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

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.

Asi 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.

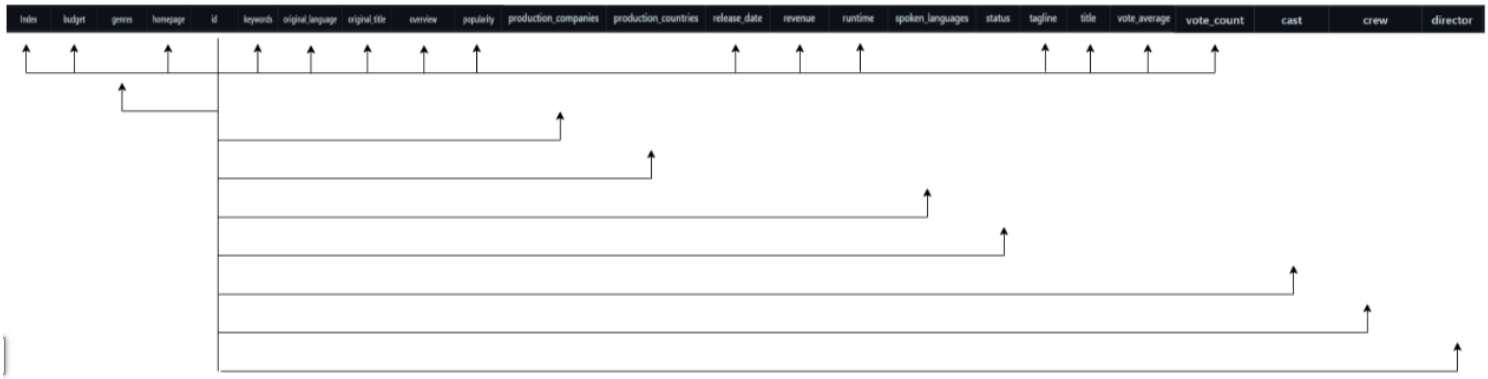


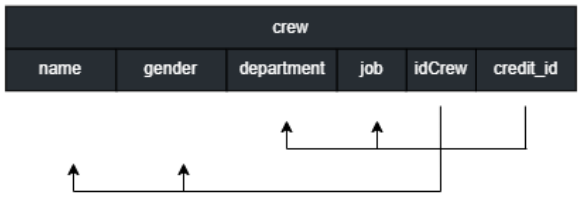
Figura 1. Tabla de dependencias primer modelo

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 → {name, gender}

credit_id → {department, job}

id → {index, budget, homepage, keywords, original_title, overview, popularity, release_date, revenue, runtime, tagline, title, vote_avarage, vote_count}

id → {idGenre, name_Genre}

id → {name_original_language, idOrigLang}

id → {pcompanyId, pcompanyName}

id → {iso_3166_1, pcountryName}

id → {nameSLang, iso_639_1}

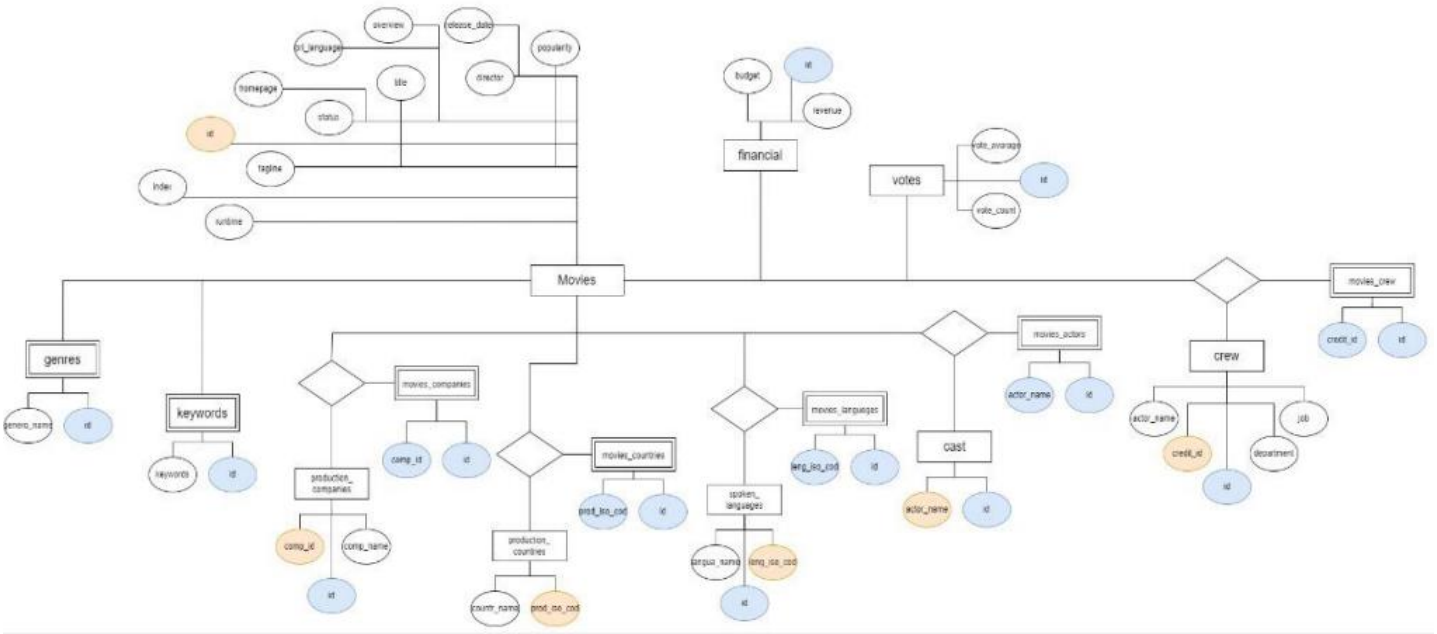
id → {idStatus, nameStatus}

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.

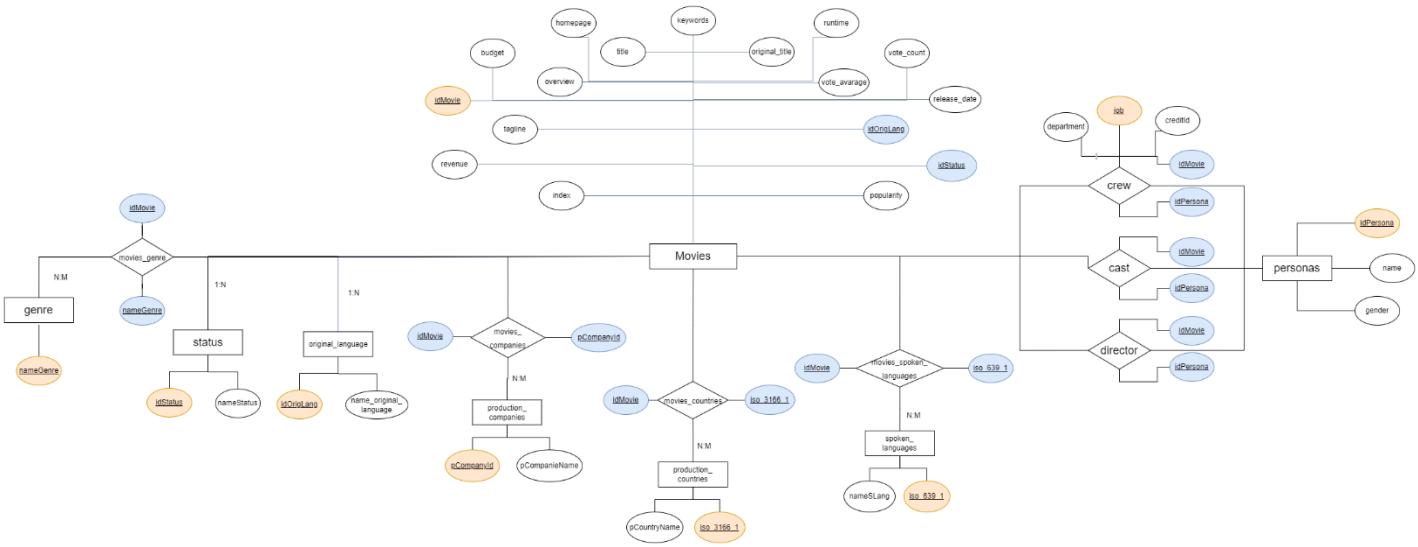
Para ilustrar de manera gráfica nuestro modelo, utilizamos la herramienta de app.diagrams.net. Esta herramienta nos permitió visualizar las relaciones entre las entidades y atributos del dataset a trabajar. Al tener una representación gráfica, es más fácil comprender la estructura y relaciones de los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



Asi 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.

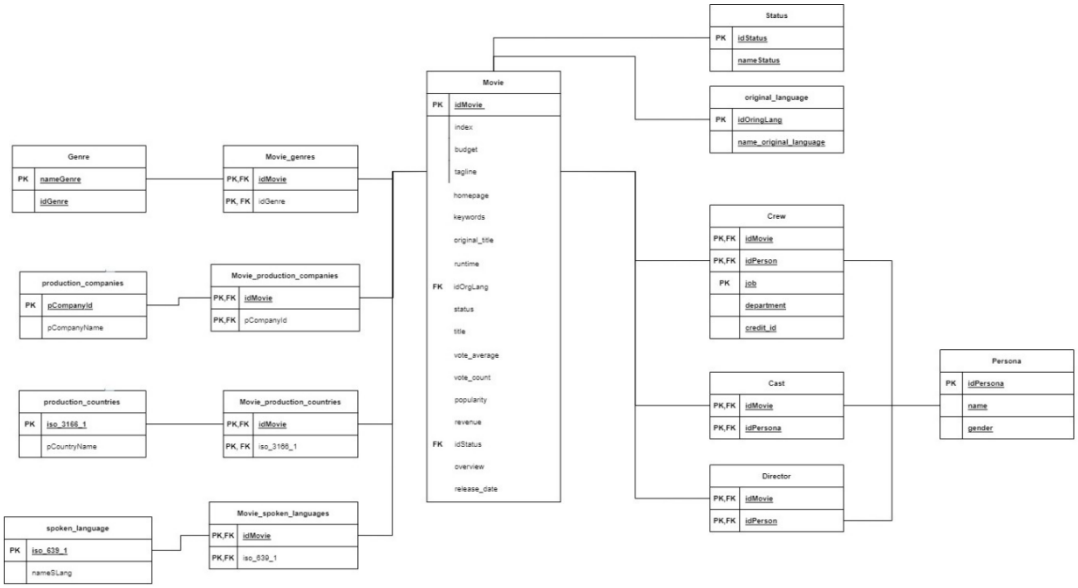


Figura 5. Modelo Lógico

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.

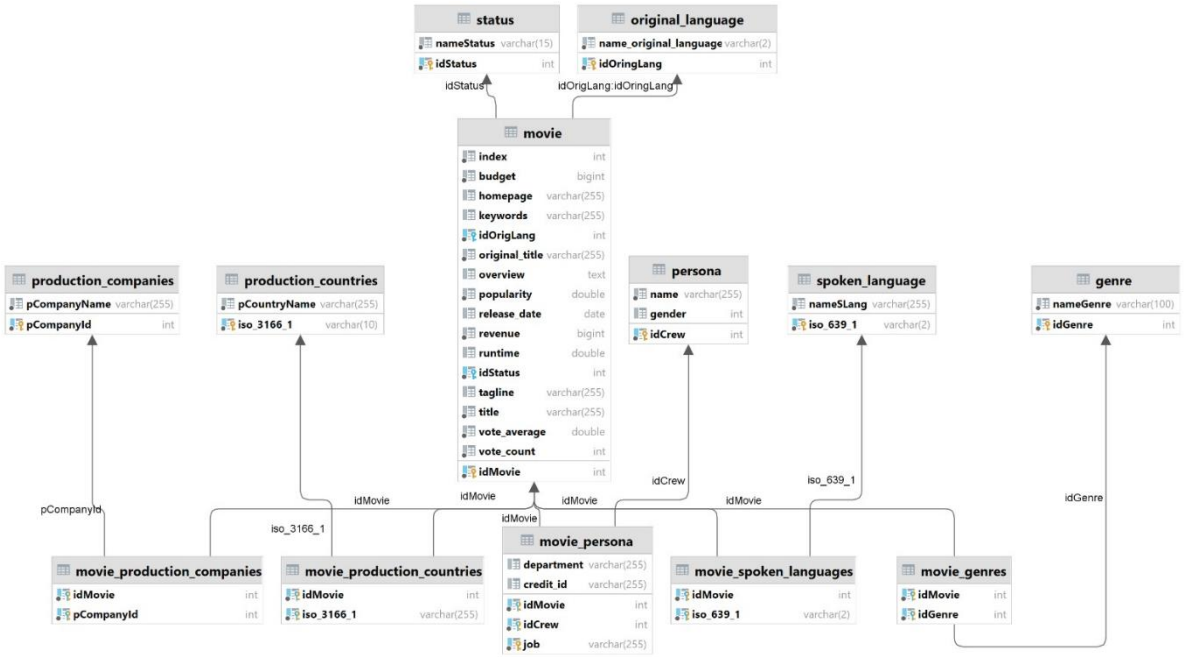


Figura 6. Modelo Físico

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.

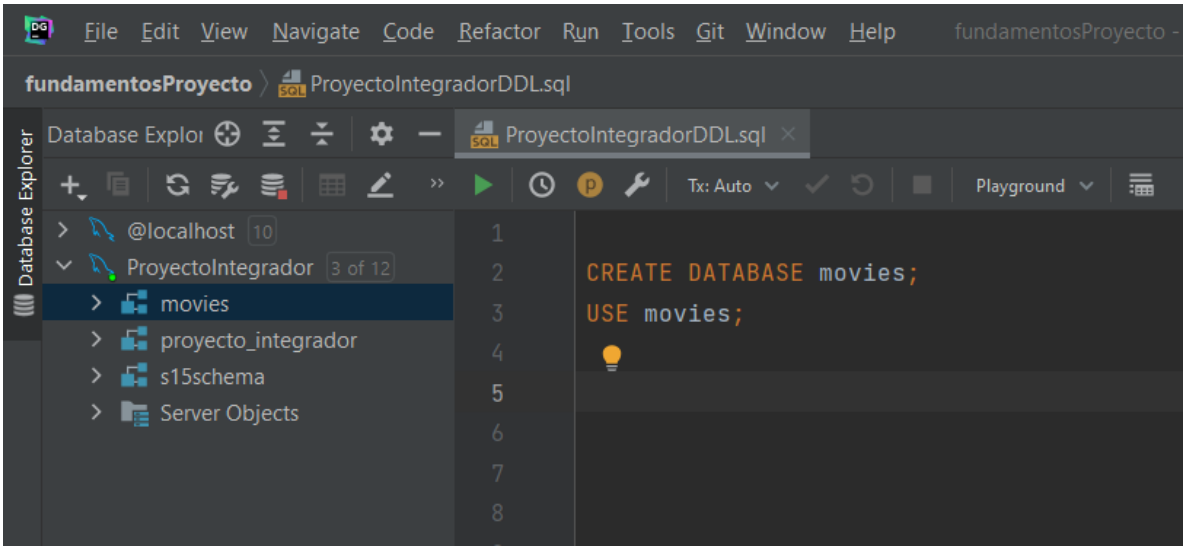
```
SELECT id,
JSON_VALID(CONVERT (
REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(crew,
'"', '\''),
'{\'', '{"'),
'\': \'', '": "'),
'\', \'', '", "'),
'\': ', '": '),
', \'', ', "')
USING UTF8mb4 )) AS Valid_YN,
CONVERT (
REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(crew,
'"', '\''),
'{\'', '{"'),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

Dentro de la columna genres se debe tomar encuentra 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

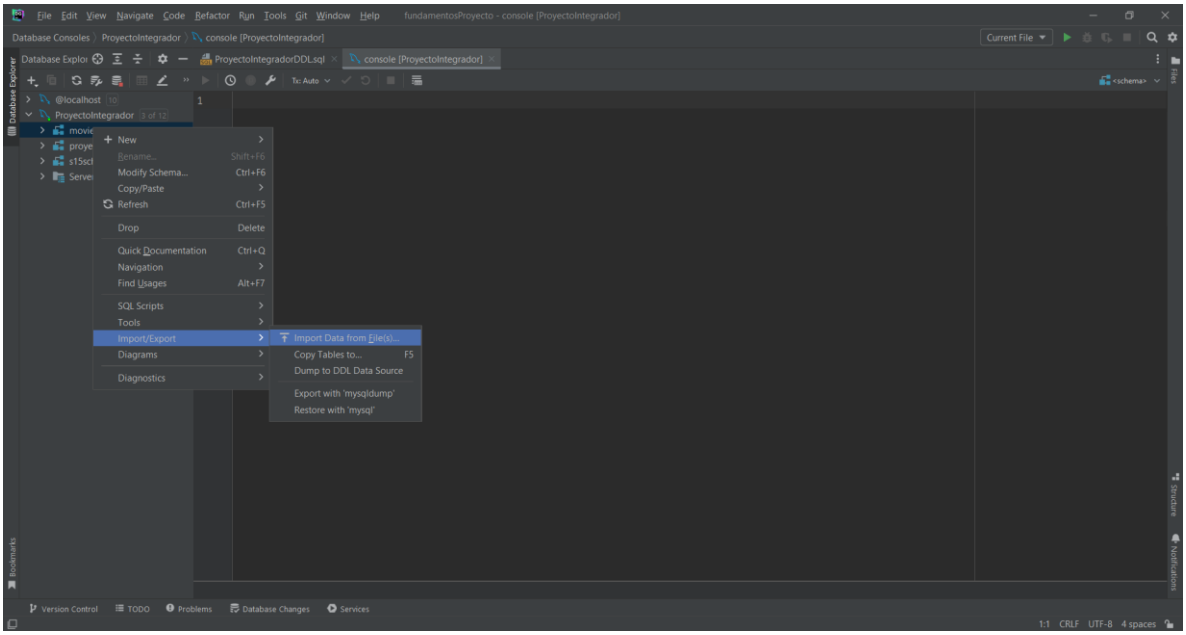
En el caso de los tipos de dato numéricos, se debe tener en cuenta que vamos a tener filas con datos en blanco, por lo que se debe ir corrigiendo eso bien al momento de que se las va a insertar en la base de data.

2.4 Importación del .csv al SQL

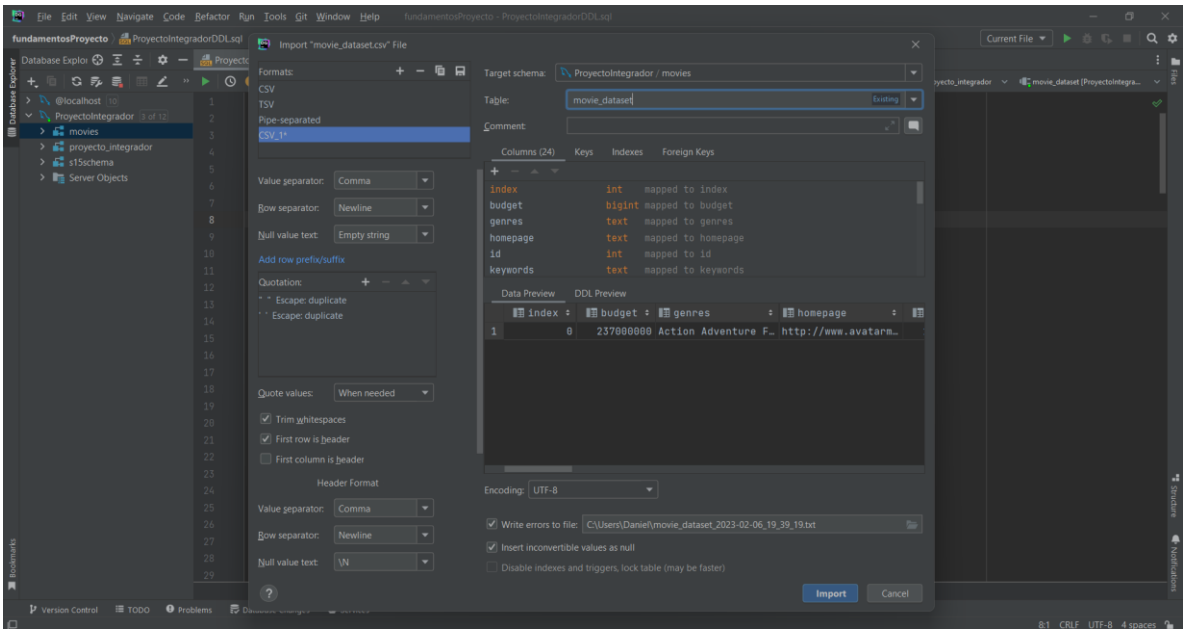
Como vamos a trabajar puramente desde SQL tenemos que tener una manera de extraer los datos del archivo SQL, por lo que vamos a importar todo el archivo dentro de nuestro esquema o base de datos donde vayamos a trabajar para que mediante la aplicación de cursores podamos ir extrayendo los datos para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



Creación del schema



Importación del CSV

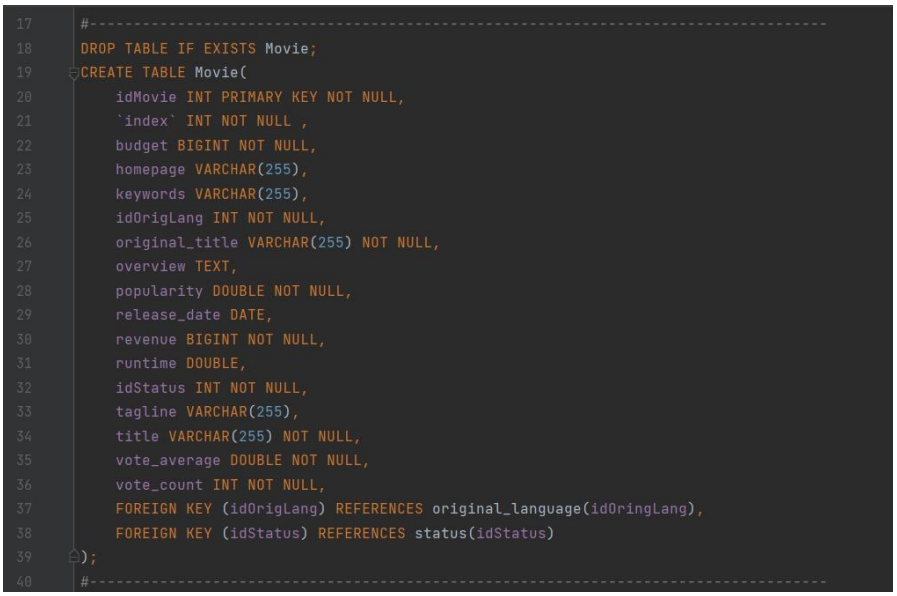


Preview de Importación CSV

2.5 Declaración de Tablas

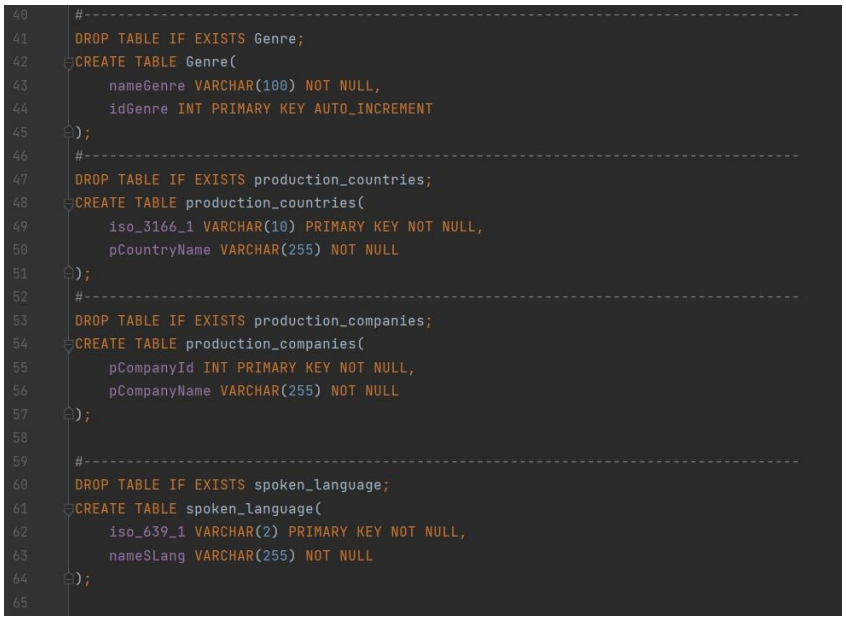
Primeramente, debemos crear las tablas en nuestra herramienta SQL para que así, mediante codificación enviemos la data ya limpiada a la base de datos que estamos creando. Así que para cada entidad vamos a crear una tabla con sus respectivos atributos definiendo tanto el tipo de variable y el tamaño que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

```sql
#-------------------------------------------------------------------------------
DROP DATABASE IF EXISTS bddFinal;
CREATE DATABASE bddFinal;
USE bddFinal;
#-----------------------------------ENTIDADES---------------------------
DROP TABLE IF EXISTS original_language;
CREATE TABLE original_language(
    idOringLang INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
    name_original_language VARCHAR(2) NOT NULL
);
#-------------------------------------------------------------------------------
DROP TABLE IF EXISTS Status;
CREATE TABLE Status(
    idStatus INT AUTO_INCREMENT PRIMARY KEY NOT NULL,
    nameStatus VARCHAR(15) NOT NULL
);
```



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

Figura 7. Creación de Tabla "Movie"

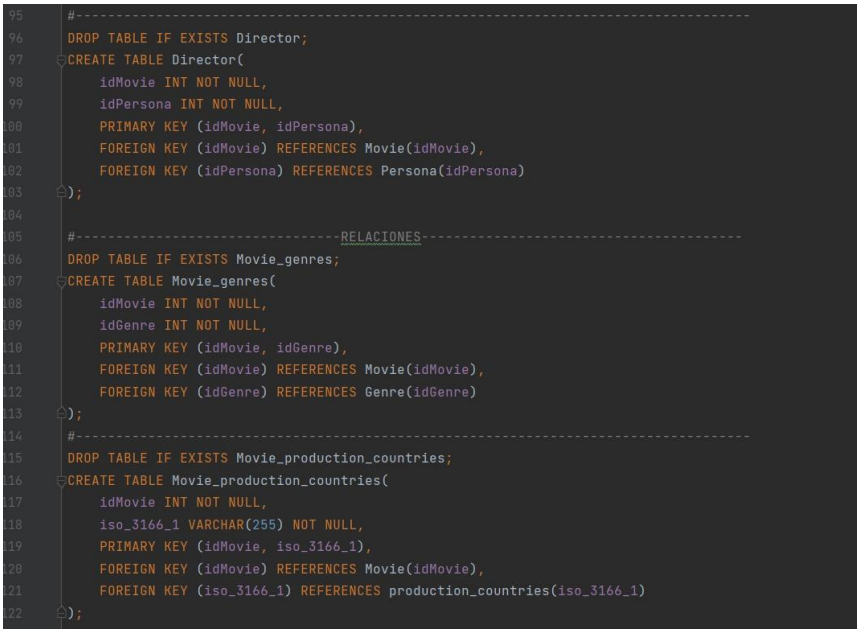


```
#-----------------------------------------------------------------------------------
DROP TABLE IF EXISTS Genre;
CREATE TABLE Genre(
    nameGenre VARCHAR(100) NOT NULL,
    idGenre INT PRIMARY KEY AUTO_INCREMENT
);
#-----------------------------------------------------------------------------------
DROP TABLE IF EXISTS production_countries;
CREATE TABLE production_countries(
    iso_3166_1 VARCHAR(10) PRIMARY KEY NOT NULL,
    pCountryName VARCHAR(255) NOT NULL
);
#-----------------------------------------------------------------------------------
DROP TABLE IF EXISTS production_companies;
CREATE TABLE production_companies(
    pCompanyId INT PRIMARY KEY NOT NULL,
    pCompanyName VARCHAR(255) NOT NULL
);

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

```

Figura 8. Creación de Tabla "Genre", "production_countries", "production_companies"

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

#-------------------------------------------------------------------------------
DROP TABLE IF EXISTS Crew;
CREATE TABLE Crew(
    idMovie INT NOT NULL,
    idPersona INT NOT NULL,
    job VARCHAR(255),
    PRIMARY KEY (idMovie, idPersona, job),
    FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
    FOREIGN KEY (idPersona) REFERENCES Persona(idPersona)
);

#-------------------------------------------------------------------------------
DROP TABLE IF EXISTS `Cast`;
CREATE TABLE `Cast`(
    idMovie INT NOT NULL,
    idPersona INT NOT NULL,
    PRIMARY KEY (idMovie, idPersona),
    FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
    FOREIGN KEY (idPersona) REFERENCES Persona(idPersona)
);
```

Figura 9. Creación de Tabla "Persona", "Crew", "Cast"



```sql
#-------------------------------------------------------------------------------
DROP TABLE IF EXISTS Director;
CREATE TABLE Director(
    idMovie INT NOT NULL,
    idPersona INT NOT NULL,
    PRIMARY KEY (idMovie, idPersona),
    FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
    FOREIGN KEY (idPersona) REFERENCES Persona(idPersona)
);

#--------------------------------RELACIONES-------------------------------------
DROP TABLE IF EXISTS Movie_genres;
CREATE TABLE Movie_genres(
    idMovie INT NOT NULL,
    idGenre INT NOT NULL,
    PRIMARY KEY (idMovie, idGenre),
    FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
    FOREIGN KEY (idGenre) REFERENCES Genre(idGenre)
);
#-------------------------------------------------------------------------------
DROP TABLE IF EXISTS Movie_production_countries;
CREATE TABLE Movie_production_countries(
    idMovie INT NOT NULL,
    iso_3166_1 VARCHAR(255) NOT NULL,
    PRIMARY KEY (idMovie, iso_3166_1),
    FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
    FOREIGN KEY (iso_3166_1) REFERENCES production_countries(iso_3166_1)
);
```

Figura 10. Creación de Tabla "Director", "Movie_genres", "Movie_production_countries"

```sql
DROP TABLE IF EXISTS Movie_production_companies;
CREATE TABLE Movie_production_companies(
    idMovie INT NOT NULL,
    pCompanyId INT NOT NULL,
    PRIMARY KEY (idMovie, pCompanyId),
    FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
    FOREIGN KEY (pCompanyId) REFERENCES production_companies(pCompanyId)
);
#-------------------------------------------------------------------------------------
DROP TABLE IF EXISTS Movie_spoken_languages;
CREATE TABLE Movie_spoken_languages(
    idMovie INT NOT NULL,
    iso_639_1 VARCHAR(2) NOT NULL,
    PRIMARY KEY (idMovie, iso_639_1),
    FOREIGN KEY (idMovie) REFERENCES Movie(idMovie),
    FOREIGN KEY (iso_639_1) REFERENCES spoken_language(iso_639_1)
);
#-------------------------------------------------------------------------------------
```

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.

```sql
-- Tabla 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 movies.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;
```

```
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"

```
    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();
```

```
DROP PROCEDURE IF EXISTS TablaOriLanguage;
DELIMITER $$
CREATE PROCEDURE TablaOriLanguage()
BEGIN
DECLARE done INT DEFAULT FALSE;
DECLARE originalLanguage VARCHAR(2);
 -- Declarar el cursor
DECLARE CursorStatus CURSOR FOR
    SELECT DISTINCT CONVERT(original_language USING UTF8MB4) AS languages from movies.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 originalLanguage;

-- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE CursorStatus_loop;
    END IF;
    IF originalLanguage IS NULL THEN
        SET originalLanguage = '';
    END IF;
    SET @_oStatement = CONCAT('INSERT INTO original_language (name_original_language) VALUES (\'',originalLanguage,'\');');
    PREPARE sent1 FROM @_oStatement;
    EXECUTE sent1;
    DEALLOCATE PREPARE sent1;
```

```
    DEALLOCATE PREPARE sent1;
END LOOP;
CLOSE CursorStatus;
END $$
DELIMITER ;

CALL TablaOriLanguage();
```

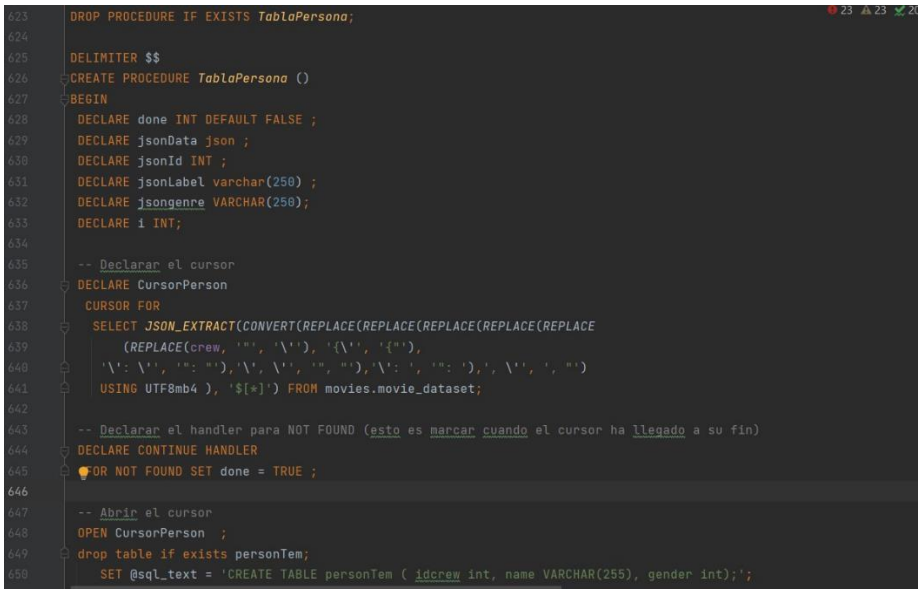
Figura 14. Cursor "TablaOriLanguage"

```sql
#------------------Tabla 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;
```

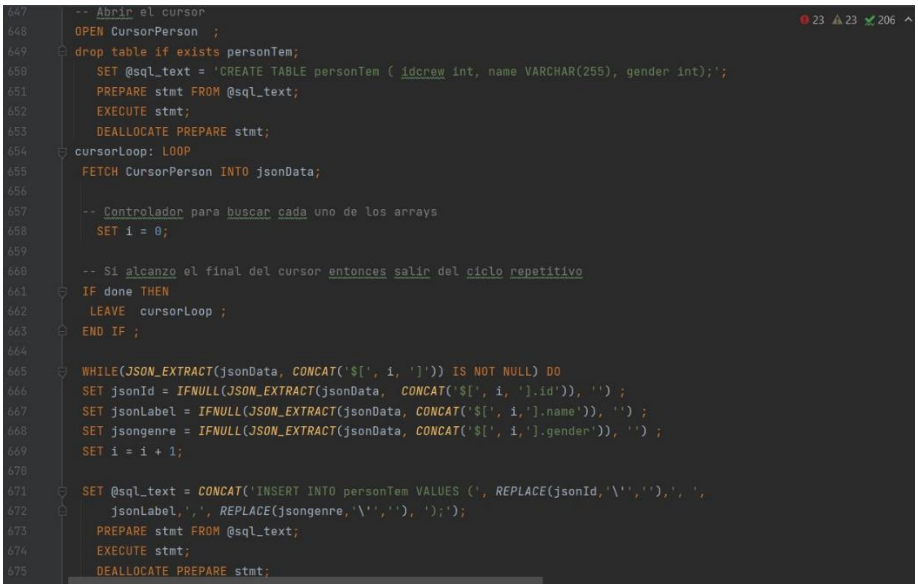
```sql
DECLARE Status_idStatus int;
DECLARE OL_idOriginal_language int;

 -- 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 movies.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 `idOringLang` INTO OL_idOriginal_language
                        FROM original_language WHERE name_original_language = Mov_name_original_language;
```

```sql

    INSERT INTO Movie (`idMovie`,`index`,budget,homepage,keywords,idOrigLang,original_title,
                       overview,popularity,release_date,revenue,runtime, idStatus,
                        tagline,title,vote_average,vote_count)
    VALUES (Mov_idMovie,Mov_index,Mov_budget, Mov_homepage, Mov_keywords,OL_idOriginal_language,
        Mov_original_title, Mov_overview, Mov_popularity, Mov_release_date, Mov_revenue, Mov_runtime,
        Status_idStatus, Mov_tagline, Mov_title, Mov_vote_average, Mov_vote_count);

END LOOP;
CLOSE CursorMovie;
END $$
DELIMITER ;

CALL TablaMovie ();
```

Figura 15. Cursor "TablaMovie"



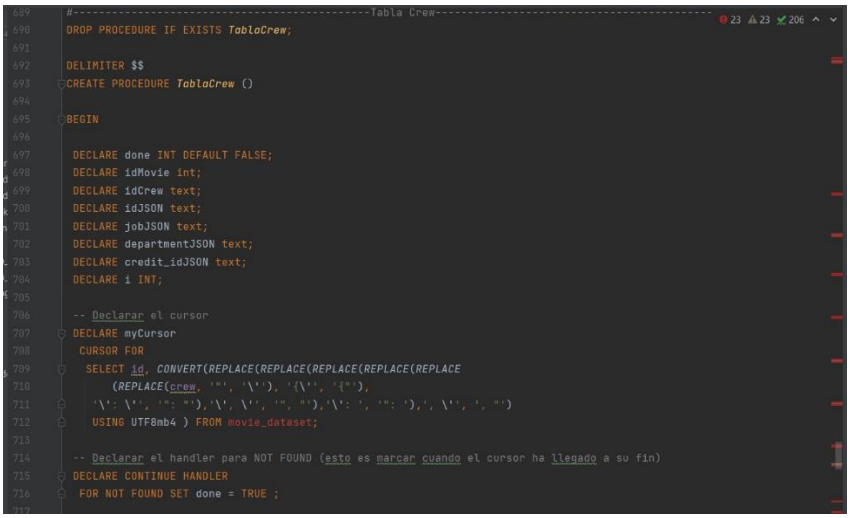
```
DROP PROCEDURE IF EXISTS TablaPersona;

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 movies.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);';
```



```
 -- 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;
```

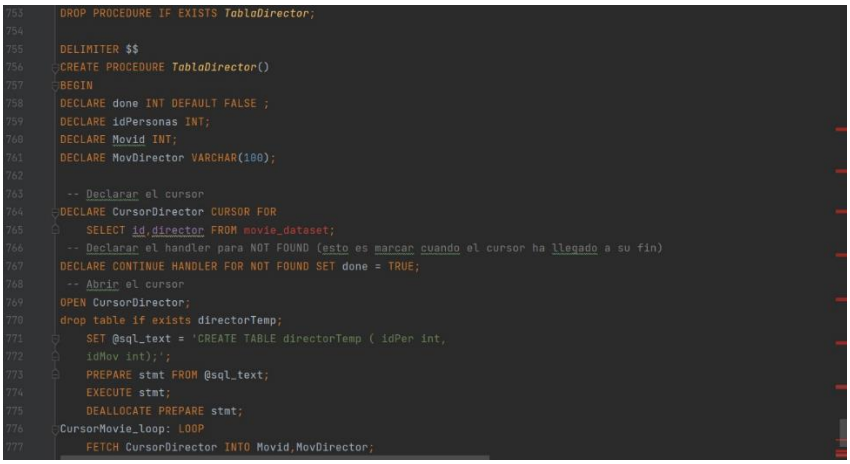
```
  SET @sql_text = CONCAT('INSERT INTO personTem VALUES (', REPLACE(jsonId,'\'',''),', ',
      jsonLabel,',', REPLACE(jsongenre,'\'',''), ');');
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;

  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();
```

Figura 16. Cursor "TablaPersona"



```
#-----------------------------------------Tabla 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,'\'
     PREPARE stmt FROM @sql_text;
     EXECUTE stmt;
     DEALLOCATE PREPARE stmt;

  END WHILE;

 END LOOP ;
 CLOSE myCursor ;
```



```
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;
```

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;
    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"

```
#------------------Tabla Production Companies------------------
DROP PROCEDURE IF EXISTS TablaProduction_companies;

DELIMITER $$
CREATE PROCEDURE TablaProduction_companies ()

BEGIN

 DECLARE done INT DEFAULT FALSE ;
 DECLARE jsonData json ;
 DECLARE jsonId varchar(250) ;
 DECLARE jsonLabel varchar(250) ;
 DECLARE i INT;

 -- Declarar el cursor
 DECLARE myCursor
  CURSOR FOR
   SELECT JSON_EXTRACT(CONVERT(production_companies USING UTF8MB4), '$[*]') FROM movies.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(100));';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
```

```
 -- Abrir el cursor
 OPEN myCursor  ;
 drop table if exists production_companietem;
    SET @sql_text = 'CREATE TABLE production_companieTem ( id int, nameCom VARCHAR(100));';
    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();

```

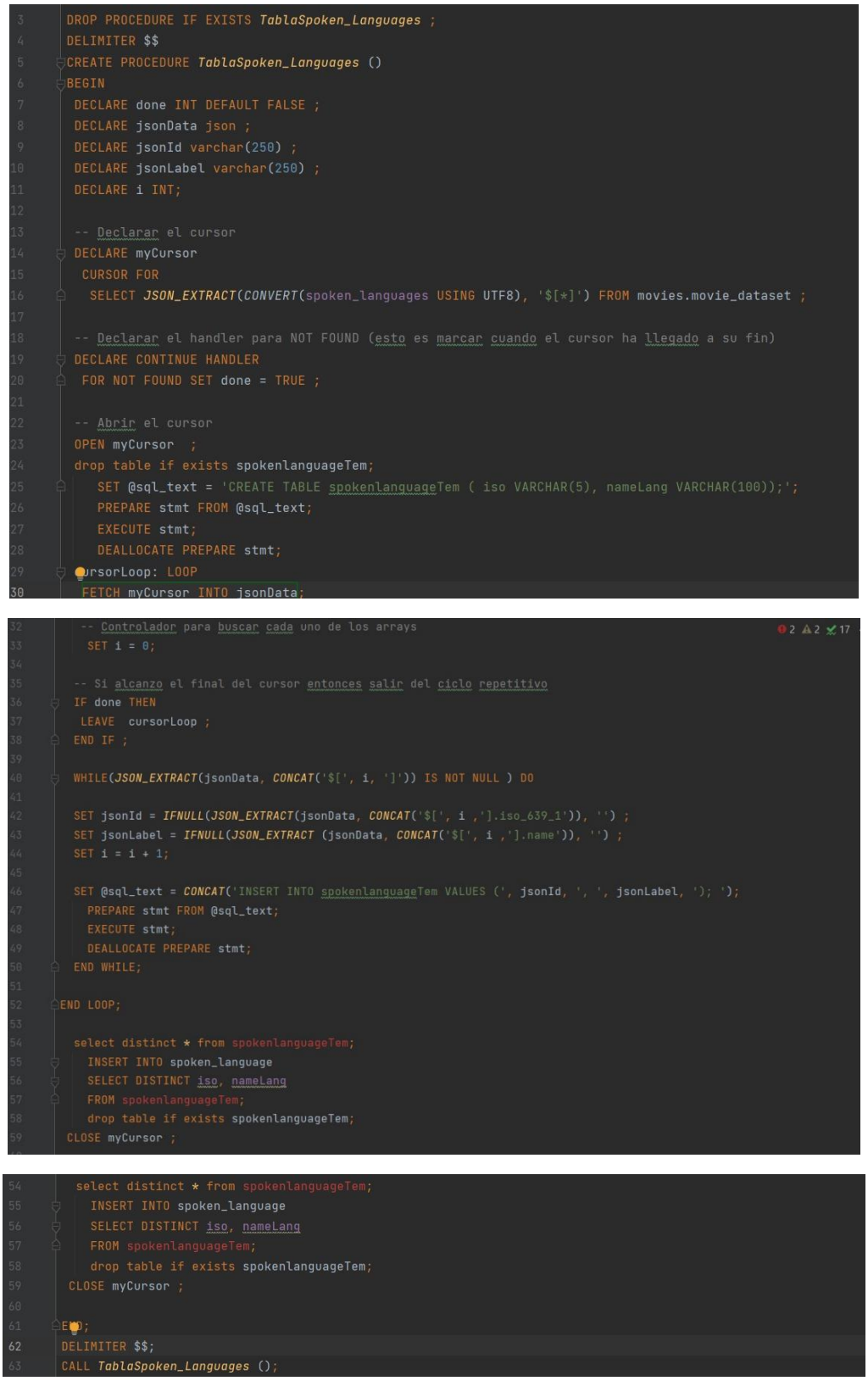
Figura 19. Cursor "TablaProduction_companies"

```sql
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 movies.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_countrieTem ( id VARCHAR(100), nameCom VARCHAR(100));';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
```

```sql
    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_countrieTem VALUES (', REPLACE(jsonId,'\'',''), ', ', jsonLabel, '); ');
   PREPARE stmt FROM @sql_text;
   EXECUTE stmt;
   DEALLOCATE PREPARE stmt;

 END WHILE;

END LOOP ;
```

```sql
 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_countrieTem VALUES (', REPLACE(jsonId,'\'',''), ', ', jsonLabel, '); ');
   PREPARE stmt FROM @sql_text;
   EXECUTE stmt;
   DEALLOCATE PREPARE stmt;

 END WHILE;

END LOOP ;

 select distinct * from production_countrieTem;
   INSERT INTO production_countries
   SELECT DISTINCT id, nameCom
   FROM production_countrieTem;
   drop table if exists production_countrieTem;
 CLOSE myCursor ;

END$$
DELIMITER ;

CALL TablaProduction_countries();
```

Figura 20. Cursor "TablaProduction_countries"



```
DROP PROCEDURE IF EXISTS TablaSpoken_Languages ;
DELIMITER $$
CREATE PROCEDURE TablaSpoken_Languages ()
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 UTF8), '$[*]') FROM movies.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 spokenlanguageTem;
   SET @sql_text = 'CREATE TABLE spokenlanguageTem ( iso VARCHAR(5), nameLang VARCHAR(100));';
   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 spokenlanguageTem VALUES (', jsonId, ', ', jsonLabel, '); ');
  PREPARE stmt FROM @sql_text;
  EXECUTE stmt;
  DEALLOCATE PREPARE stmt;
 END WHILE;

END LOOP;

 select distinct * from spokenlanguageTem;
  INSERT INTO spoken_language
  SELECT DISTINCT iso, nameLang
  FROM spokenlanguageTem;
  drop table if exists spokenlanguageTem;
 CLOSE myCursor ;
```

```
 select distinct * from spokenlanguageTem;
  INSERT INTO spoken_language
  SELECT DISTINCT iso, nameLang
  FROM spokenlanguageTem;
  drop table if exists spokenlanguageTem;
 CLOSE myCursor ;

END;
DELIMITER $$;
CALL TablaSpoken_Languages ();
```

Figura 21. Cursor "TablaSpoken_Languages"

```sql
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 movies.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  ;
```

```sql
 -- 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;
```

```sql
  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();
```

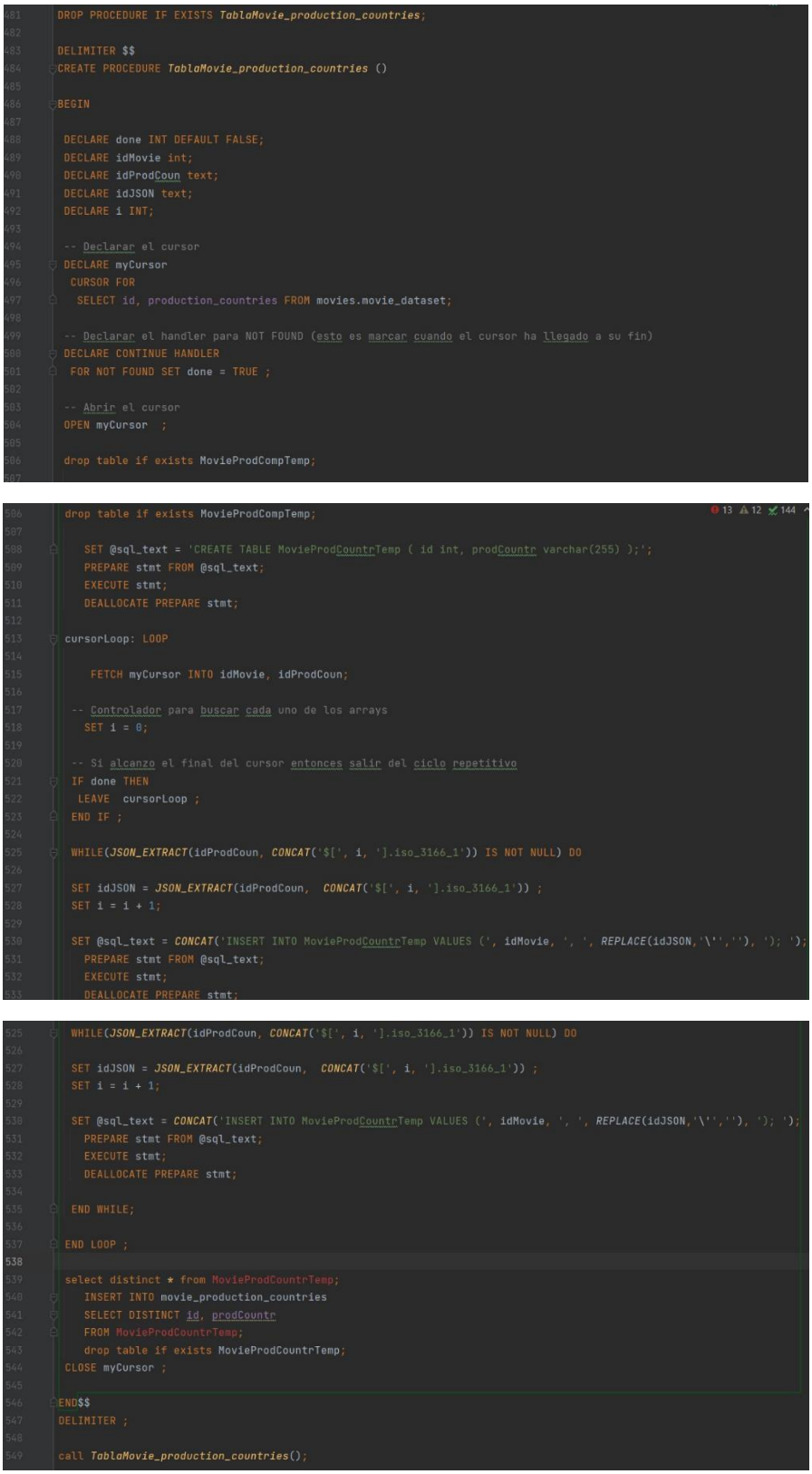
Figura 22. Cursor "TablaMovie_production_companies"

```
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 movies.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;
```

```
 -- 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;
```

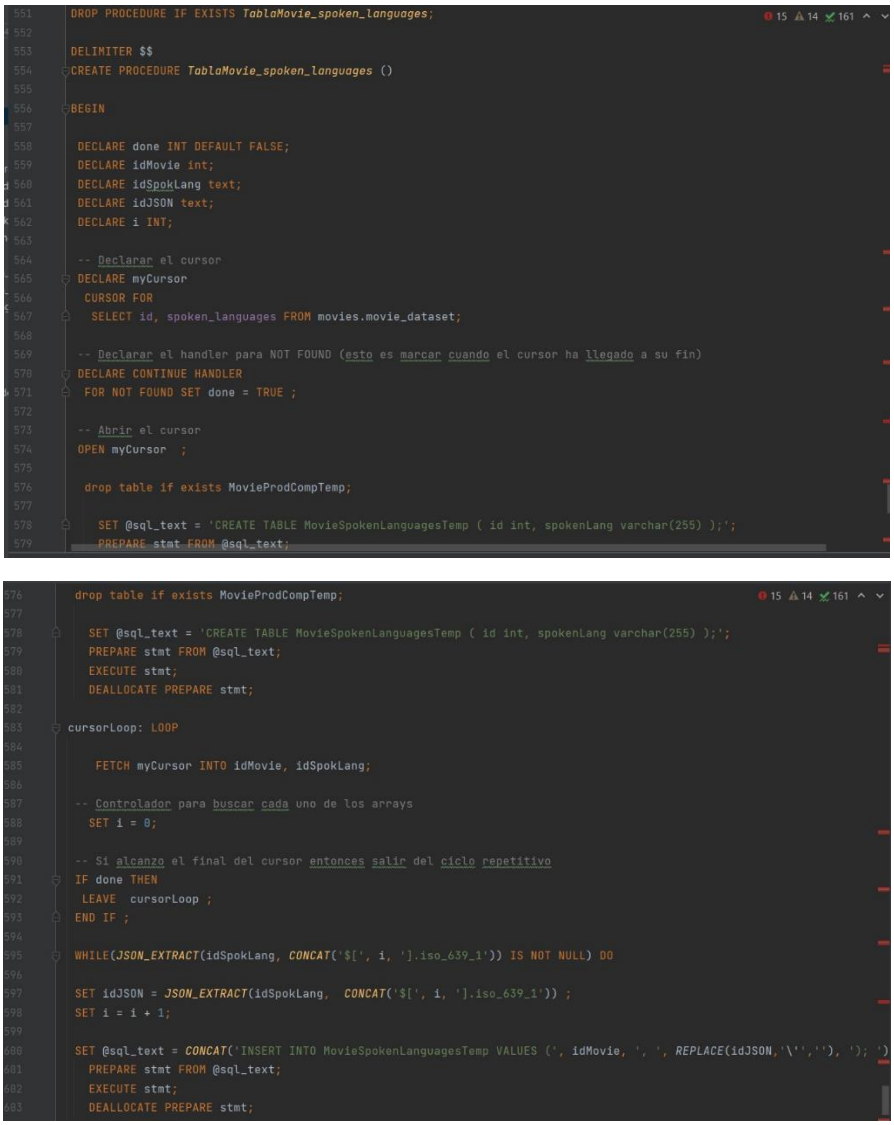
```
  SET @sql_text = CONCAT('INSERT INTO MovieProdCompTemp VALUES (', idMovie, ', ', REPLACE(idJSON,'\'',''), '); ');
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;

  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();
```

Figura 23. Cursor "TablaMovie_production_companies"



```sql
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 movies.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;

 drop table if exists MovieProdCompTemp;

    SET @sql_text = 'CREATE TABLE MovieProdCountrTemp ( id int, prodCountr 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 MovieProdCountrTemp VALUES (', idMovie, ', ', REPLACE(idJSON,'\'',''), '); ');
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;

  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 MovieProdCountrTemp VALUES (', idMovie, ', ', REPLACE(idJSON,'\'',''), '); ');
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;

  END WHILE;

 END LOOP ;

 select distinct * from MovieProdCountrTemp;
    INSERT INTO movie_production_countries
    SELECT DISTINCT id, prodCountr
    FROM MovieProdCountrTemp;
    drop table if exists MovieProdCountrTemp;
 CLOSE myCursor ;

END$$
DELIMITER ;

call TablaMovie_production_countries();
```

Figura 24. Cursor "TablaMovie_production_countries"



```
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 movies.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 MovieSpokenLanguagesTemp ( id int, spokenLang varchar(255) );';
    PREPARE stmt FROM @sql_text;
```

```
  drop table if exists MovieProdCompTemp;

    SET @sql_text = 'CREATE TABLE MovieSpokenLanguagesTemp ( id int, spokenLang varchar(255) );';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;

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 MovieSpokenLanguagesTemp VALUES (', idMovie, ', ', REPLACE(idJSON,'\'',''), '); ')
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    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.