



**UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA**

*La Universidad Católica de Loja*

**Entrega Proyecto Final**

**Fundamentos de Base de Datos**

**Autor:**

- Cristian Rodríguez

Octubre 2022 – Febrero 2023

## Contenido

1	Introducción .....	3
2	Desarrollo del Componente.....	3
2.1	Diseño y Modelado de la Base de datos.....	3
2.1.1	Tabla de dependencias funcionales.....	5
2.1.2	Diseño Conceptual.....	7
2.1.3	Diseño Lógico.....	8
2.1.4	Diseño Físico .....	10
2.2	Esquema de la Base de Datos .....	11
2.3	Limpieza de datos.....	12
2.4	Importación del .csv al SQL .....	14
2.5	Declaración de Tablas .....	15
2.6	Cursores .....	18
3	Conclusiones .....	32

# 1 Introducción

En el presente proyecto de la materia Fundamentos de Base de datos, se pretende aplicar los conocimientos obtenidos durante todo el ciclo para de esa forma trabajar con el archivo CSV llamado “movie\_dataset”, el cual fue obtenido de un repositorio de GitHub. El Dataset tendrá que ser analizado, leído, modelado, limpiado y por último explotado utilizando el lenguaje de consulta estructurada SQL específicamente en el sistema de gestión de bases de datos relacional MySQL.

## 2 Desarrollo del Componente

### 2.1 Diseño y Modelado de la Base de datos

El diseño y modelado de una base de datos son procesos críticos para el correcto almacenamiento y manipulación de datos en nuestro sistema. Para el dataset "movie\_dataset", es necesario identificar los tipos de datos relevantes y las relaciones entre ellos. A continuación, se describen los pasos para diseñar y modelar la base de datos.

1. Identificación de entidades: Las entidades son objetos o conceptos que se desean almacenar en la base de datos. En el caso del dataset "movie\_dataset", algunas de las entidades pueden ser "movies", "genre", "production\_companies", "production\_countries" y "spoken\_language" que vienen a ser atributos multivaluados o bien compuestos, por ende, tienden a ser entidades.
2. Identificación de atributos: Cada entidad tendrá una serie de atributos que describan sus características. Por ejemplo, la entidad "movies" podría tener atributos como "original\_title", "release\_date", "budget", entre otros. Aquellos que aportan con información adicional a la entidad.
3. Identificación de relaciones: Las relaciones son los vínculos entre las entidades, que nos permiten saber cómo se va a comunicar la información de una tabla con otra.

4. Creación del modelo entidad-relación (ER): Una vez identificadas las entidades, atributos y relaciones, se puede crear un modelo ER que represente la estructura de la base de datos.
5. Conversión del modelo ER a un modelo relacional: El modelo ER se convierte en un modelo relacional, en el que las entidades se convierten en tablas y las relaciones se representan mediante claves foráneas.
6. Creación de la base de datos: Finalmente, se crea la base de datos en el sistema de gestión de bases de datos elegido (por ejemplo, MySQL) y se cargan los datos desde el dataset "movie\_dataset".

Con esta información en mente, podemos identificamos la cardinalidad de cada atributo con nuestra tabla principal que va a ser “movies” y nos encontramos con los siguientes resultados:

Por otro lado, tenemos:

genres	→	N:M
production_companies	→	N:M
production countries	→	N:M
spoken_language	→	N:M
cast	→	N:M
crew	→	N:M

Atributos de los cuales pudimos apreciar que “production\_companies”, “production countries”, “spoken\_language”, “cast” y “crew” al estar en formato de texto de tipo JSON van a tener más atributos dentro de cada uno, por lo que los convierte en un atributo compuesto.

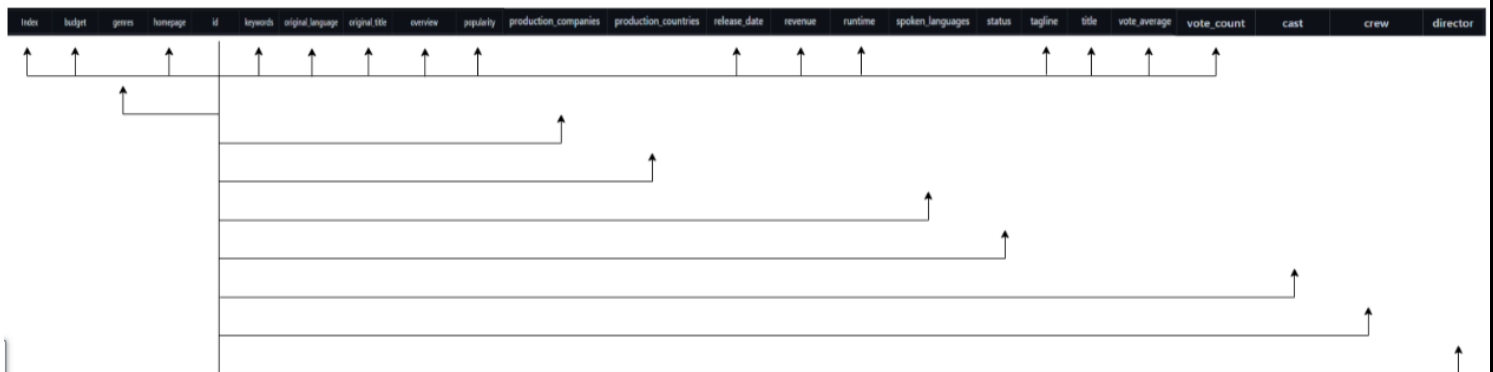
Así mismo, apreciamos que “genres” va a ser un atributo multivaluado y así como los atributos compuestos, se van a separar en nuevas tablas, con la diferencia que en el caso de la multivaluada va a ser una entidad débil por el hecho de que depende directamente de llave principal “id” para que se pueda relacionar.

Como lo que buscamos es tener la mayor cantidad de información posible al momento de que se desee hacer una explotación de datos, decidimos tomar los atributos “status” y “original\_language” ya que analizando el contenido del archivo vimos conveniente tomar los datos de estos dos atributos por la cantidad significativa de información que brindan.

### 2.1.1 Tabla de dependencias funcionales

Antes de entrar al modelado de la base de datos, vamos a analizar nuestros datos dados individualmente para ver que atributo depende de cual y los atributos que se convierten en entidades al momento de que tienen mas atributos dentro de ellos. Esto nos va a ayudar a de definir las llaves tanto primarias como foráneas.

Basamos nuestro primer modelo de una manera simple colocando todos los atributos e identificando la dependencia que cada uno tenía con id.



*Figura 1. Tabla de dependencias primer modelo*

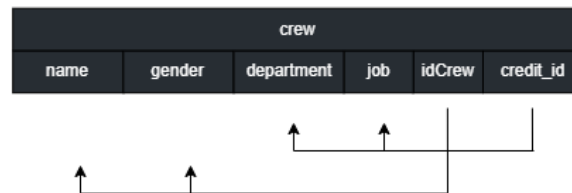
Luego de trabajar en un pensamiento mas a fondo de las relaciones que llevan las tablas u entidades, se pudo mejorar el diseño de la tabla para que se tome en cuenta cada atributo que contenía

individualmente las entidades para tomar en cuenta verdaderamente todos los atributos con los que se van a trabajar.



Figura 2. Tabla de dependencias segundo modelo

Cayendo en cuenta que dentro de “crew” se puede encontrar alrededor de seis atributos, se puede inferir que dentro de ella debe haber un grupo de dependencias funcionales aparte. Por lo que nuestro diseño de dependencias funcionales va a estar basado en las dependencias de ambas tablas:



$idCrew \rightarrow \{name, gender\}$

$credit\_id \rightarrow \{department, job\}$

$id \rightarrow \{index, budget, homepage, keywords, original\_title, overview, popularity, release\_date, revenue, runtime, tagline, title, vote\_avarage, vote\_count\}$

$id \rightarrow \{idGenre, name\_Genre\}$

$id \rightarrow \{name\_original\_language, idOrigLang\}$

$id \rightarrow \{pcompanyId, pcompanyName\}$

id → {iso\_3166\_1, pcountryName}

id → {nameSLang, iso\_639\_1}

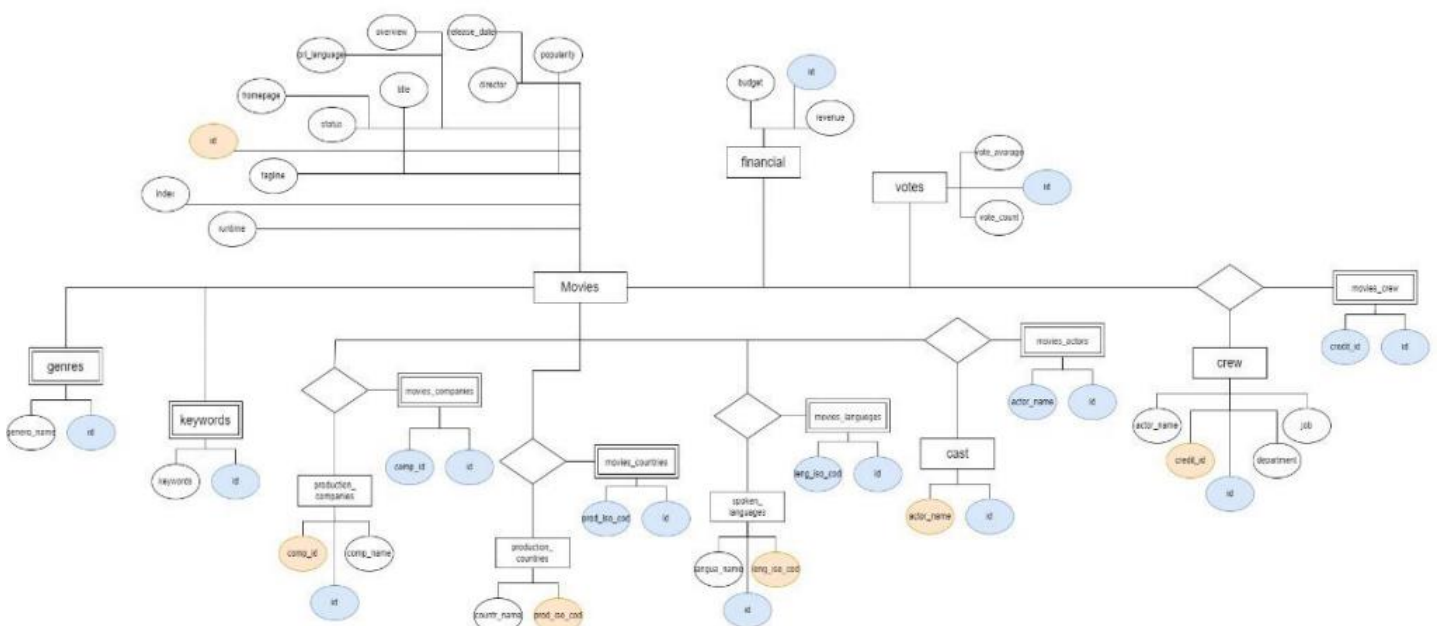
id → {idStatus, nameStatus}

id → {name, gender, department, job, idCrew, credit\_id, director}

## 2.1.2 Diseño Conceptual

Para tener una comprensión clara de la estructura de nuestro proyecto "movie\_dataset", es necesario realizar el modelamiento conceptual que describa cómo se relacionan los datos dentro del mismo. Este proceso incluye la utilización de un modelo Entidad-Relación para determinar la cantidad de entidades y atributos presentes en nuestro dataset.

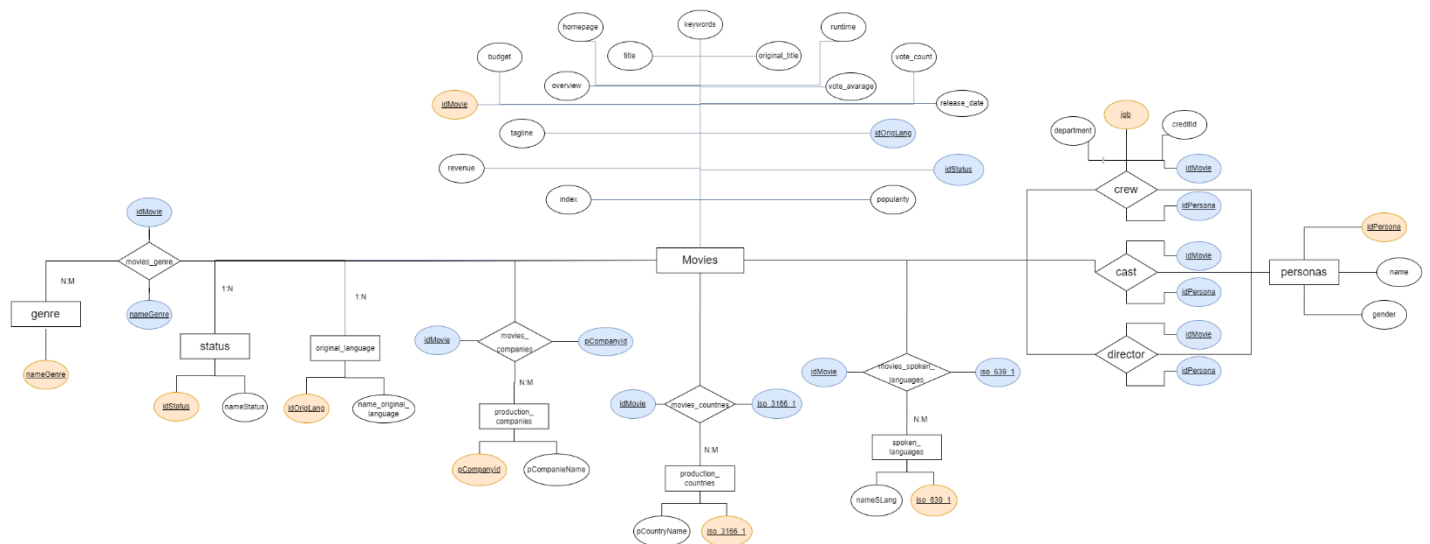
Para ilustrar de manera gráfica nuestro modelo, utilizamos la herramienta de [app.diagrams.net](https://app.diagrams.net). Esta herramienta nos permitió visualizar las relaciones entre las entidades y atributos del dataset a trabajar. Al tener una representación gráfica, es más fácil comprender la estructura y relaciones de los datos.



Así como se mencionó en la tabla de dependencias, se fue madurando el diseño de los modelos, empezando con un modelo conceptual tal como se muestra a continuación:

Con la nueva tabla de dependencia procedimos a diseñar de nuevo a nuestro modelo conceptual con el cual íbamos a trabajar de largo como la estructura de nuestra base de datos.

Figura 4. Modelo Conceptual diseño final



Dentro de este diseño vamos a tomar en cuenta que podemos unir a todas las personas que participan en una película, por lo que vamos a tomar las columnas de crew, cast y director para formar una tabla que las contenga a estas tres para tener mejor almacenada la información.

### 2.1.3 Diseño Lógico

Para diseñar un modelo lógico y representar las dependencias funcionales del dataset, debemos primero identificar las entidades y relaciones que existen en los datos, por lo que primeramente vamos a definir tanto llaves primarias como llaves foráneas. Una vez identificadas, podemos crear un modelo entidad-relación para representar esas relaciones de manera gráfica.



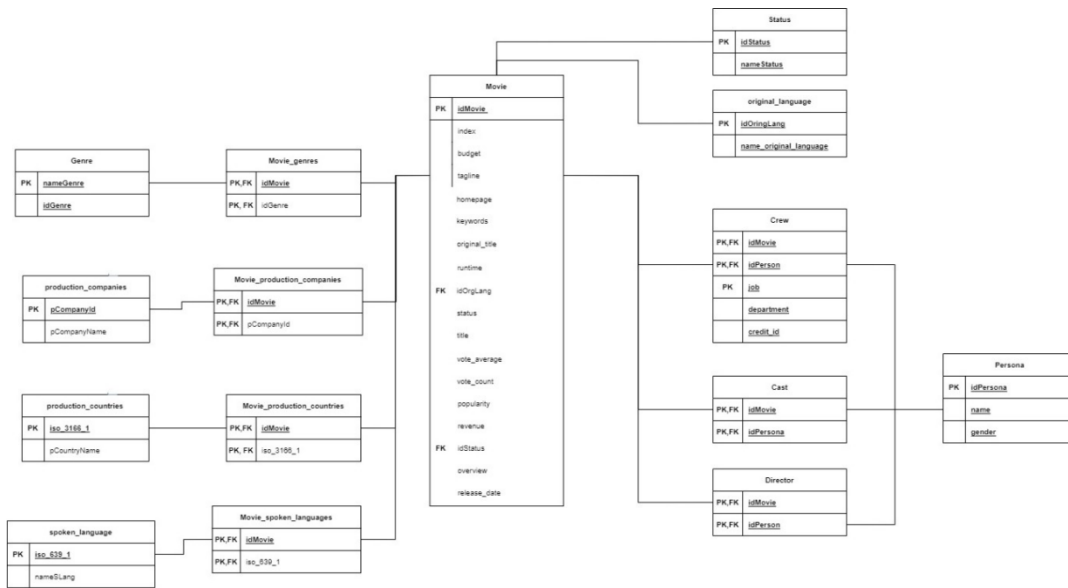


Figura 5. Modelo Lógico

## 2.1.4 Diseño Físico

Una vez con este modelo podemos pasar a la limpieza e inserción de los datos dentro de nuestra base de datos SQL, lo cual, una vez creadas las tablas nos dará como resultado nuestro modelo final que será nuestro modelo físico que es aquel con el cual vamos a trabajar en base a código.

Adelantándome al resultado mencionado, el modelo físico una vez realizado los pasos previamente mencionados.

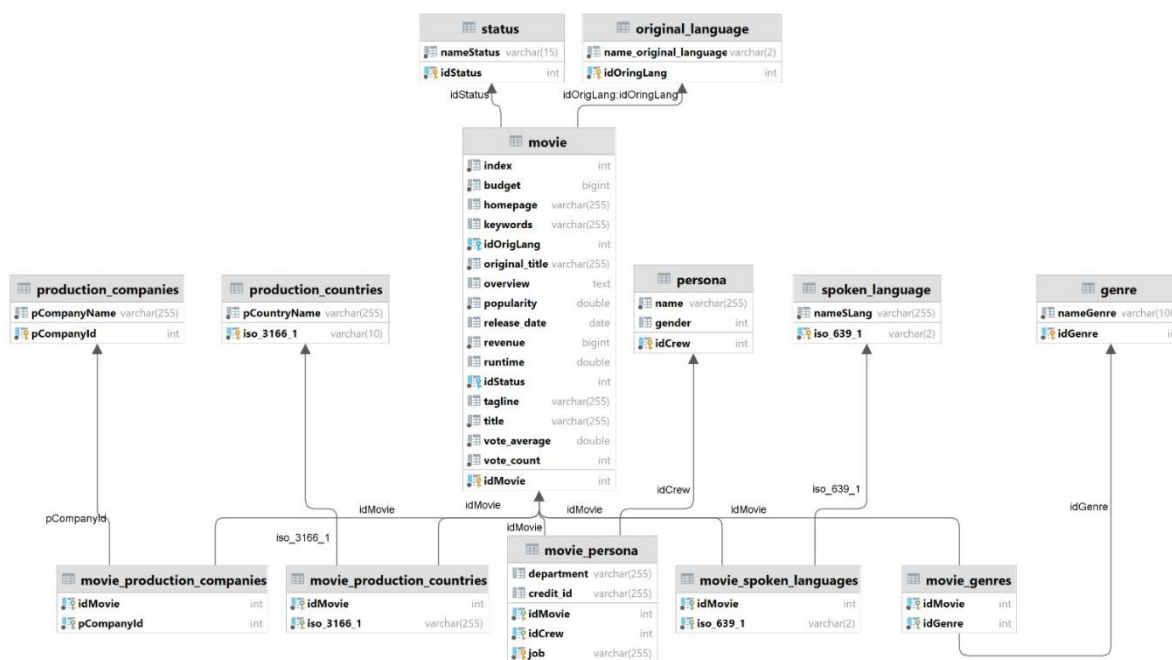


Figura 6. Modelo Físico

El diseño físico de una base de datos se refiere a la implementación concreta de la estructura de datos que se ha definido en el modelo lógico. En el caso del dataset "movie\_dataset", este se podría implementar en un DBMS (Sistema de Gestión de Bases de Datos) como DataGrip, utilizamos el diseño lógico como entrada para crear un diseño físico que cumpla con los requisitos de rendimiento y escalabilidad.

Para generar el diseño físico del dataset de nuestro proyecto, primero se necesita crear una conexión con la base de datos y luego crear una nueva tabla. A continuación, se pueden definir las columnas y restricciones de la tabla, y especificar el tipo de datos para cada columna. Finalmente, se puede guardar el diseño físico y ejecutar scripts SQL para crear la tabla en la base de datos.

## **2.2 Esquema de la Base de Datos**

El esquema de la base de datos "movie\_dataset" es una colección de tablas relacionadas entre sí para almacenar información sobre películas, actores, productoras, países de producción, lenguajes hablados, géneros

La base de datos "movie" consta de las siguientes tablas:

- **movies:** Almacena información básica de las películas, incluyendo el ID, el nombre del director, título original, título, idioma original, palabras clave, página web, descripción general, popularidad, estado de producción, fecha de lanzamiento, ingresos, duración, frase promocional, promedio de votos y número de votos.
- **status:** Almacena el estado de producción de las películas.
- **director:** Almacena información sobre los directores de las películas.
- **production\_companies:** Almacena información sobre las compañías de producción de las películas.
- **production\_countries:** Almacena información sobre los países de producción de las películas.
- **genres:** Almacena información sobre los géneros de las películas.

- spoken\_language: Almacena información sobre los idiomas hablados en las películas.
- cast: Almacena información sobre los actores de las películas.
- crew: Almacena información sobre el equipo de producción de las películas, incluyendo el nombre, género, departamento, trabajo e ID de crédito.
- movies\_companies: Almacena relaciones entre películas y compañías de producción.
- movies\_countries: Almacena relaciones entre películas y países de producción.
- movies\_languages: Almacena relaciones entre películas e idiomas hablados.
- movies\_cast: Almacena relaciones entre películas y actores.
- movies\_crew: Almacena relaciones entre películas y equipo de producción.
- movies\_genres: Almacena relaciones entre películas y géneros.

## **2.3 Limpieza de datos**

Para la limpieza de datos vamos a centrarnos en los problemas puntuales que tenga cada columna del dataset. Para eso vamos a crear un constructor por cada tabla que tengamos basándonos en nuestro modelado y dentro del objeto vamos a ir recogiendo la información tomando en cuenta las excepciones respectivas respetando el tipo de dato. Al ya saber cuáles datos son de tipo numérico y cuáles de tipo cadena de texto podemos aplicar ciertas funciones que nos permitan obtener el dato lo más aproximado a que se busca.

Primero hablando de los datos de tipo texto; notamos que en ciertos casos nuestra cadena de texto va a estar acompañada de caracteres especiales que confundirían a nuestro programa al momento de leerlos para lo cual vamos a definir ciertas funciones que nos permitan corregir estos caracteres a caracteres que nuestro programa pueda procesar, tal como en el caso “original\_title”

donde se tiene que corregir la existencia de apostrofes. Asi mismo, con ayuda del docente, encontramos un patrón que nos permite identificar la existencia de un apostrofe dentro de un par de comillas, viceversa en el caso de que exista una comilla doble dentro de dos apostrofes.

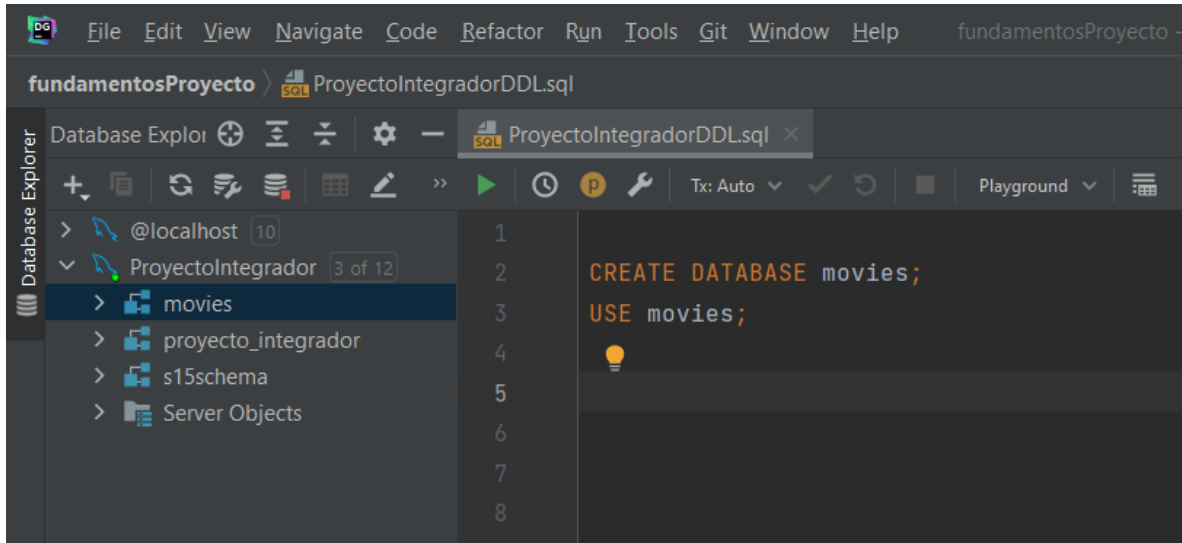
```
SELECT id,
JSON_VALID(CONVERT (
REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(crew,
''', '\'),
'{\ ', '{ '),
'\ ': \', ': '),
'\ ', \', ', '),
'\ ': ', ': '),
', \ ', ', ')
USING UTF8mb4 )) AS Valid_YN,
CONVERT (
REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(crew,
''', '\'),
'{\ ', '{ '),
'\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

Dentro de la columna genres se debe tomar encuenta que no existe un separador definido y no se sabe a ciencia cierta si son géneros de una o mas palabras, para ello hacemos una comparación de la cantidad de veces que se repite cierta palabra, ya que si se repiten el mismo numero de veces, se puede inferir que aquellas palabras forman una sola. Para esto vamos a usar “tokens” que vienen a ser un reemplazo de la palabra que se encuentra en el documento con una nueva palabra que leía a través del programa se lo interpreta como una sola palabra

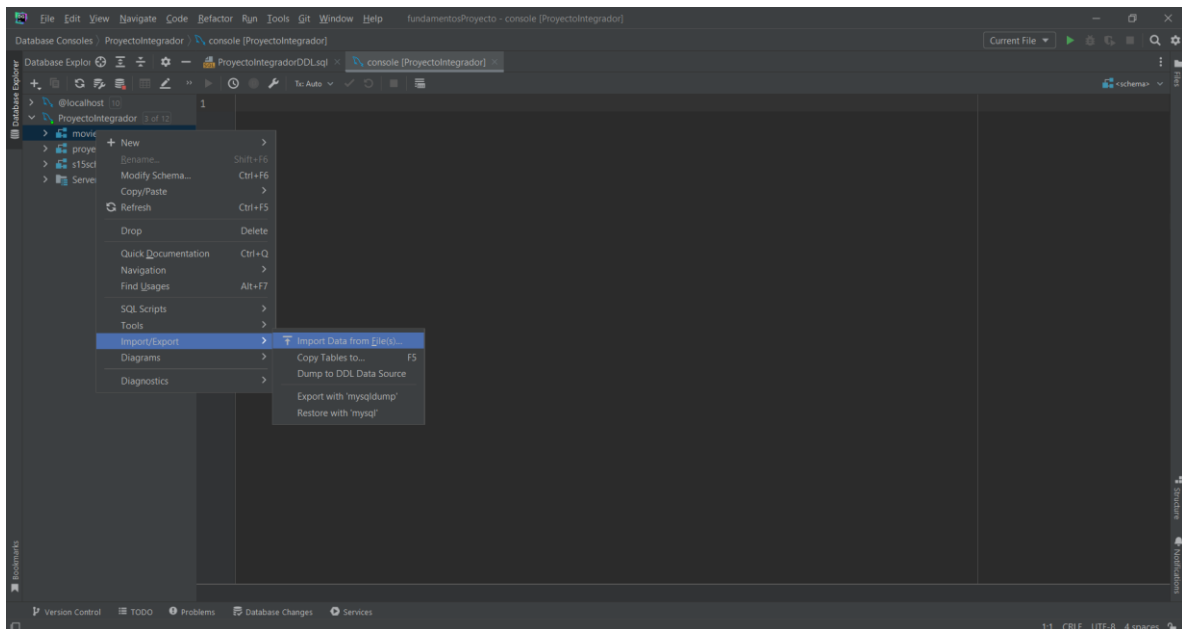
En el caso de los tipos de dato numéricos, se debe tener en cuenta que vamos a tener filas con datos en blanco, por lo que se debe ir corrigiendo eso bien al momento de que se las va a insertar en la base de data.

## 2.4 Importación del .csv al SQL

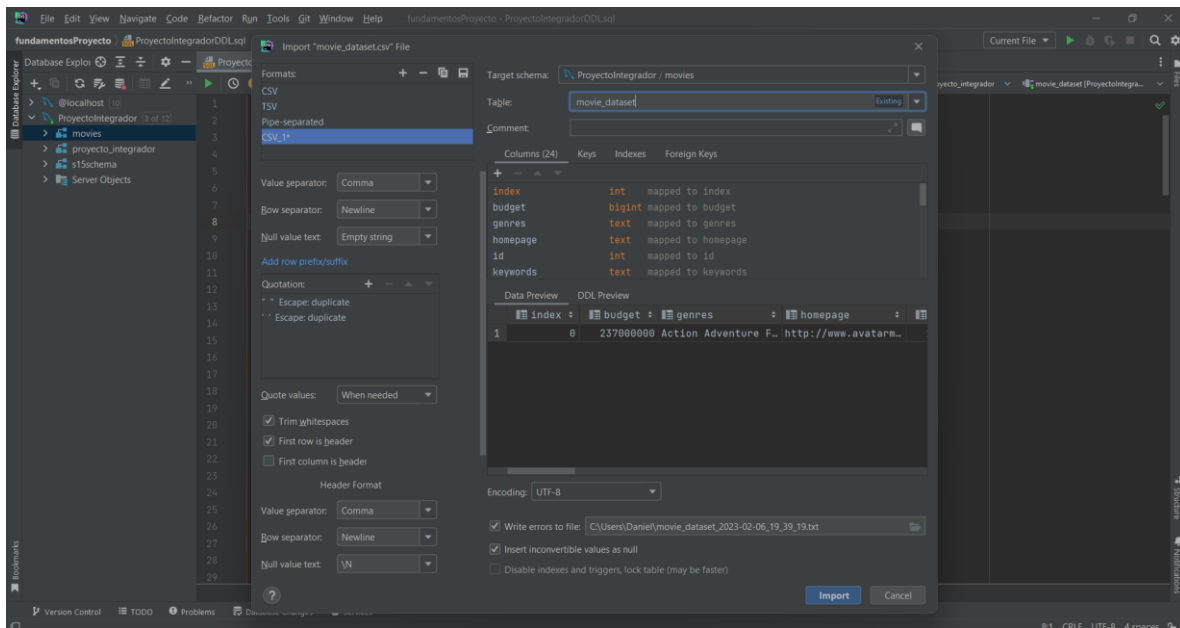
Como vamos a trabajar puramente desde SQL tenemos que tener una manera de extraer los datos del archivo SQL, por lo que vamos a importar todo el archivo dentro de nuestro esquema o base de datos donde vayamos a trabajar para que mediante la aplicación de cursores podamos ir extrayendo los datos para luego proceder a guardarlos dentro de las columnas respectivas de nuestra base de datos.



### Creación del schema



### Importación del CSV



Preview de Importación CSV

## 2.5 Declaración de Tablas

Primeramente, debemos crear las tablas en nuestra herramienta SQL para que así, mediante codificación enviemos la data ya limpiada a la base de datos que estamos creando. Así que para cada entidad vamos a crear una tabla con sus respectivos atributos definiendo tanto el tipo de variable y el tamaño que va a ocupar la misma como las llaves primarias y foráneas en caso de tenerlas.

```

1  #-----
2  DROP DATABASE IF EXISTS bddFinal;
3  CREATE DATABASE bddFinal;
4  USE bddFinal;
5  #-----ENTIDADES-----
6  DROP TABLE IF EXISTS original_language;
7  CREATE TABLE original_language(
8      idOrigLang INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
9      name_original_language VARCHAR(2) NOT NULL
10 );
11 #-----
12 DROP TABLE IF EXISTS Status;
13 CREATE TABLE Status(
14     idStatus INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
15     nameStatus VARCHAR(15) NOT NULL
16 );

```

```

17  #-----
18  DROP TABLE IF EXISTS Movie;
19  CREATE TABLE Movie(
20      idMovie INT PRIMARY KEY NOT NULL,
21      'index' INT NOT NULL ,
22      budget BIGINT NOT NULL,
23      homepage VARCHAR(255),
24      keywords VARCHAR(255),
25      idOrigLang INT NOT NULL,
26      original_title VARCHAR(255) NOT NULL,
27      overview TEXT,
28      popularity DOUBLE NOT NULL,
29      release_date DATE,
30      revenue BIGINT NOT NULL,
31      runtime DOUBLE,
32      idStatus INT NOT NULL,
33      tagline VARCHAR(255),
34      title VARCHAR(255) NOT NULL,
35      vote_average DOUBLE NOT NULL,
36      vote_count INT NOT NULL,
37      FOREIGN KEY (idOrigLang) REFERENCES original_language(idOrigLang),
38      FOREIGN KEY (idStatus) REFERENCES status(idStatus)
39  );
40  #-----

```

*Figura 7. Creación de Tabla “Movie”*

```

40  #-----
41  DROP TABLE IF EXISTS Genre;
42  CREATE TABLE Genre(
43      nameGenre VARCHAR(100) NOT NULL,
44      idGenre INT PRIMARY KEY AUTO_INCREMENT
45  );
46  #-----
47  DROP TABLE IF EXISTS production_countries;
48  CREATE TABLE production_countries(
49      iso_3166_1 VARCHAR(10) PRIMARY KEY NOT NULL,
50      pCountryName VARCHAR(255) NOT NULL
51  );
52  #-----
53  DROP TABLE IF EXISTS production_companies;
54  CREATE TABLE production_companies(
55      pCompanyId INT PRIMARY KEY NOT NULL,
56      pCompanyName VARCHAR(255) NOT NULL
57  );
58  #-----
59  DROP TABLE IF EXISTS spoken_language;
60  CREATE TABLE spoken_language(
61      iso_639_1 VARCHAR(2) PRIMARY KEY NOT NULL,
62      nameSLang VARCHAR(255) NOT NULL
63  );
64  #-----
65

```

*Figura 8. Creación de Tabla “Genre”, “production\_countries”, “production\_companies”*



```

66 #-----
67 DROP TABLE IF EXISTS Persona;
68 CREATE TABLE Persona(
69     idPersona INT PRIMARY KEY NOT NULL,
70     name VARCHAR(255) NOT NULL,
71     gender INT
72 );
73
74 #-----
75 DROP TABLE IF EXISTS Crew;
76 CREATE TABLE Crew(
77     idMovie INT NOT NULL,
78     idPersona INT NOT NULL,
79     job VARCHAR(255),
80     PRIMARY KEY (idMovie, idPersona, job),
81     FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
82     FOREIGN KEY (idPersona) REFERENCES Persona(idPersona)
83 );
84
85 #-----
86 DROP TABLE IF EXISTS `Cast`;
87 CREATE TABLE `Cast`(
88     idMovie INT NOT NULL,
89     idPersona INT NOT NULL,
90     PRIMARY KEY (idMovie, idPersona),
91     FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
92     FOREIGN KEY (idPersona) REFERENCES Persona(idPersona)
93 );

```

*Figura 9. Creación de Tabla “Persona”, “Crew”, “Cast”*

```

95 #-----
96 DROP TABLE IF EXISTS Director;
97 CREATE TABLE Director(
98     idMovie INT NOT NULL,
99     idPersona INT NOT NULL,
100     PRIMARY KEY (idMovie, idPersona),
101     FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
102     FOREIGN KEY (idPersona) REFERENCES Persona(idPersona)
103 );
104
105 #-----RELACIONES-----
106 DROP TABLE IF EXISTS Movie_genres;
107 CREATE TABLE Movie_genres(
108     idMovie INT NOT NULL,
109     idGenre INT NOT NULL,
110     PRIMARY KEY (idMovie, idGenre),
111     FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
112     FOREIGN KEY (idGenre) REFERENCES Genre(idGenre)
113 );
114
115 #-----
116 DROP TABLE IF EXISTS Movie_production_countries;
117 CREATE TABLE Movie_production_countries(
118     idMovie INT NOT NULL,
119     iso_3166_1 VARCHAR(255) NOT NULL,
120     PRIMARY KEY (idMovie, iso_3166_1),
121     FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
122     FOREIGN KEY (iso_3166_1) REFERENCES production_countries(iso_3166_1)

```

*Figura 10. Creación de Tabla “Director”, “Movie\_genres”, “Movie\_production\_countries”*

```

124 DROP TABLE IF EXISTS Movie_production_companies;
125 CREATE TABLE Movie_production_companies(
126     idMovie INT NOT NULL,
127     pCompanyId INT NOT NULL,
128     PRIMARY KEY (idMovie, pCompanyId),
129     FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
130     FOREIGN KEY (pCompanyId) REFERENCES production_companies(pCompanyId)
131 );
132 #-----
133 DROP TABLE IF EXISTS Movie_spoken_languages;
134 CREATE TABLE Movie_spoken_languages(
135     idMovie INT NOT NULL,
136     iso_639_1 VARCHAR(2) NOT NULL,
137     PRIMARY KEY (idMovie, iso_639_1),
138     FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
139     FOREIGN KEY (iso_639_1) REFERENCES spoken_language(iso_639_1)
140 );
141 #-----

```

Figura 11. Creación de Tabla “production\_companies”, “Movie\_spoken\_language”

## 2.6 Cursores

Este procedimiento nos permite que de una forma repetitiva tanto la extracción como la inserción de datos encontrados en cada columna hasta que encuentre el ultimo dato de la misma, donde se mantiene guardada la información almacenada en el cursor hasta que le sobre escriba un valor o hasta que se borre la memoria del mismo. Al finalizar el proceso, solo se puede llamar al proceso tal como una función sin atributos y se realizara el proceso previamente mencionado de extracción e inserción.

```

143 -- Tabla Genre--
144 DROP PROCEDURE IF EXISTS TablaGenre;
145
146 DELIMITER $$
147 CREATE PROCEDURE TablaGenre()
148 BEGIN
149
150     DECLARE done INT DEFAULT FALSE;
151     DECLARE nameGenre VARCHAR(100);
152
153     -- Declarar el cursor
154     DECLARE Cursorgenre CURSOR FOR
155         SELECT DISTINCT CONVERT(REPLACE(REPLACE(genres, 'Science Fiction', 'Science-Fiction'),
156             'TV Movie', 'TV-Movie') USING UTF8MB4) from movies.movie_dataset;
157
158     -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
159     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
160
161     -- Abrir el cursor
162     OPEN Cursorgenre;
163     drop table if exists temperolgenre;
164     SET @sql_text = 'CREATE TABLE temperolgenre (name VARCHAR(100));';
165     PREPARE stmt FROM @sql_text;
166     EXECUTE stmt;
167     DEALLOCATE PREPARE stmt;
168     CursorDirector_loop: LOOP
169         FETCH Cursorgenre INTO nameGenre;
170

```

```

171      -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
172      IF done THEN
173          LEAVE CursorDirector_loop;
174      END IF;
175
176      -- Separar los géneros en una tabla temporal
177      DROP TEMPORARY TABLE IF EXISTS temp_genres;
178      CREATE TEMPORARY TABLE temp_genres (genre VARCHAR(50));
179      SET @_genres = nameGenre;
180      WHILE (LENGTH(@_genres) > 0) DO
181          SET @_genre = TRIM(SUBSTRING_INDEX(@_genres, ' ', 1));
182          INSERT INTO temp_genres (genre) VALUES (@_genre);
183          SET @_genres = SUBSTRING(@_genres, LENGTH(@_genre) + 2);
184      END WHILE;
185
186      -- Insertar los géneros separados en filas individuales
187      INSERT INTO temperolgenre (name)
188      SELECT genre FROM temp_genres;
189  END LOOP CursorDirector_loop;
190
191  select distinct * from temperolgenre;
192  INSERT INTO genre (genre.nameGenre)
193  SELECT DISTINCT name
194  FROM temperolgenre;
195  drop table if exists temperolgenre;
196
197  CLOSE Cursorgenre;
198  END $$
199  DELIMITER ;

```

Figura 12. Cursor “TablaGenre”

```

205  DROP PROCEDURE IF EXISTS TablaStatus;
206  DELIMITER $$
207  CREATE PROCEDURE TablaStatus()
208  BEGIN
209      DECLARE done INT DEFAULT FALSE;
210      DECLARE nameStatus VARCHAR(100);
211
212      -- Declarar el cursor
213      DECLARE CursorStatus CURSOR FOR
214      SELECT DISTINCT CONVERT(status USING UTF8MB4) AS names from movies.movie_dataset;
215
216      -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
217      DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
218
219      -- Abrir el cursor
220      OPEN CursorStatus;
221  CursorStatus_loop: LOOP
222      FETCH CursorStatus INTO nameStatus;
223
224      -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
225      IF done THEN
226          LEAVE CursorStatus_loop;
227      END IF;
228      IF nameStatus IS NULL THEN
229          SET nameStatus = '';
230      END IF;
231      SET @_oStatement = CONCAT('INSERT INTO Status (nameStatus) VALUES (\',nameStatus,')');
232      PREPARE sent1 FROM @_oStatement;
233      EXECUTE sent1;

```

Figura 13. Cursor “TablaStatus”

```

230     END IF;
231     SET @_oStatement = CONCAT('INSERT INTO Status (nameStatus) VALUES (\',nameStatus,')');
232     PREPARE sent1 FROM @_oStatement;
233     EXECUTE sent1;
234     DEALLOCATE PREPARE sent1;
235
236 END LOOP;
237 CLOSE CursorStatus;
238 END $$
239 DELIMITER ;
240 CALL TablaStatus();

```

```

244 DROP PROCEDURE IF EXISTS TablaOriLanguage;
245 DELIMITER $$
246 CREATE PROCEDURE TablaOriLanguage()
247 BEGIN
248     DECLARE done INT DEFAULT FALSE;
249     DECLARE originalLanguage VARCHAR(2);
250     -- Declarar el cursor
251     DECLARE CursorStatus CURSOR FOR
252     SELECT DISTINCT CONVERT(original_language USING UTF8MB4) AS languages from movies.movie_dataset;
253
254     -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
255     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
256
257     -- Abrir el cursor
258     OPEN CursorStatus;
259     CursorStatus_loop: LOOP
260         FETCH CursorStatus INTO originalLanguage;
261
262         -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
263         IF done THEN
264             LEAVE CursorStatus_loop;
265         END IF;
266         IF originalLanguage IS NULL THEN
267             SET originalLanguage = '';
268         END IF;
269         SET @_oStatement = CONCAT('INSERT INTO original_language (name_original_language) VALUES (\',originalLanguage,')');
270         PREPARE sent1 FROM @_oStatement;
271         EXECUTE sent1;
272         DEALLOCATE PREPARE sent1;

```

```

272         DEALLOCATE PREPARE sent1;
273     END LOOP;
274     CLOSE CursorStatus;
275 END $$
276 DELIMITER ;
277
278 CALL TablaOriLanguage();

```

Figura 14. Cursor “TablaOriLanguage”

```

281 #-----Tabla Movie-----
282 DROP PROCEDURE IF EXISTS TablaMovie;
283
284 DELIMITER $$
285 CREATE PROCEDURE TablaMovie()
286 BEGIN
287
288     DECLARE done INT DEFAULT FALSE;
289
290     DECLARE Mov_idMovie INT;
291     DECLARE Mov_index INT;
292     DECLARE Mov_budget BIGINT;
293     DECLARE Mov_homepage VARCHAR(255);
294     DECLARE Mov_keywords VARCHAR(255);
295     DECLARE Mov_name_original_language VARCHAR(2);
296     DECLARE Mov_original_title VARCHAR(255);
297     DECLARE Mov_overview TEXT;
298     DECLARE Mov_popularity DOUBLE;
299     DECLARE Mov_release_date DATE;
300     DECLARE Mov_revenue BIGINT;
301     DECLARE Mov_runtime DOUBLE;
302     DECLARE Mov_nameStatus VARCHAR(15);
303     DECLARE Mov_tagline VARCHAR(255);
304     DECLARE Mov_title VARCHAR(255);
305     DECLARE Mov_vote_average DOUBLE;
306     DECLARE Mov_vote_count INT;
307
308     DECLARE Status_idStatus int;
309
310     DECLARE OL_idOriginal_language int;
311
312     -- Declarar el cursor
313     DECLARE CursorMovie CURSOR FOR
314     SELECT id,'index',budget,homepage,keywords,original_language,original_title,overview,
315           popularity,release_date,revenue,runtime,'status',tagline,title,
316           vote_average,vote_count FROM movies.movie_dataset;
317
318     -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
319     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
320
321     -- Abrir el cursor
322     OPEN CursorMovie;
323
324     CursorMovie_loop: LOOP
325         FETCH CursorMovie INTO Mov_idMovie,Mov_index,Mov_budget, Mov_homepage, Mov_keywords,Mov_name_original_language,
326         Mov_original_title, Mov_overview, Mov_popularity, Mov_release_date, Mov_revenue, Mov_runtime,
327         Mov_nameStatus, Mov_tagline, Mov_title, Mov_vote_average, Mov_vote_count;
328
329         -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
330         IF done THEN
331             LEAVE CursorMovie_loop;
332         END IF;
333
334         SELECT 'idStatus' INTO Status_idStatus
335         FROM Status WHERE nameStatus = Mov_nameStatus;
336
337         SELECT 'idOrigLang' INTO OL_idOriginal_language
338         FROM original_language WHERE name_original_language = Mov_name_original_language;
339
340         INSERT INTO Movie ('idMovie','index',budget,homepage,keywords,idOrigLang,original_title,
341         overview,popularity,release_date,revenue,runtime, idStatus,
342         tagline,title,vote_average,vote_count)
343         VALUES (Mov_idMovie,Mov_index,Mov_budget, Mov_homepage, Mov_keywords,OL_idOriginal_language,
344         Mov_original_title, Mov_overview, Mov_popularity, Mov_release_date, Mov_revenue, Mov_runtime,
345         Status_idStatus, Mov_tagline, Mov_title, Mov_vote_average, Mov_vote_count);
346
347     END LOOP;
348     CLOSE CursorMovie;
349     DELIMITER ;
350
351     CALL TablaMovie ();

```

Figura 15. Cursor “TablaMovie”





```

689 #-----Tabla Crew-----
690 DROP PROCEDURE IF EXISTS TablaCrew;
691
692 DELIMITER $$
693 CREATE PROCEDURE TablaCrew ()
694
695 BEGIN
696
697     DECLARE done INT DEFAULT FALSE;
698     DECLARE idMovie INT;
699     DECLARE idCrew TEXT;
700     DECLARE idJSON TEXT;
701     DECLARE jobJSON TEXT;
702     DECLARE departmentJSON TEXT;
703     DECLARE credit_idJSON TEXT;
704     DECLARE i INT;
705
706     -- Declarar el cursor
707     DECLARE myCursor
708     CURSOR FOR
709     SELECT id, CONVERT(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE
710     (REPLACE(crew, '"', '\'), '{', '{'), '\', '\\'), '\n', '\\n'),
711     '\r', '\\r'), '\t', '\\t'), '\t', '\\t'), '\t', '\\t')
712     USING UTF8mb4 ) FROM movie_dataset;
713
714     -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
715     DECLARE CONTINUE HANDLER
716     FOR NOT FOUND SET done = TRUE ;
717
718

```

```

718 -- Abrir el cursor
719 OPEN myCursor ;
720 cursorLoop: LOOP
721     FETCH myCursor INTO idMovie, idCrew;
722     -- Controlador para buscar cada uno de los arrays
723     SET i = 0;
724     -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
725     IF done THEN
726         LEAVE cursorLoop ;
727     END IF ;
728     WHILE(JSON_EXTRACT(idCrew, CONCAT('$[', i, '].id')) IS NOT NULL) DO
729
730         SET idJSON = JSON_EXTRACT(idCrew, CONCAT('$[', i, '].id')) ;
731         SET jobJSON = JSON_EXTRACT(idCrew, CONCAT('$[', i, '].job')) ;
732         SET departmentJSON = JSON_EXTRACT(idCrew, CONCAT('$[', i, '].department')) ;
733         SET credit_idJSON = JSON_EXTRACT(idCrew, CONCAT('$[', i, '].credit_id')) ;
734
735         SET i = i + 1;
736
737         SET @sql_text = CONCAT('INSERT INTO Crew VALUES (', idMovie, ', ', REPLACE(idJSON, '\\', '\\'), ', ', REPLACE(jobJSON, '\\', '\\'),
738         PREPARE stmt FROM @sql_text;
739         EXECUTE stmt;
740         DEALLOCATE PREPARE stmt;
741
742     END WHILE;
743
744     END LOOP ;
745     CLOSE myCursor ;
746

```

```

753 DROP PROCEDURE IF EXISTS TablaDirector;
754
755 DELIMITER $$
756 CREATE PROCEDURE TablaDirector()
757 BEGIN
758     DECLARE done INT DEFAULT FALSE ;
759     DECLARE idPersonas INT;
760     DECLARE MovId INT;
761     DECLARE MovDirector VARCHAR(100);
762
763     -- Declarar el cursor
764     DECLARE CursorDirector CURSOR FOR
765     SELECT id_director FROM movie_dataset;
766     -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
767     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
768     -- Abrir el cursor
769     OPEN CursorDirector;
770     drop table if exists directorTemp;
771     SET @sql_text = 'CREATE TABLE directorTemp ( idPer INT,
772     idMov INT);';
773     PREPARE stmt FROM @sql_text;
774     EXECUTE stmt;
775     DEALLOCATE PREPARE stmt;
776     CursorMovie_loop: LOOP
777         FETCH CursorDirector INTO MovId, MovDirector;
778

```

Figura 17. Cursor “TablaCrew”

```

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;

    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();

```

Figura 18. Cursor “TablaDirector”



```

351 #-----Tabla Production Companies-----
352 DROP PROCEDURE IF EXISTS TablaProduction_companies;
353
354 DELIMITER $$
355 CREATE PROCEDURE TablaProduction_companies ()
356
357 BEGIN
358
359     DECLARE done INT DEFAULT FALSE ;
360     DECLARE jsonData json ;
361     DECLARE jsonId varchar(250) ;
362     DECLARE jsonLabel varchar(250) ;
363     DECLARE i INT;
364
365     -- Declarar el cursor
366     DECLARE myCursor
367     CURSOR FOR
368     SELECT JSON_EXTRACT(CONVERT(production_companies USING UTF8MB4), '$[*]') FROM movies.movie_dataset ;
369
370     -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
371     DECLARE CONTINUE HANDLER
372     FOR NOT FOUND SET done = TRUE ;
373
374     -- Abrir el cursor
375     OPEN myCursor ;
376     drop table if exists production_companietem;
377     SET @sql_text = 'CREATE TABLE production_companietem ( id int, nameCom VARCHAR(100));';
378     PREPARE stmt FROM @sql_text;
379     EXECUTE stmt;
380
381     -- Abrir el cursor
382     OPEN myCursor ;
383     drop table if exists production_companietem;
384     SET @sql_text = 'CREATE TABLE production_companietem ( id int, nameCom VARCHAR(100));';
385     PREPARE stmt FROM @sql_text;
386     EXECUTE stmt;
387     DEALLOCATE PREPARE stmt;
388     cursorLoop: LOOP
389     FETCH myCursor INTO jsonData;
390
391     -- Controlador para buscar cada uno de los arrays
392     SET i = 0;
393
394     -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
395     IF done THEN
396     LEAVE cursorLoop ;
397     END IF ;
398
399     WHILE(JSON_EXTRACT(jsonData, CONCAT('$[', i, ']')) IS NOT NULL) DO
400
401     SET jsonId = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].id')), '');
402     SET jsonLabel = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].name')), '');
403     SET i = i + 1;
404
405     SET @sql_text = CONCAT('INSERT INTO production_companietem VALUES (', REPLACE(jsonId, '\'', ''), ', ', jsonLabel, '); ');
406     PREPARE stmt FROM @sql_text;
407     EXECUTE stmt;
408     DEALLOCATE PREPARE stmt;
409
410     END WHILE;
411
412     END LOOP ;
413
414     select distinct * from production_companietem;
415     INSERT INTO production_companies
416     SELECT DISTINCT id, nameCom
417     FROM production_companietem;
418     drop table if exists production_companietem;
419     CLOSE myCursor ;
420
421 END$$
422 DELIMITER ;
423
424 CALL TablaProduction_companies();
425

```

Figura 19. Cursor “TablaProduction\_companies”

```

420 DROP PROCEDURE IF EXISTS TablaProduction_countries;
421 DELIMITER $$
422 CREATE PROCEDURE TablaProduction_countries ()
423
424 BEGIN
425
426     DECLARE done INT DEFAULT FALSE ;
427     DECLARE jsonData json ;
428     DECLARE jsonId varchar(250) ;
429     DECLARE jsonLabel varchar(250) ;
430     DECLARE i INT;
431
432     -- Declarar el cursor
433     DECLARE myCursor
434     CURSOR FOR
435     SELECT JSON_EXTRACT(CONVERT(production_countries USING UTF8MB4), '$[*]') FROM movies.movie_dataset ;
436
437     -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
438     DECLARE CONTINUE HANDLER
439     FOR NOT FOUND SET done = TRUE ;
440
441     -- Abrir el cursor
442     OPEN myCursor ;
443     drop table if exists production_companietem;
444     SET @sql_text = 'CREATE TABLE production_countrieTem ( id VARCHAR(100), nameCom VARCHAR(100));';
445     PREPARE stmt FROM @sql_text;
446     EXECUTE stmt;
447     DEALLOCATE PREPARE stmt;
448
449     EXECUTE stmt;
450     DEALLOCATE PREPARE stmt;
451
452     cursorLoop: LOOP
453     FETCH myCursor INTO jsonData;
454
455     -- Controlador para buscar cada uno de los arrays
456     SET i = 0;
457
458     -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
459     IF done THEN
460     LEAVE cursorLoop ;
461     END IF ;
462
463     WHILE(JSON_EXTRACT(jsonData, CONCAT('$[', i, ']')) IS NOT NULL) DO
464
465     SET jsonId = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].iso_3166_1')), '');
466     SET jsonLabel = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].name')), '');
467     SET i = i + 1;
468
469     SET @sql_text = CONCAT('INSERT INTO production_countrieTem VALUES (' , REPLACE(jsonId, '\'', ''), ', ' , jsonLabel, '); ');
470     PREPARE stmt FROM @sql_text;
471     EXECUTE stmt;
472     DEALLOCATE PREPARE stmt;
473
474     END WHILE;
475
476     END LOOP ;
477
478     WHILE(JSON_EXTRACT(jsonData, CONCAT('$[', i, ']')) IS NOT NULL) DO
479
480     SET jsonId = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].iso_3166_1')), '');
481     SET jsonLabel = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].name')), '');
482     SET i = i + 1;
483
484     SET @sql_text = CONCAT('INSERT INTO production_countrieTem VALUES (' , REPLACE(jsonId, '\'', ''), ', ' , jsonLabel, '); ');
485     PREPARE stmt FROM @sql_text;
486     EXECUTE stmt;
487     DEALLOCATE PREPARE stmt;
488
489     END WHILE;
490
491     END LOOP ;
492
493     select distinct * from production_countrieTem;
494     INSERT INTO production_countries
495     SELECT DISTINCT id, nameCom
496     FROM production_countrieTem;
497     drop table if exists production_countrieTem;
498     CLOSE myCursor ;
499
500 END$$
501 DELIMITER ;
502
503 CALL TablaProduction_countries();
504

```

Figura 20. Cursor “TablaProduction\_countries”

```

3 DROP PROCEDURE IF EXISTS TablaSpoken_Languages ;
4 DELIMITER $$
5 CREATE PROCEDURE TablaSpoken_Languages ()
6 BEGIN
7     DECLARE done INT DEFAULT FALSE ;
8     DECLARE jsonData json ;
9     DECLARE jsonId varchar(250) ;
10    DECLARE jsonLabel varchar(250) ;
11    DECLARE i INT;
12
13    -- Declarar el cursor
14    DECLARE myCursor
15    CURSOR FOR
16        SELECT JSON_EXTRACT(CONVERT(spoken_languages USING UTF8), '$[*]') FROM movies.movie_dataset ;
17
18    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
19    DECLARE CONTINUE HANDLER
20    FOR NOT FOUND SET done = TRUE ;
21
22    -- Abrir el cursor
23    OPEN myCursor ;
24    drop table if exists spokenLanguageTem;
25    SET @sql_text = 'CREATE TABLE spokenLanguageTem ( iso VARCHAR(5), nameLang VARCHAR(100));';
26    PREPARE stmt FROM @sql_text;
27    EXECUTE stmt;
28    DEALLOCATE PREPARE stmt;
29    cursorLoop: LOOP
30        FETCH myCursor INTO jsonData;
31
32        -- Controlador para buscar cada uno de los arrays
33        SET i = 0;
34
35        -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
36        IF done THEN
37            LEAVE cursorLoop ;
38        END IF ;
39
40        WHILE(JSON_EXTRACT(jsonData, CONCAT('$[', i, ']')) IS NOT NULL ) DO
41
42            SET jsonId = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].iso_639_1')), '');
43            SET jsonLabel = IFNULL(JSON_EXTRACT (jsonData, CONCAT('$[', i, '].name')), '');
44            SET i = i + 1;
45
46            SET @sql_text = CONCAT('INSERT INTO spokenLanguageTem VALUES (' , jsonId, ', ', jsonLabel, '); ');
47            PREPARE stmt FROM @sql_text;
48            EXECUTE stmt;
49            DEALLOCATE PREPARE stmt;
50        END WHILE;
51    END LOOP;
52
53    select distinct * from spokenLanguageTem;
54    INSERT INTO spoken_language
55    SELECT DISTINCT iso, nameLang
56    FROM spokenLanguageTem;
57    drop table if exists spokenLanguageTem;
58    CLOSE myCursor ;
59
60
61
62 DELIMITER $$;
63 CALL TablaSpoken_Languages ();

```

Figura 21. Cursor “TablaSpoken\_Languages”

```

411 DROP PROCEDURE IF EXISTS TablaMovie_production_companies;
412
413 DELIMITER $$
414 CREATE PROCEDURE TablaMovie_production_companies ()
415
416 BEGIN
417
418     DECLARE done INT DEFAULT FALSE;
419     DECLARE idMovie int;
420     DECLARE idProdComp JSON;
421     DECLARE idJSON text;
422     DECLARE i INT;
423
424     -- Declarar el cursor
425     DECLARE myCursor
426     CURSOR FOR
427     SELECT id, production_companies FROM movies.movie_dataset;
428
429     -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
430     DECLARE CONTINUE HANDLER
431     FOR NOT FOUND SET done = TRUE ;
432
433     -- Abrir el cursor
434     OPEN myCursor ;
435
436     drop table if exists MovieProdCompTemp;
437     SET @sql_text = 'CREATE TABLE MovieProdCompTemp ( id int, idGenre int );';
438     PREPARE stmt FROM @sql_text;
439     EXECUTE stmt;
440     DEALLOCATE PREPARE stmt;
441
442     cursorLoop: LOOP
443
444         FETCH myCursor INTO idMovie, idProdComp;
445
446         -- Controlador para buscar cada uno de los arrays
447         SET i = 0;
448
449         -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
450         IF done THEN
451             LEAVE cursorLoop ;
452         END IF ;
453
454         WHILE(JSON_EXTRACT(idProdComp, CONCAT('$[', i, '].id')) IS NOT NULL) DO
455
456             SET idJSON = JSON_EXTRACT(idProdComp, CONCAT('$[', i, '].id')) ;
457             SET i = i + 1;
458
459             SET @sql_text = CONCAT('INSERT INTO MovieProdCompTemp VALUES (', idMovie, ', ', REPLACE(idJSON, '\\', '\\\\'), '); ');
460             PREPARE stmt FROM @sql_text;
461             EXECUTE stmt;
462             DEALLOCATE PREPARE stmt;
463
464         END WHILE;
465
466     END LOOP ;
467
468     select distinct * from MovieProdCompTemp;
469     INSERT INTO Movie_production_companies
470     SELECT DISTINCT id, idGenre
471     FROM MovieProdCompTemp;
472     drop table if exists MovieProdCompTemp;
473     CLOSE myCursor ;
474
475 END$$
476 DELIMITER ;
477
478 call TablaMovie_production_companies();

```

Figura 22. Cursor “TablaMovie\_production\_companies”

```

481 DROP PROCEDURE IF EXISTS TablaMovie_production_countries;
482
483 DELIMITER $$
484 CREATE PROCEDURE TablaMovie_production_countries ()
485
486 BEGIN
487
488     DECLARE done INT DEFAULT FALSE;
489     DECLARE idMovie int;
490     DECLARE idProdCoun text;
491     DECLARE idJSON text;
492     DECLARE i INT;
493
494     -- Declarar el cursor
495     DECLARE myCursor
496     CURSOR FOR
497     SELECT id, production_countries FROM movies.movie_dataset;
498
499     -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
500     DECLARE CONTINUE HANDLER
501     FOR NOT FOUND SET done = TRUE ;
502
503     -- Abrir el cursor
504     OPEN myCursor ;
505
506     drop table if exists MovieProdCompTemp;
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999

```

Figura 23. Cursor “TablaMovie\_production\_companies”

```

481 DROP PROCEDURE IF EXISTS TablaMovie_production_countries;
482
483 DELIMITER $$
484 CREATE PROCEDURE TablaMovie_production_countries ()
485
486 BEGIN
487
488     DECLARE done INT DEFAULT FALSE;
489     DECLARE idMovie int;
490     DECLARE idProdCounr text;
491     DECLARE idJSON text;
492     DECLARE i INT;
493
494     -- Declarar el cursor
495     DECLARE myCursor
496     CURSOR FOR
497     SELECT id, production_countries FROM movies.movie_dataset;
498
499     -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
500     DECLARE CONTINUE HANDLER
501     FOR NOT FOUND SET done = TRUE ;
502
503     -- Abrir el cursor
504     OPEN myCursor ;
505
506     drop table if exists MovieProdCompTemp;
507
508
509     drop table if exists MovieProdCompTemp;
510
511     SET @sql_text = 'CREATE TABLE MovieProdCountrTemp ( id int, prodCountr varchar(255) );';
512     PREPARE stmt FROM @sql_text;
513     EXECUTE stmt;
514     DEALLOCATE PREPARE stmt;
515
516     cursorLoop: LOOP
517
518         FETCH myCursor INTO idMovie, idProdCounr;
519
520         -- Controlador para buscar cada uno de los arrays
521         SET i = 0;
522
523         -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
524         IF done THEN
525             LEAVE cursorLoop ;
526         END IF ;
527
528         WHILE (JSON_EXTRACT(idProdCounr, CONCAT('[', i, '].iso_3166_1')) IS NOT NULL) DO
529
530             SET idJSON = JSON_EXTRACT(idProdCounr, CONCAT('[', i, '].iso_3166_1')) ;
531             SET i = i + 1;
532
533             SET @sql_text = CONCAT('INSERT INTO MovieProdCountrTemp VALUES (', idMovie, ', ', REPLACE(idJSON, '\'', ''), '); ');
534             PREPARE stmt FROM @sql_text;
535             EXECUTE stmt;
536             DEALLOCATE PREPARE stmt;
537
538         WHILE (JSON_EXTRACT(idProdCounr, CONCAT('[', i, '].iso_3166_1')) IS NOT NULL) DO
539
540             SET idJSON = JSON_EXTRACT(idProdCounr, CONCAT('[', i, '].iso_3166_1')) ;
541             SET i = i + 1;
542
543             SET @sql_text = CONCAT('INSERT INTO MovieProdCountrTemp VALUES (', idMovie, ', ', REPLACE(idJSON, '\'', ''), '); ');
544             PREPARE stmt FROM @sql_text;
545             EXECUTE stmt;
546             DEALLOCATE PREPARE stmt;
547
548         END WHILE;
549     END LOOP ;
550
551     select distinct * from MovieProdCountrTemp;
552     INSERT INTO movie_production_countries
553     SELECT DISTINCT id, prodCountr
554     FROM MovieProdCountrTemp;
555     drop table if exists MovieProdCountrTemp;
556     CLOSE myCursor ;
557
558 END$$
559 DELIMITER ;
560
561 call TablaMovie_production_countries();

```

Figura 24. Cursor “TablaMovie\_production\_countries”



```

551 DROP PROCEDURE IF EXISTS TablaMovie_spoken_languages;
552
553 DELIMITER $$
554 CREATE PROCEDURE TablaMovie_spoken_languages ()
555
556 BEGIN
557
558     DECLARE done INT DEFAULT FALSE;
559     DECLARE idMovie int;
560     DECLARE idSpokLang text;
561     DECLARE idJSON text;
562     DECLARE i INT;
563
564     -- Declarar el cursor
565     DECLARE myCursor
566     CURSOR FOR
567     SELECT id, spoken_languages FROM movies.movie_dataset;
568
569     -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
570     DECLARE CONTINUE HANDLER
571     FOR NOT FOUND SET done = TRUE ;
572
573     -- Abrir el cursor
574     OPEN myCursor ;
575
576     drop table if exists MovieProdCompTemp;
577
578     SET @sql_text = 'CREATE TABLE MovieSpokenLanguagesTemp ( id int, spokenLang varchar(255) );';
579     PREPARE stmt FROM @sql_text;

```

```

574 drop table if exists MovieProdCompTemp;
575
576 SET @sql_text = 'CREATE TABLE MovieSpokenLanguagesTemp ( id int, spokenLang varchar(255) );';
577 PREPARE stmt FROM @sql_text;
578 EXECUTE stmt;
579 DEALLOCATE PREPARE stmt;
580
581 cursorLoop: LOOP
582
583     FETCH myCursor INTO idMovie, idSpokLang;
584
585     -- Controlador para buscar cada uno de los arrays
586     SET i = 0;
587
588     -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
589     IF done THEN
590         LEAVE cursorLoop ;
591     END IF ;
592
593     WHILE(JSON_EXTRACT(idSpokLang, CONCAT('$[', i, '].iso_639_1')) IS NOT NULL) DO
594
595         SET idJSON = JSON_EXTRACT(idSpokLang, CONCAT('$[', i, '].iso_639_1')) ;
596         SET i = i + 1;
597
598         SET @sql_text = CONCAT('INSERT INTO MovieSpokenLanguagesTemp VALUES (', idMovie, ', ', REPLACE(idJSON, '\\', '\\\\'), '); ');
599         PREPARE stmt FROM @sql_text;
600         EXECUTE stmt;
601         DEALLOCATE PREPARE stmt;

```

Figura 25. Cursor “TablaMovie\_spoken\_languages”

Dentro de este proyecto se tomó la decisión de manera de hacer la población de datos solamente de manera directa creando primero todas las tablas que vamos a ocupar según el modelo realizado al inicio del todo, para luego extraer la data, limpiarla e insertarla en estas tablas; teniendo así nuestra base de datos poblada y lista para ocuparla en un ámbito de investigación.

Se acota que dentro del trabajo se pudo poblar toda la base de datos a excepción de la columna “cast” por el hecho de que para hacerlo con SQL se requería un proceso de análisis de todos los datos dentro de esta columna para luego aplicar *tokens* e ir arreglando cada

nombre de cada actor de manera casi individual, ya que en algunos casos se podían arreglar varios datos con un solo *token* pero al notar que son alrededor de 4803 filas de datos, se vuelve un trabajo extenso realizarlo de esta manera. Por otro lado, por medio de programación funcional se pudo encontrar un método de encontrar la mayor cantidad de nombres de dicha columna aplicando un servicio web llamado “meaning cloud” que su mismo propósito es buscar cierto tipo de palabras dentro de una rawdata.

### 3 Conclusiones

- Este proyecto sirvió de mucho conocimiento al saber tanto utilizando programación funcional como por medio de SQL como aplicar ciertas funciones y las ventajas de cada lenguaje para realizar una tarea que se nos puede presentar dentro de nuestro ambiente laboral a futuro.
- A pesar de los inconvenientes se pudo lograr cumplir el objetivo del proyecto que era aplicar los conocimientos adquiridos durante el presente periodo académico para realizar un trabajo que considero de un alto nivel de complejidad.