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
    SELECT id,`index`,budget,homepage, 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 `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,overvie
w,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 $$
DELIMITER ;

CALL TablaMovie ();

SELECT * FROM Movie;

#####

# 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;

        -- 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();

SELECT * FROM production_countries;

#####
# 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;

  -- 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();

SELECT * FROM production_companies;

#####

# 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;

        -- 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;

```



```

    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();

SELECT * FROM spoken_language;

#####

# 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;
    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();

#####

# MUCHOS A MUCHOS

#####

# 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;

    -- 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();

SELECT COUNT(*) FROM Movie_production_companies;

#####

# 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;

  -- 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();

SELECT COUNT(*) FROM Movie_production_countries;

#####

# 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 @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();  
  
SELECT COUNT(*) FROM Movie_spoken_languages;  
  
#####  
# 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();

SELECT * FROM Persona;

#####

# Crew

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;

  -- 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();

SELECT COUNT(*) FROM Crew;

#####

# 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
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();

SELECT COUNT(*) FROM Director;

```

```
#####

# CONSULTAS

SELECT * FROM Movie;

SELECT title, vote_average FROM Movie WHERE budget > 100000008 AND runtime > 100;

SELECT * FROM Movie_spoken_languages WHERE idMovie = 19995;

SELECT mpc.idMovie, pd.pCompanyName
FROM Movie_production_companies mpc, production_companies pd
WHERE mpc.pCompanyId = pd.pCompanyId AND idMovie = 19995;

#####

# 1

SELECT ol.name_original_language, COUNT(m.idOrigLang)
FROM Movie m, original_language ol
WHERE m.idOrigLang = ol.idOringLang
GROUP BY m.idOrigLang;

# 3

SELECT COUNT (DISTINCT p.name)
FROM Persona p;

# 4

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;

# 5

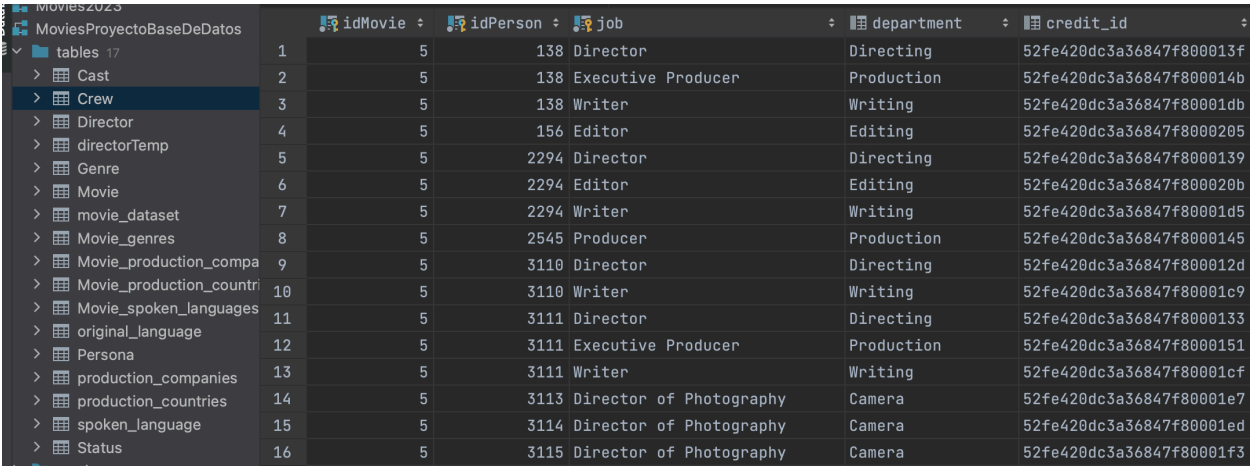
SELECT mpc.pCompanyId, COUNT(mpc.idMovie)
FROM Movie_production_companies mpc
GROUP BY mpc.pCompanyId
ORDER BY 2 DESC;

# 8

SELECT mp.job, COUNT(mp.idCrew)
FROM Movie_Persona mp
GROUP BY mp.job;

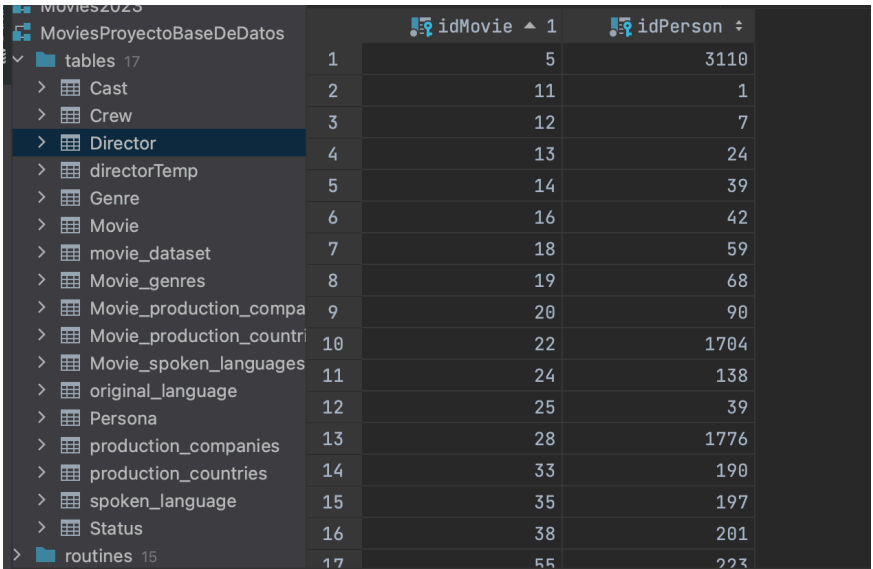
#####
```

Población



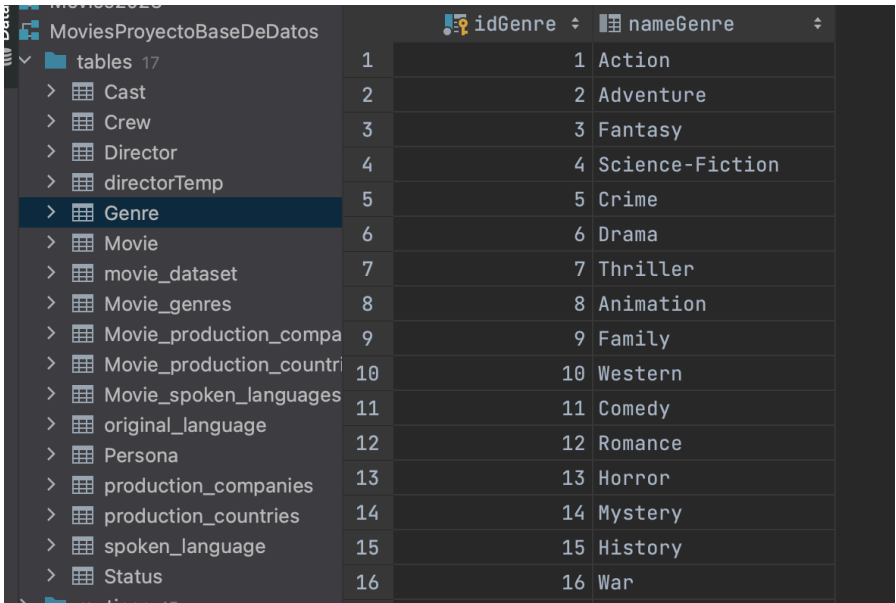
|    | idMovie | idPerson | job                     | department | credit_id                |
|----|---------|----------|-------------------------|------------|--------------------------|
| 1  | 5       | 138      | Director                | Directing  | 52fe420dc3a36847f800013f |
| 2  | 5       | 138      | Executive Producer      | Production | 52fe420dc3a36847f800014b |
| 3  | 5       | 138      | Writer                  | Writing    | 52fe420dc3a36847f80001db |
| 4  | 5       | 156      | Editor                  | Editing    | 52fe420dc3a36847f8000205 |
| 5  | 5       | 2294     | Director                | Directing  | 52fe420dc3a36847f8000139 |
| 6  | 5       | 2294     | Editor                  | Editing    | 52fe420dc3a36847f800020b |
| 7  | 5       | 2294     | Writer                  | Writing    | 52fe420dc3a36847f80001d5 |
| 8  | 5       | 2545     | Producer                | Production | 52fe420dc3a36847f8000145 |
| 9  | 5       | 3110     | Director                | Directing  | 52fe420dc3a36847f800012d |
| 10 | 5       | 3110     | Writer                  | Writing    | 52fe420dc3a36847f80001c9 |
| 11 | 5       | 3111     | Director                | Directing  | 52fe420dc3a36847f8000133 |
| 12 | 5       | 3111     | Executive Producer      | Production | 52fe420dc3a36847f8000151 |
| 13 | 5       | 3111     | Writer                  | Writing    | 52fe420dc3a36847f80001cf |
| 14 | 5       | 3113     | Director of Photography | Camera     | 52fe420dc3a36847f80001e7 |
| 15 | 5       | 3114     | Director of Photography | Camera     | 52fe420dc3a36847f80001ed |
| 16 | 5       | 3115     | Director of Photography | Camera     | 52fe420dc3a36847f80001f3 |

Figura 18: Poblacion de la table Crew



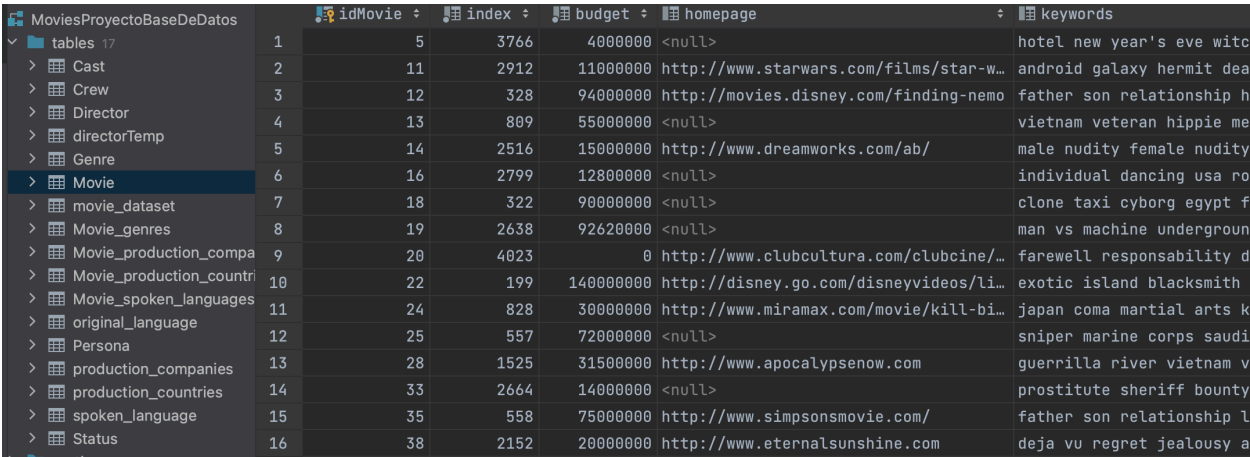
|    | idMovie | idPerson | credit_id |
|----|---------|----------|-----------|
| 1  | 5       | 3110     |           |
| 2  | 11      | 1        |           |
| 3  | 12      | 7        |           |
| 4  | 13      | 24       |           |
| 5  | 14      | 39       |           |
| 6  | 16      | 42       |           |
| 7  | 18      | 59       |           |
| 8  | 19      | 68       |           |
| 9  | 20      | 90       |           |
| 10 | 22      | 1704     |           |
| 11 | 24      | 138      |           |
| 12 | 25      | 39       |           |
| 13 | 28      | 1776     |           |
| 14 | 33      | 190      |           |
| 15 | 35      | 197      |           |
| 16 | 38      | 201      |           |
| 17 | 55      | 223      |           |

Figura 19: Poblacion de la table Director



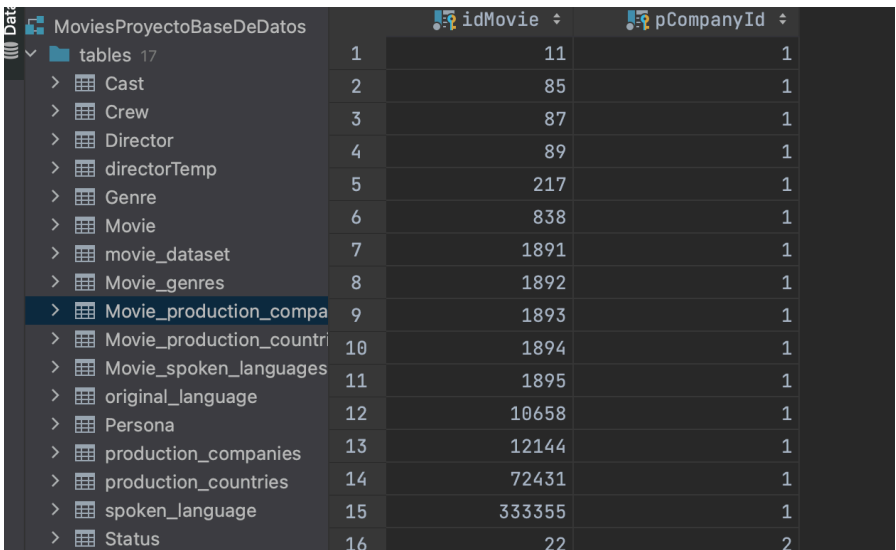
|    | idGenre | nameGenre       |
|----|---------|-----------------|
| 1  | 1       | Action          |
| 2  | 2       | Adventure       |
| 3  | 3       | Fantasy         |
| 4  | 4       | Science-Fiction |
| 5  | 5       | Crime           |
| 6  | 6       | Drama           |
| 7  | 7       | Thriller        |
| 8  | 8       | Animation       |
| 9  | 9       | Family          |
| 10 | 10      | Western         |
| 11 | 11      | Comedy          |
| 12 | 12      | Romance         |
| 13 | 13      | Horror          |
| 14 | 14      | Mystery         |
| 15 | 15      | History         |
| 16 | 16      | War             |

Figura 20: Poblacion de la table Genre



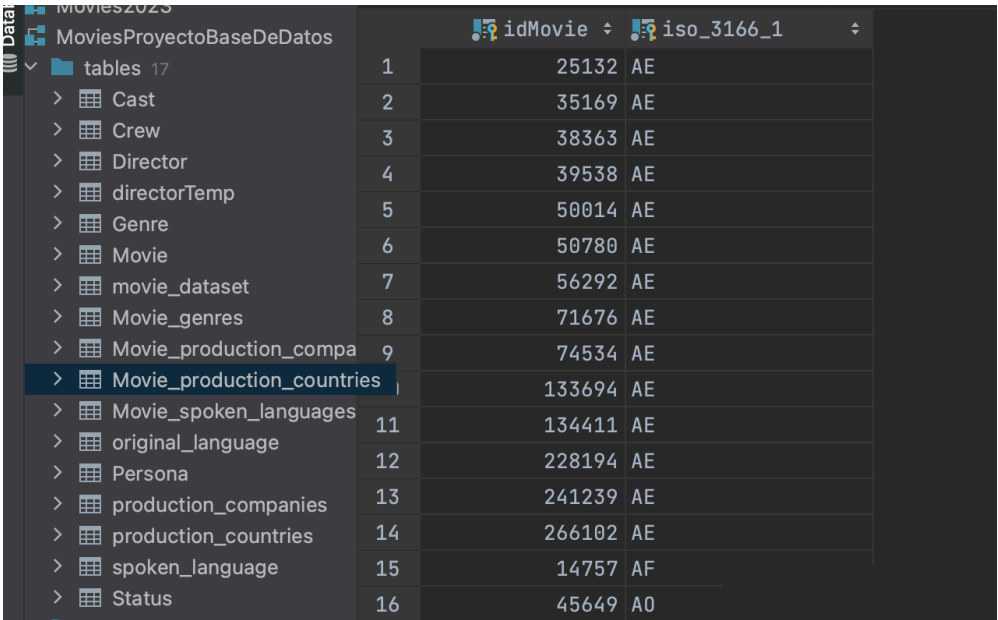
|    | idMovie | index | budget    | homepage                                | keywords                  |
|----|---------|-------|-----------|---|---------------------------|
| 1  | 5       | 3766  | 4000000   | <null>                                  | hotel new year's eve wito |
| 2  | 11      | 2912  | 11000000  | http://www.starwars.com/films/star-w... | android galaxy hermit dea |
| 3  | 12      | 328   | 94000000  | http://movies.disney.com/finding-nemo   | father son relationship h |
| 4  | 13      | 809   | 55000000  | <null>                                  | vietnam veteran hippie me |
| 5  | 14      | 2516  | 15000000  | http://www.dreamworks.com/ab/           | male nudity female nudity |
| 6  | 16      | 2799  | 12800000  | <null>                                  | individual dancing usa ro |
| 7  | 18      | 322   | 90000000  | <null>                                  | clone taxi cyborg egypt f |
| 8  | 19      | 2638  | 92620000  | <null>                                  | man vs machine undergroun |
| 9  | 20      | 4023  | 0         | http://www.clubcultura.com/clubcine/... | farewell responsibility d |
| 10 | 22      | 199   | 140000000 | http://disney.go.com/disneyvideos/li... | exotic island blacksmith  |
| 11 | 24      | 828   | 30000000  | http://www.miramax.com/movie/kill-bi... | japan coma martial arts k |
| 12 | 25      | 557   | 72000000  | <null>                                  | sniper marine corps saudi |
| 13 | 28      | 1525  | 31500000  | http://www.apocalypsenow.com            | guerrilla river vietnam v |
| 14 | 33      | 2664  | 14000000  | <null>                                  | prostitute sheriff bounty |
| 15 | 35      | 558   | 75000000  | http://www.simpsonsmovie.com/           | father son relationship l |
| 16 | 38      | 2152  | 20000000  | http://www.eternalsunshine.com          | deja vu regret jealousy a |

Figura 21: Poblacion de la table Movie



|    | idMovie | pCompanyId |
|----|---------|------------|
| 1  | 11      | 1          |
| 2  | 85      | 1          |
| 3  | 87      | 1          |
| 4  | 89      | 1          |
| 5  | 217     | 1          |
| 6  | 838     | 1          |
| 7  | 1891    | 1          |
| 8  | 1892    | 1          |
| 9  | 1893    | 1          |
| 10 | 1894    | 1          |
| 11 | 1895    | 1          |
| 12 | 10658   | 1          |
| 13 | 12144   | 1          |
| 14 | 72431   | 1          |
| 15 | 333355  | 1          |
| 16 | 22      | 2          |

Figura 22: Poblacion de la table *Movie\_production\_companies*



|    | idMovie | iso_3166_1 |
|----|---------|------------|
| 1  | 25132   | AE         |
| 2  | 35169   | AE         |
| 3  | 38363   | AE         |
| 4  | 39538   | AE         |
| 5  | 50014   | AE         |
| 6  | 50780   | AE         |
| 7  | 56292   | AE         |
| 8  | 71676   | AE         |
| 9  | 74534   | AE         |
| 10 | 133694  | AE         |
| 11 | 134411  | AE         |
| 12 | 228194  | AE         |
| 13 | 241239  | AE         |
| 14 | 266102  | AE         |
| 15 | 14757   | AF         |
| 16 | 45649   | A0         |

Figura 23: Poblacion de la table *Movie\_production\_countries*

|    | idMovie | iso_639_1 |  |
|----|---------|-----------|--|
| 1  | 868     | af        |  |
| 2  | 1123    | af        |  |
| 3  | 1372    | af        |  |
| 4  | 17654   | af        |  |
| 5  | 22600   | af        |  |
| 6  | 59961   | af        |  |
| 7  | 192136  | af        |  |
| 8  | 9839    | am        |  |
| 9  | 25      | ar        |  |
| 10 | 85      | ar        |  |
| 11 | 231     | ar        |  |
| 12 | 409     | ar        |  |
| 13 | 435     | ar        |  |
| 14 | 612     | ar        |  |
| 15 | 691     | ar        |  |
| 16 | 708     | ar        |  |

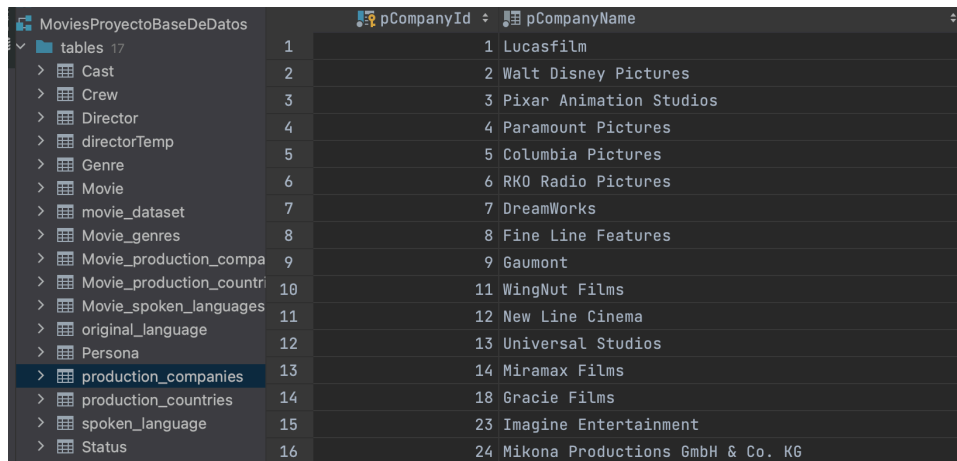
Figura 24: Poblacion de la table *Movie\_spoken\_languages*

|    | idOriginalLang | name_original_language |
|----|----------------|------------------------|
| 1  | 1              | en                     |
| 2  | 2              | ja                     |
| 3  | 3              | fr                     |
| 4  | 4              | zh                     |
| 5  | 5              | es                     |
| 6  | 6              | de                     |
| 7  | 7              | hi                     |
| 8  | 8              | ru                     |
| 9  | 9              | ko                     |
| 10 | 10             | te                     |
| 11 | 11             | cn                     |
| 12 | 12             | it                     |
| 13 | 13             | nl                     |
| 14 | 14             | ta                     |
| 15 | 15             | sv                     |
| 16 | 16             | th                     |

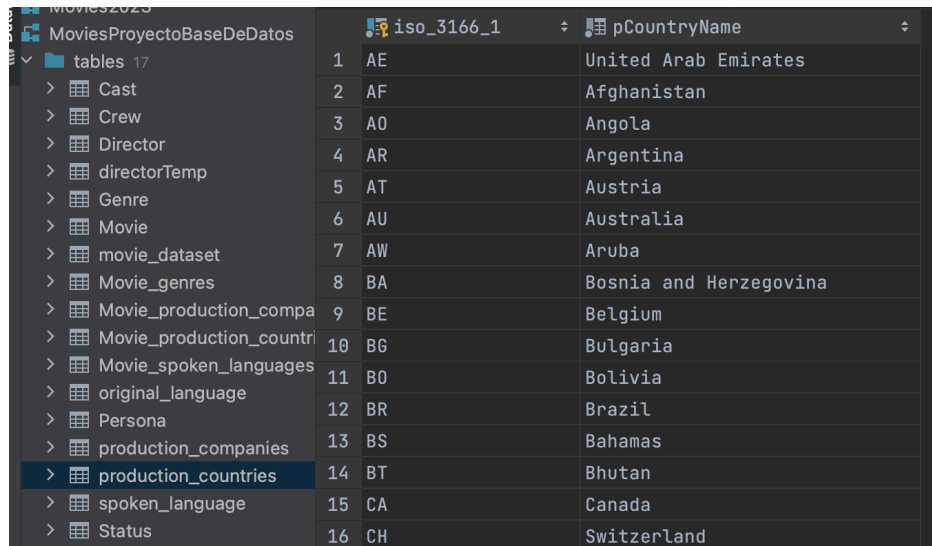
Figura 25: Poblacion de la table *original\_language*

|    | idPerson | name            | gender |
|----|----------|-----------------|--------|
| 1  | 1        | George Lucas    | 2      |
| 2  | 3        | Harrison Ford   | 2      |
| 3  | 7        | Andrew Stanton  | 2      |
| 4  | 8        | Lee Unkrich     | 2      |
| 5  | 9        | Graham Walters  | 2      |
| 6  | 10       | Bob Peterson    | 2      |
| 7  | 11       | David Reynolds  | 2      |
| 8  | 13       | Albert Brooks   | 2      |
| 9  | 24       | Robert Zemeckis | 2      |
| 10 | 26       | Winston Groom   | 2      |
| 11 | 27       | Eric Roth       | 2      |
| 12 | 28       | Wendy Finerman  | 1      |
| 13 | 29       | Steve Tisch     | 2      |
| 14 | 30       | Steve Starkey   | 2      |
| 15 | 31       | Tom Hanks       | 2      |
| 16 | 33       | Gary Sinise     | 2      |

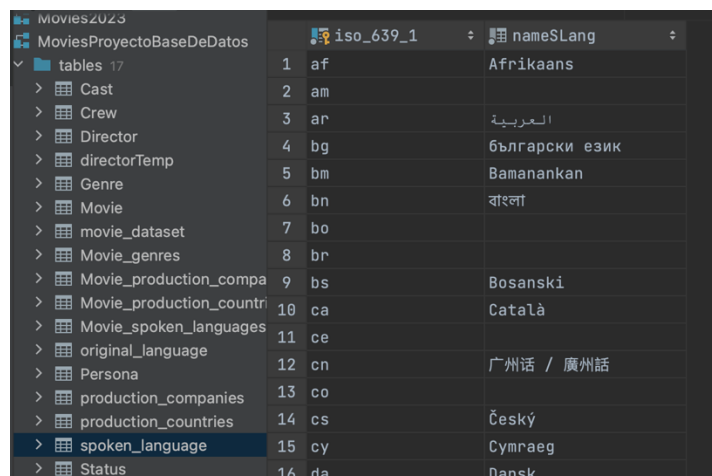
Figura 26: Poblacion de la table *Persona*



| pCompanyId | pCompanyName                     |
|------------|----------------------------------|
| 1          | Lucasfilm                        |
| 2          | Walt Disney Pictures             |
| 3          | Pixar Animation Studios          |
| 4          | Paramount Pictures               |
| 5          | Columbia Pictures                |
| 6          | RKO Radio Pictures               |
| 7          | DreamWorks                       |
| 8          | Fine Line Features               |
| 9          | Gaumont                          |
| 10         | WingNut Films                    |
| 11         | New Line Cinema                  |
| 12         | Universal Studios                |
| 13         | Miramax Films                    |
| 14         | Gracie Films                     |
| 15         | Imagine Entertainment            |
| 16         | Mikona Productions GmbH & Co. KG |

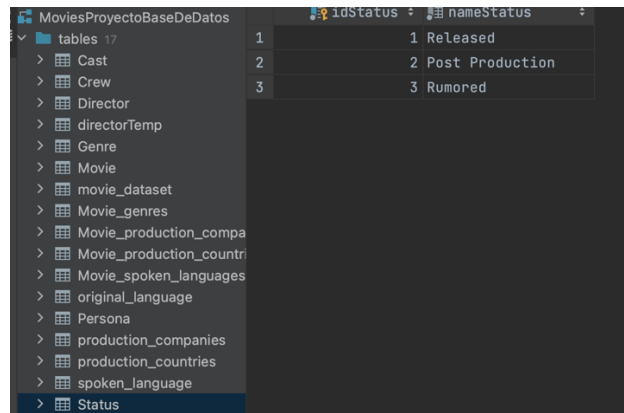
Figura 27: Poblacion de la table *production\_companies*


| iso_3166_1 | pCountryName              |
|------------|---------------------------|
| 1          | AE United Arab Emirates   |
| 2          | AF Afghanistan            |
| 3          | AO Angola                 |
| 4          | AR Argentina              |
| 5          | AT Austria                |
| 6          | AU Australia              |
| 7          | AW Aruba                  |
| 8          | BA Bosnia and Herzegovina |
| 9          | BE Belgium                |
| 10         | BG Bulgaria               |
| 11         | BO Bolivia                |
| 12         | BR Brazil                 |
| 13         | BS Bahamas                |
| 14         | BT Bhutan                 |
| 15         | CA Canada                 |
| 16         | CH Switzerland            |

Figura 28: Poblacion de la table *production\_countries*


| iso_639_1 | nameSLang         |
|-----------|-------------------|
| 1         | af Afrikaans      |
| 2         | am                |
| 3         | ar العربية        |
| 4         | bg български език |
| 5         | bm Bamanankan     |
| 6         | bn বাংলা          |
| 7         | bo                |
| 8         | br                |
| 9         | bs Bosanski       |
| 10        | ca Català         |
| 11        | ce                |
| 12        | cn 广州话 / 廣州話      |
| 13        | co                |
| 14        | cs Český          |
| 15        | cy Cymraeg        |
| 16        | da Dansk          |

Figura 29: Poblacion de la table *spoken\_languages*



| idStatus | nameStatus      |
|----------|-----------------|
| 1        | Released        |
| 2        | Post Production |
| 3        | Rumored         |

*Figura 23: Poblacion de la table status*

Es importante recalcar el orden de ejecucion de las sentencias CREATE TABLE.

Debido a que Movie hace referencia a dos llaves foraneas, es necesario primero crear esas dos tablas que contienen esas llaves foraneas como primarias para poder crear esta tabla.

Igualmente con las tablas con relacion de muchos a muchos es necesario primero crear la tabla Movie y las tablas de las cuales se utilizara su llave primaria para esta relacion de muchos a muchos. Por ejemplo, ya deben estar creadas las tablas status y original\_language para poder crear la tabla Movie. De ahí, debe estar creada la tabla spoken\_languages para poder crear la tabla muchos a muchos llamada Movie\_spoken\_languages.

## Cursors & Procedures

Ahora es momento de comentar acerca de los cursores y procedimientos. Se utilizaron procedimientos para poblar la base de datos, y dentro de los procedimientos, existen cursores. Cursores son objetos, los cuales nos apuntan por fila a los resultados de una sentencia de consulta. En otras palabras, permiten procesar resultados de una consulta de manera secuencial y controlada. En cambio procedimientos son un conjunto de instrucciones SQL almacenadas en la base de datos y pueden ser invocados mediante una llamada. Podemos utilizar procedimientos para realizar tareas repetitivas, pero en este caso, han sido utilizados para poblar las tablas, creadas en base al modelo fisico.



```

244 DROP PROCEDURE IF EXISTS TablaMovie;
245
246 DELIMITER $$
247 CREATE PROCEDURE TablaMovie()
248 BEGIN
249
250 DECLARE done INT DEFAULT FALSE;
251
252 DECLARE Mov_idMovie INT;
253 DECLARE Mov_index INT;
254 DECLARE Mov_budget BIGINT;
255 DECLARE Mov_homepage VARCHAR(255);
256 DECLARE Mov_keywords VARCHAR(255);
257 DECLARE Mov_name_original_language VARCHAR(2);
258 DECLARE Mov_original_title VARCHAR(255);
259 DECLARE Mov_overview TEXT;
260 DECLARE Mov_popularity DOUBLE;
261 DECLARE Mov_release_date DATE;
262 DECLARE Mov_revenue BIGINT;
263 DECLARE Mov_runtime DOUBLE;
264 DECLARE Mov_nameStatus VARCHAR(15);
265 DECLARE Mov_tagline VARCHAR(255);
266 DECLARE Mov_title VARCHAR(255);
267 DECLARE Mov_vote_average DOUBLE;
268 DECLARE Mov_vote_count INT;

```

Figura 24: Procedimiento parte 1

```

269
270 DECLARE Status_idStatus int;
271 DECLARE OL_idOriginal_language int;
272
273 -- Declarar el cursor
274 DECLARE CursorMovie CURSOR FOR
275 SELECT id,'index',budget,homepage, keywords, original_language, original_title,
276 popularity, release_date, revenue, runtime, 'status', tagline, title,
277 vote_average, vote_count FROM movie_dataset;
278
279 -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha
280 DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
281
282 -- Abrir el cursor
283 OPEN CursorMovie;
284
285 CursorMovie_loop: LOOP
286 FETCH CursorMovie INTO Mov_idMovie,Mov_index,Mov_budget, Mov_homepage,
287 Mov_original_title, Mov_overview, Mov_popularity, Mov_release_date,
288 Mov_nameStatus, Mov_tagline, Mov_title, Mov_vote_average, Mov_vote_
289
290 -- Si alcanza el final del cursor entonces salir del ciclo repetitivo
291 IF done THEN
292 LEAVE CursorMovie_loop;
293 END IF;

```

Figura 25: Procedimiento parte 2

```

294 SELECT 'idStatus' INTO Status_idStatus FROM Status WHERE nameStatus = Mov_nameStatus;
295 SELECT 'idOrigLang' INTO OL_idOriginal_language FROM original_language WHERE name_origi
296
297 INSERT INTO Movie ('idMovie','index',budget,homepage,keywords,idOrigLang,original_title,
298 tagline,title,vote_average,vote_count)
299 VALUES (Mov_idMovie,Mov_index,Mov_budget, Mov_homepage, Mov_keywords,OL_idOriginal_langue
300 Mov_original_title, Mov_overview, Mov_popularity, Mov_release_date, Mov_revenue, Mov_
301 Status_idStatus, Mov_tagline, Mov_title, Mov_vote_average, Mov_vote_count);
302
303 END LOOP;
304 CLOSE CursorMovie;
305 END $$
306 DELIMITER ;
307
308 CALL TablaMovie ();
309
310 SELECT * FROM Movie;
311

```

Figura 26: Procedimiento parte 3

## Explicacion del procedimiento TablaMovie()

1. Si es que existe ya el procedimiento, realizamos un DROP PROCEDURE.
2. Colocamos el DELIMITER \$\$ seguido por el CREATE PROCEDURE.
3. Colocamos BEGIN y empezamos a escribir nuestras variables donde declaramos una llamada donde, de tipo entero, con un valor por default falso. Este lo utilizaremos para saber cuando hemos acabado de recorrer la sentencia de busqueda.
4. Declaramos todas las variables con su respectivo tipo de dato, los cuales hicimos que coincidieran con el tipo de dato que utilizan esos valores en sus columnas. Ademas de las variables para guardar esos valores, generamos dos mas variables de tipo entero para almacenar el id de status y el id de original\_language ya que en nuestra tabla, no utilizaremos los valores String de status y original\_language para ahorrar espacio en memoria debido a la gran cantidad de filas.
5. Declaramos nuestro cursor y ponemos nuestra sentencia de busqueda.
6. Declaramos nuestro handler, el cual, cuando regrese NOT FOUND, cambiara el estado de la variable donde mencionada anteriormente a true.
7. Abrimos el cursor donde realizaremos un loop y hacemos fetch para obtener los reultados de la busqueda y los colocamos en nuestras variables establecidas al inicio.
8. Una vez que done sea verdadero, salimos del loop del cursor.
9. Seleccionamos el id de status donde el nombre de status sea igual al status que tenemos guardado en la variable gracias al fetch. Guardamos ese id en la variable de id de status.
10. Seleccionamos el id de original\_language donde el nombre de original\_language sea igual al original\_language que tenemos guardado en la variable gracias al fetch. Guardamos ese original\_language en la variable de id de original\_language.

11. Insertamos los valores especificando a cual tabla seran ingresados y en vez de enviar status y original\_language, enviamos sus id's en esos parametros.
12. Colocamos los valores (VALUES), los cuales seran insertados tomando en cuenta los id's de status y original\_language.
13. Terminamos el loop.
14. Cerramos el cursor.
15. Finalizamos el \$\$ DELIMITER.
16. Para realizar la instruccion y pasos del procedimiento, lo llamamos, CALL TablaMovie(); y empieza la ejecucion.

## Conclusiones

Este proyecto fue realizado bajo unas normas de modelado de base de datos, como normalización, diseño conceptual, lógico. Cabe resaltar que se ha tenido que afrontar varios desafíos, como transformación y limpieza de datos, a medida que íbamos progresando en el proyecto, sin embargo, bajo la guía de nuestros docentes y al aplicar nuestros conocimientos para la resolución de problemas, pudimos encontrar diferentes técnicas que nos llevaban a la misma solución. Gracias a esta práctica, he podido descubrir nuevos métodos de análisis y maneras de abarcar un problema, y cuando aparecen diferentes ejercicios, puedo aplicar la mejor manera de resolución del problema.

Una vez realizado todos los pasos correspondientes para culminar nuestro proyecto, hemos podido evidenciar un gran progreso en nuestra toma de decisiones referentes al modelado y población de una base de datos, además de aprender nuevas estrategias de programación aplicables a casos de la vida real como ocupaciones, personajes y casos de cultura popular. Este proyecto me ha permitido crecer en muchos aspectos y, además, me ha permitido fusionar conocimientos de diferentes materias, permitiéndome tener una práctica de gran calidad con un tema que nos abre la puerta a los posibles problemas presentes en la vida real.

Al momento de tratar con datos, se debe evitar consistencia y errores. Contamos con muchas herramientas que han sido de gran ayuda e importancia para interactuar con la base de datos. Hemos podido aplicar todo lo que ha sido indicado en este ciclo y se resaltó la importancia de los diferentes temas estudiados.