

**UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA**

*La Universidad Católica de Loja*

**Documento de Informe**

**Fundamentos de Bases de Datos**

**Autor**:

* Pinos Armijos Oliver Anthony

Octubre 2022 – Febrero 2023

Contenido

[Introducción 3](#_heading=h.gjdgxs)

[Modelos 4](#_heading=h.30j0zll)

[Modelo Conceptual 4](#_heading=h.1fob9te)

[Modelo Lógico 5](#_heading=h.3znysh7)

[Dependencias Funcionales – Tabla Universal 6](#_heading=h.2et92p0)

[Modelo Físico 7](#_heading=h.tyjcwt)

[Normalización 7](#_heading=h.3dy6vkm)

[First Normal Form (1NF) 7](#_heading=h.1t3h5sf)

[Second Normal Form (2NF) 8](#_heading=h.4d34og8)

[Third Normal Form (3NF) 8](#_heading=h.2s8eyo1)

[Pasos para Realizar el Proyecto 14](#_heading=h.17dp8vu)

[Creación de un “Schema”. 14](#_heading=h.3rdcrjn)

[Conexión del “Schema” a la Base De Datos. 16](#_heading=h.26in1rg)

[Importación del CSV. 16](#_heading=h.lnxbz9)

[Creación de funciones que Permitan Extraer y Limpiar los Datos. 19](#_heading=h.35nkun2)

[LIMPIEZA COLUMNA CREW 32](#_heading=h.44sinio)

[Consultas. 37](#_heading=h.2jxsxqh)

[Conclusiones 39](#_heading=h.z337ya)

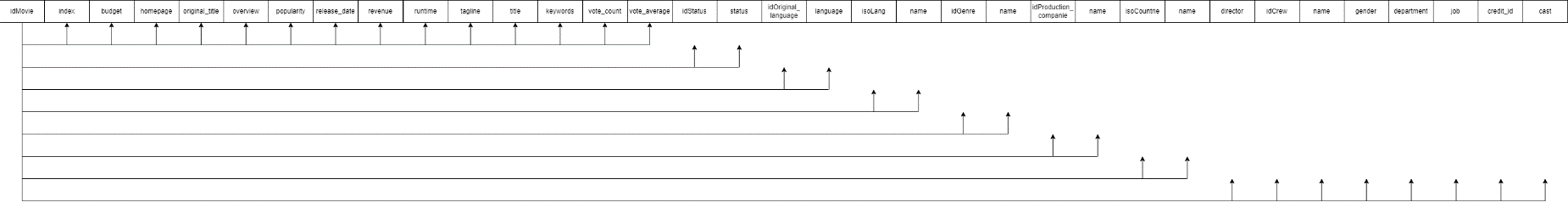
# Introducción

En este proyecto de la materia de Fundamentos de Bases de Datos se nos ha propuesto un problema para proponer una solución, en dicho problema se nos entrega un archivo csv con raw data, nuestra tarea es realizar las distintas fases del sistema ETL para hacer un buen manejo de los datos que contiene dicho csv. En especial, a la hora de hacer la limpieza de los datos, ya que existen muchos retos en torno a esta sección.

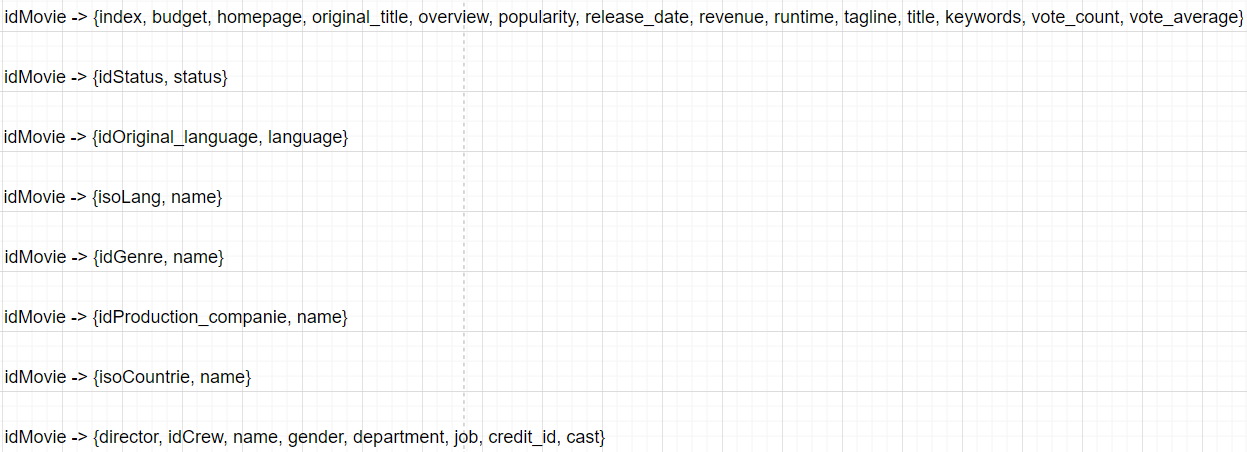
El objetivo de este trabajo es que los estudiantes puedan desempeñarse a la hora de trabajar con raw data, ya que, en su vida laboral, es muy probable que se lleguen a encontrar con estos tipos de problemas entorno a los datos. Por esta razón, es muy importante que los estudiantes realicen el presente proyecto.

## Dependencias Funcionales – Tabla Universal

Lo primero que debemos de hacer, luego de revisar la data que existen dentro del csv, es crear las dependencias funcionales que tendrá nuestra base de datos.

****

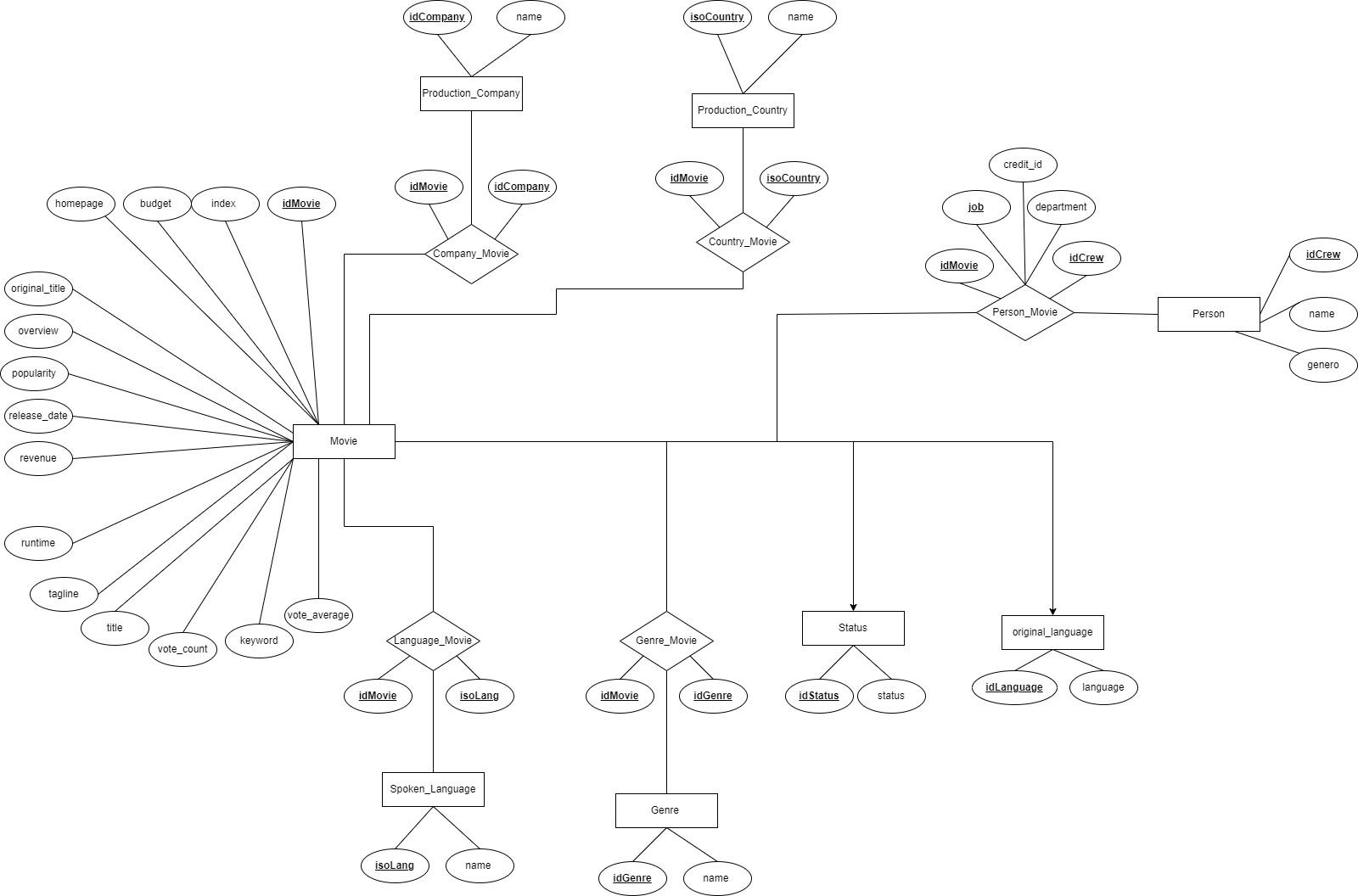
*Figura 1: La imagen muestra una descripción gráfica de las dependencias existentes en las columnas del dataset*

****

*Figura 2: Dependencias Funcionales textualizadas para un mejor entendimiento*

# Modelos

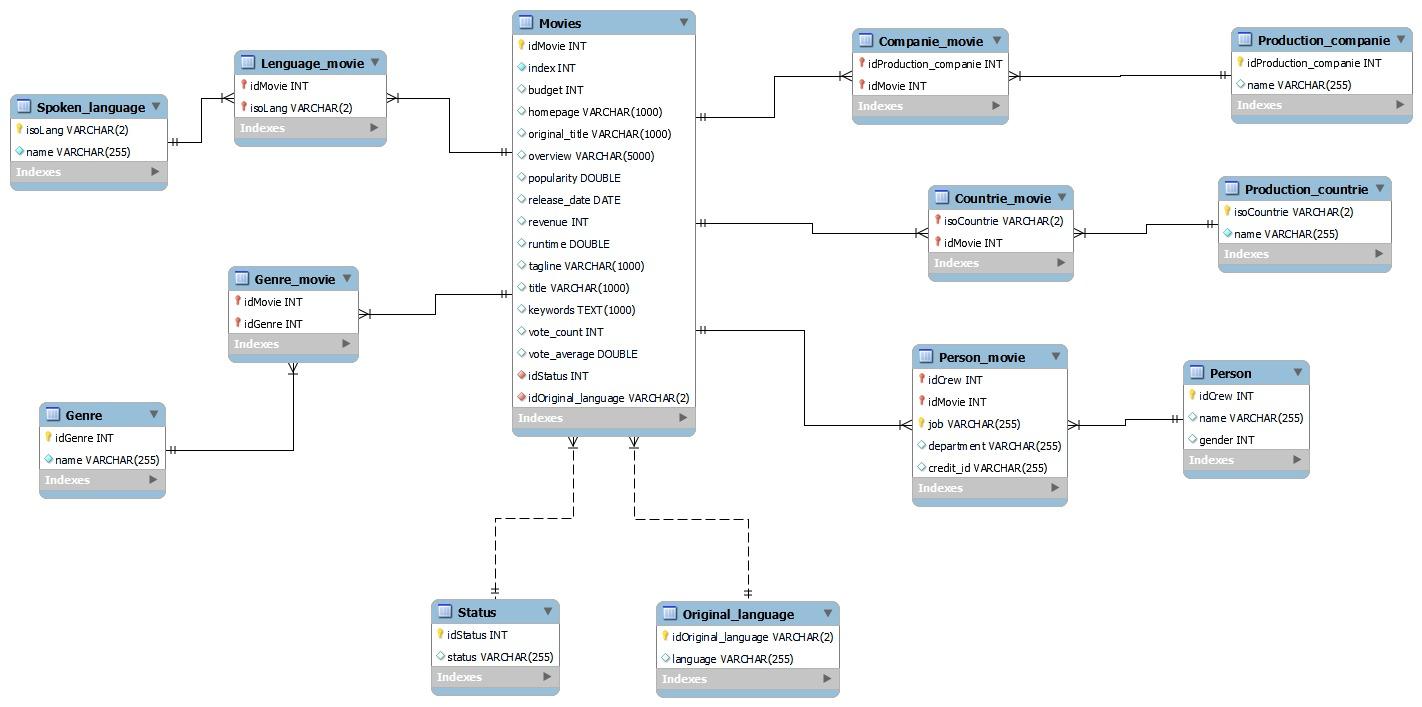
## Modelo Conceptual

Una vez acabadas las dependencias funcionales podemos pasar a realizar el modelo conceptual, este modelo es muy importante ya que sigue la metodología de entidad-relación, además, debemos de tener muy en cuenta los nombres de las tablas y los atributos que las conforman, así como también las relaciones que existen en cada tabla.

*Figura 3: Representación del Modelo Conceptual*

## Modelo Lógico

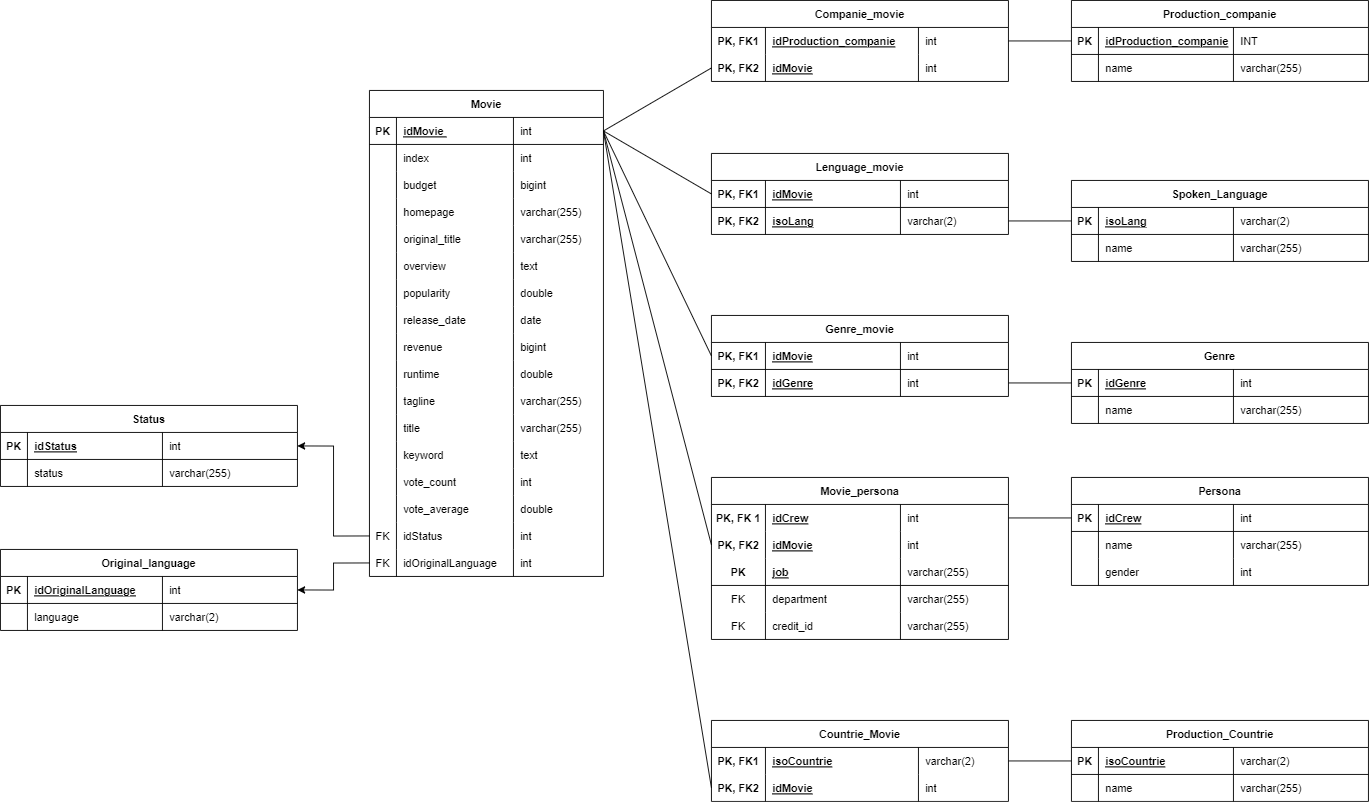
Ahora es turno del modelo lógico, en este modelo ya debemos de tener claro los atributos que contendrán cada tabla de acuerdo al modelo hecho anteriormente. Además de los atributos, debemos de especificar cuales actuarán como llave primaria y/o llave foránea.



*Figura 4: Representación gráfica del Modelo Lógico*

## Modelo Físico

Por último, tenemos al modelo físico, en dicho modelo crearemos un gráfico final el cual nos ayudará a la hora de crear el DDL en nuestra base de datos. Dicho gráfico ya debe de tener las tablas normalizadas, junto con todos sus atributos.



*Figura 5: Representación gráfica del Modelo Físico*

Una vez acabado el modelo físico, podemos pasar a crear las tablas en nuestra base de datos, siguiendo lo hecho anteriormente en los modelos.

-- -----------------------------------------------------  
-- Projecto Base  
-- -----------------------------------------------------  
DROP DATABASE IF EXISTS Projectobase;  
CREATE DATABASE Projectobase CHARACTER SET utf8mb4;  
USE Projectobase;  
  
-- -----------------------------------------------------  
-- Table `Status`  
-- -----------------------------------------------------  
DROP TABLE IF EXISTS `Status` ;  
CREATE TABLE IF NOT EXISTS `Status` (  
 `idStatus` INT NOT NULL AUTO\_INCREMENT,  
 `status` VARCHAR(255) NULL,  
PRIMARY KEY (`idStatus`));  
  
  
-- -----------------------------------------------------  
-- Table `Original\_language`  
-- -----------------------------------------------------  
DROP TABLE IF EXISTS `Original\_language` ;  
CREATE TABLE IF NOT EXISTS `Original\_language` (  
 `idOriginal\_language` INT AUTO\_INCREMENT NOT NULL ,  
 `language` VARCHAR(255) NULL,  
PRIMARY KEY (`idOriginal\_language`));  
  
  
-- -----------------------------------------------------  
-- Table `Movies`  
-- -----------------------------------------------------  
DROP TABLE IF EXISTS `Movies` ;  
CREATE TABLE IF NOT EXISTS `Movies` (  
 `idMovie` INT NOT NULL AUTO\_INCREMENT,  
 `index` INT NOT NULL,  
 `budget` INT NULL,  
 `homepage` VARCHAR(1000) NULL,  
 `original\_title` VARCHAR(1000) NULL,  
 `overview` VARCHAR(5000) NULL,  
 `popularity` DOUBLE NULL,  
 `release\_date` DATE NULL,  
 `revenue` BIGINT NULL,  
 `runtime` DOUBLE NULL,  
 `tagline` VARCHAR(1000) NULL,  
 `title` VARCHAR(1000) NULL,  
 `keywords` TEXT NULL,  
 `vote\_count` INT NULL,  
 `vote\_average` DOUBLE NULL,  
 `idStatus` INT NOT NULL,  
 `idOriginal\_language` int NOT NULL,  
PRIMARY KEY (`idMovie`),  
FOREIGN KEY (`idStatus`) REFERENCES `Status` (`idStatus`),  
FOREIGN KEY (`idOriginal\_language`) REFERENCES `Original\_language` (`idOriginal\_language`));  
  
-- -----------------------------------------------------  
-- Table `Production\_companie`  
-- -----------------------------------------------------  
DROP TABLE IF EXISTS `Production\_companie` ;  
CREATE TABLE IF NOT EXISTS `Production\_companie` (  
 `idProduction\_companie` INT NOT NULL AUTO\_INCREMENT,  
 `name` VARCHAR(255) NULL,  
 PRIMARY KEY (`idProduction\_companie`));  
  
  
-- -----------------------------------------------------  
-- Table `Companie\_movie`  
-- -----------------------------------------------------  
DROP TABLE IF EXISTS `Companie\_movie` ;  
CREATE TABLE IF NOT EXISTS `Companie\_movie` (  
 `idProduction\_companie` INT NOT NULL,  
 `idMovie` INT NOT NULL,  
PRIMARY KEY (`idProduction\_companie`, `idMovie`),  
FOREIGN KEY (`idProduction\_companie`) REFERENCES `Production\_companie` (`idProduction\_companie`),  
FOREIGN KEY (`idMovie`) REFERENCES `Movies` (`idMovie`));  
  
  
-- -----------------------------------------------------  
-- Table `Production\_countrie`  
-- -----------------------------------------------------  
DROP TABLE IF EXISTS `Production\_countrie` ;  
CREATE TABLE IF NOT EXISTS `Production\_countrie` (  
 `isoCountrie` VARCHAR(2) NOT NULL,  
 `name` VARCHAR(255) NOT NULL,  
 PRIMARY KEY (`isoCountrie`));  
  
  
-- -----------------------------------------------------  
-- Table `Countrie\_movie`  
-- -----------------------------------------------------  
DROP TABLE IF EXISTS `Countrie\_movie` ;  
CREATE TABLE IF NOT EXISTS `Countrie\_movie` (  
 `isoCountrie` VARCHAR(2) NOT NULL,  
 `idMovie` INT NOT NULL,  
PRIMARY KEY (`isoCountrie`, `idMovie`),  
FOREIGN KEY (`isoCountrie`)REFERENCES `Production\_countrie` (`isoCountrie`),  
FOREIGN KEY (`idMovie`) REFERENCES `Movies` (`idMovie`));  
  
-- -----------------------------------------------------  
-- Table `Genre`  
-- -----------------------------------------------------  
DROP TABLE IF EXISTS `Genre` ;  
CREATE TABLE IF NOT EXISTS `Genre` (  
 `idGenre` INT NOT NULL AUTO\_INCREMENT,  
 `name` VARCHAR(255) NOT NULL,  
PRIMARY KEY (`idGenre`));  
  
  
-- -----------------------------------------------------  
-- Table `Genre\_movie`  
-- -----------------------------------------------------  
DROP TABLE IF EXISTS `Genre\_movie` ;  
CREATE TABLE IF NOT EXISTS `Genre\_movie` (  
 `idMovie` INT NOT NULL,  
 `idGenre` INT NOT NULL,  
 PRIMARY KEY (`idMovie`, `idGenre`),  
 FOREIGN KEY (`idMovie`)REFERENCES `Movies` (`idMovie`),  
 FOREIGN KEY (`idGenre`) REFERENCES `Genre` (`idGenre`));  
  
-- -----------------------------------------------------  
-- Table `Spoken\_language`  
-- -----------------------------------------------------  
DROP TABLE IF EXISTS `Spoken\_language` ;  
CREATE TABLE IF NOT EXISTS `Spoken\_language` (  
 `isoLang` VARCHAR(2) NOT NULL,  
 `name` VARCHAR(255) NOT NULL,  
 PRIMARY KEY (`isoLang`));  
  
-- -----------------------------------------------------  
-- Table `Lenguage\_movie`  
-- -----------------------------------------------------  
DROP TABLE IF EXISTS `Lenguage\_movie` ;  
CREATE TABLE IF NOT EXISTS `Lenguage\_movie` (  
 `idMovie` INT NOT NULL,  
 `isoLang` VARCHAR(2) NOT NULL,  
PRIMARY KEY (`idMovie`, `isoLang`),  
FOREIGN KEY (`idMovie`) REFERENCES `Movies` (`idMovie`),  
FOREIGN KEY (`isoLang`)REFERENCES `Spoken\_language` (`isoLang`));  
  
-- -----------------------------------------------------  
-- Table `Person`  
-- -----------------------------------------------------  
DROP TABLE IF EXISTS `Person` ;  
CREATE TABLE IF NOT EXISTS `Person` (  
 `idCrew` INT NOT NULL,  
 `name` VARCHAR(255) NULL,  
 `gender` INT NULL,  
PRIMARY KEY (`idCrew`));  
  
  
-- -----------------------------------------------------  
-- Table `Person\_movie`  
-- -----------------------------------------------------  
DROP TABLE IF EXISTS `Person\_movie` ;  
CREATE TABLE IF NOT EXISTS `Person\_movie` (  
 `idCrew` INT NOT NULL,  
 `idMovie` INT NOT NULL,  
 `job` VARCHAR(255) NOT NULL,  
 `department` VARCHAR(255) NULL,  
 `credit\_id` VARCHAR(255) NULL,  
PRIMARY KEY (`idCrew`, `job`, `idMovie`),  
FOREIGN KEY (`idCrew`)REFERENCES `Person` (`idCrew`),  
FOREIGN KEY (`idMovie`)REFERENCES `Movies` (`idMovie`));

# Normalización

## First Normal Form (1NF)

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

## Second Normal Form (2NF)

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

## Third Normal Form (3NF)

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

Primero identificamos y eliminamos grupos repetitivos dentro de la tabla. Un grupo repetitivo es un atributo, o grupo de atributos, dentro de una tabla que ocurre

con múltiples valores para una sola ocurrencia de los atributos clave designados para esa tabla.

Tabla Uno a Muchos (1 : N)

Status es una columna con el nombre de un único estado.

* Una película puede tener un solo estado, pero un estado puede estar en varias películas.
* Solucionamos este tipo de relación haciendo una nueva tabla para la columna Status, donde crearemos una llave primaria de nombre idStatus
* Además, en la tabla Movies se ha colocado idStatus como una clave foránea.

Tabla Muchos a Muchos (N : N)

En nuestro caso, ninguna de las 4803 entradas se repite, cada película es diferente (única).

Cada Movie tiene un index y un id diferente. Como llave primaria utilizamos id el cual nombramos idMovie para no confundirlo con otros id que existan en otras columnas.

* Genres contiene un String de géneros que puede contener 0 a n géneros.
* Para solucionar, cada columna debe contener un solo valor.
* En este caso tenemos una lista de géneros. Existen múltiples valores.
* La solución es crear una tabla separada que se llame Genres.
* La llave primaria es genreName
* Una Movie puede tener muchos géneros, y un género puede estar en muchas Movie.
* La relación (muchos a muchos) se soluciona con una tabla llamada Movie\_Genres utilizando la llave primaria de cada uno.
* Production\_companies contiene un String de JSON que contiene un name y un id que puede contener 0 a n production\_companies.
* Para solucionar, cada columna debe contener un solo valor.
* En este caso tenemos una lista de production\_companies. Existen múltiples valores.
* La solución es crear una tabla separada que se llame Production\_companies.
* Tiene dos atributos, name e id los cuales los nombramos prodCompName y prodCompId el cual es la primary key.
* Una Movie puede tener muchos production\_companies, y un production\_companies puede aparecer en muchas Movie.
* La relación (muchos a muchos) se soluciona con una tabla llamada Movie\_ Production\_companies utilizando la llave primaria de cada uno.
* Production\_countries contiene un String de JSON que contiene un iso\_3166\_1 y un name que puede contener 0 a n production\_countries.
* Para solucionar, cada columna debe contener un solo valor.
* En este caso tenemos una lista de production\_countries. Existen múltiples valores.
* La solución es crear una tabla separada que se llame Production\_countries.
* Tiene dos atributos, iso\_3166\_1 y name los cuales los nombramos, iso\_3166\_1 el cual es la primary key y prodCompName.
* Una Movie puede tener muchos production\_countries, y un production\_countries puede aparecer en muchas Movie.
* La relación (muchos a muchos) se soluciona con una tabla llamada Movie\_ Production\_countries utilizando la llave primaria de cada uno.
* spoken\_languages contiene un String de JSON que contiene un iso\_639\_1 y un name que puede contener 0 a n spoken\_languages.
* Para solucionar, cada columna debe contener un solo valor.
* En este caso tenemos una lista de spoken\_languages. Existen múltiples valores.
* La solución es crear una tabla separada que se llame spoken\_languages.
* Tiene dos atributos, iso\_639\_1 y name los cuales los nombramos, iso\_639\_1 el cual es la primary key y spokLangName.
* Una Movie puede tener muchos spoken\_languages, y un spoken\_languages puede aparecer en muchas Movie.
* La relación (muchos a muchos) se soluciona con una tabla llamada Movie\_ Spoken\_languages utilizando la llave primaria de cada uno.

Además, en la relación muchos a muchas se crea una tabla adicional, en dicha tabla se ubicarán las dos claves primarias de las tablas que tienen dicha relación, por ejemplo, entre la tabla Movie y Production\_companies existe una relación muchos a muchos, entonces, se crea una tabla que actué como intermediario entre estas dos tablas, los atributos que tendrá la tabla adicional son la primary key tanto de Movie como Production\_companies7

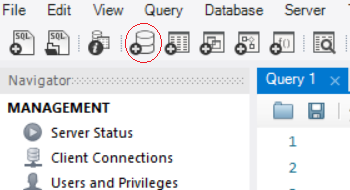
# Pasos para Realizar el Proyecto

## Creación de un “Schema”.

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

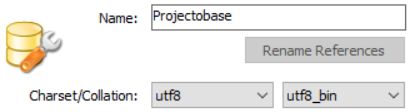
**OPCIÓN 1 – MEDIANTE LA CREACIÓN AUTOMÁTICA**

Para la creación automática del Schema dentro del lenguaje de Base de Datos MySql, primero se debe entrar se debe ubicar en la parte superior izquierda, donde se encuentra los íconos, señalar la que es para crear un Schema como se puede ver en el siguiente Anexo:

**

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

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

**

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

**OPCIÓN 2 – MEDIANTE SENTENCIAS SQL**

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



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

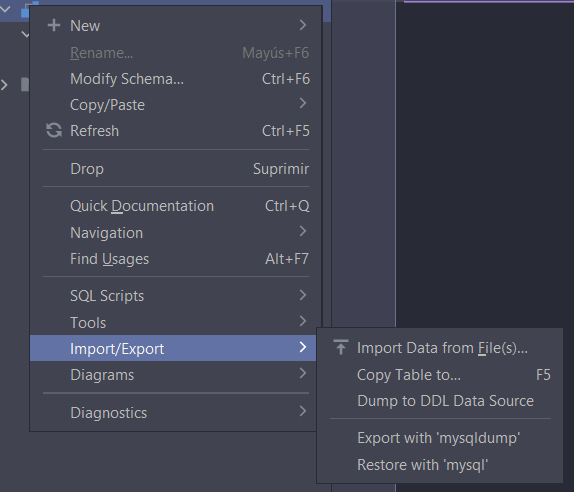
## Conexión del “Schema” a la Base De Datos.

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



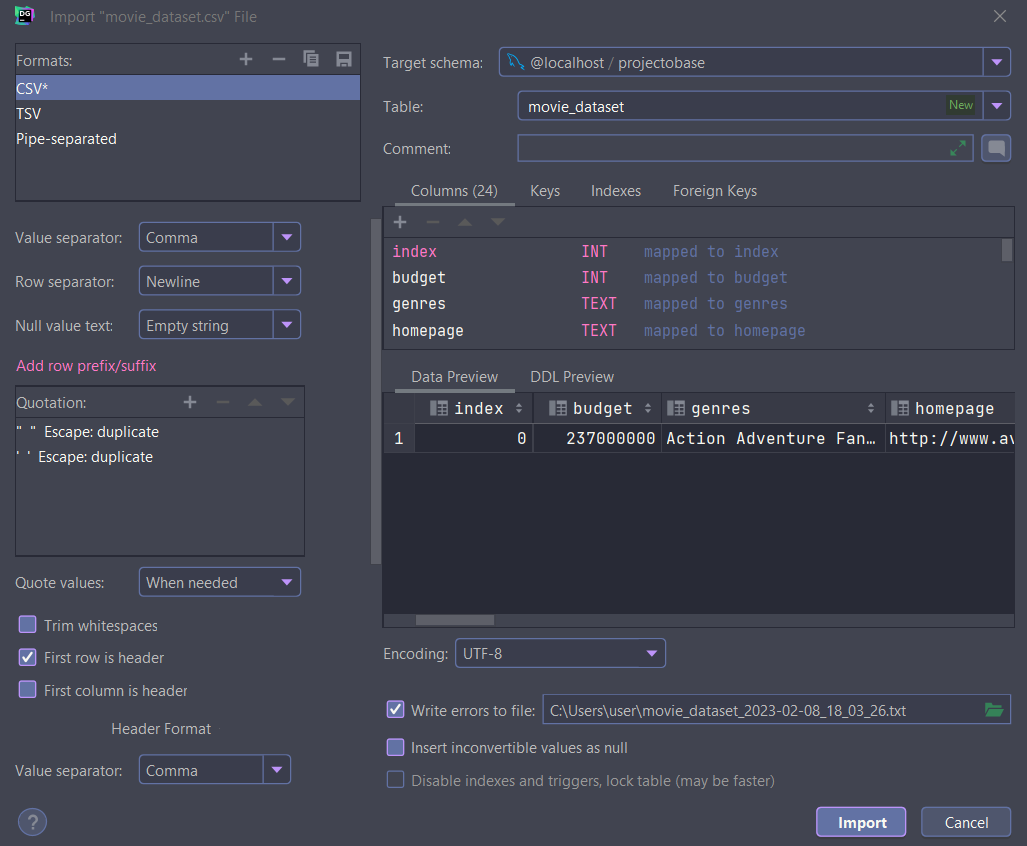
## Importación del CSV.

DataGrip es un IDE de JetBrains para el manejo de base de datos. Dentro de este existe una herramienta facilitadora de importación de varios tipos de archivos a tablas MySQL. A continuación, se muestran los pasos seguidos en la siguiente imagen:



*Figura 8: Representación de la funcionalidad de importación de archivos en DataGrip*

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



*Figura 8: Representación de la interfaz de importación de archivos en DataGrip*

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

Descripción de las tablas Status y original\_langauge

(Tablas uno a muchos)

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

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

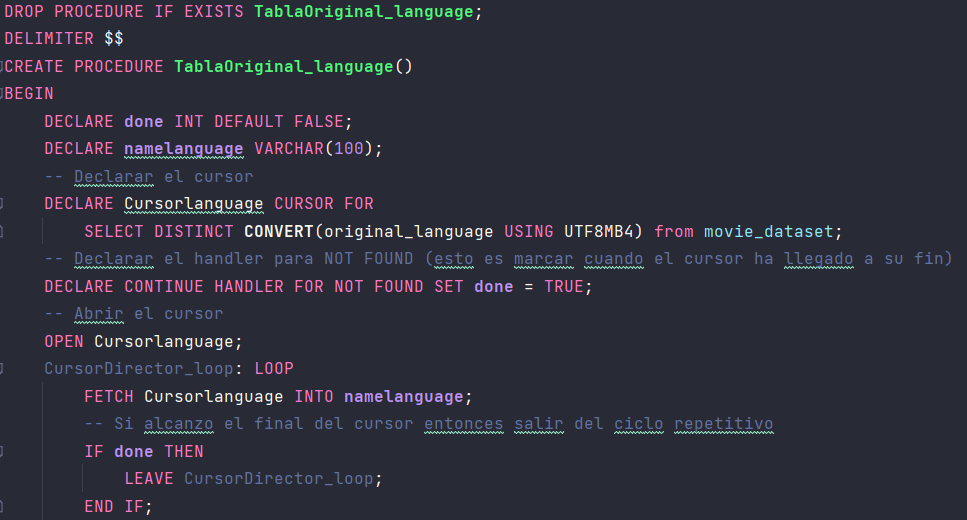
En cada iteración del ciclo, se asigna el nombre del director a la variable "namelanguage". Si se ha alcanzado el final del cursor, se sale del ciclo. Si el nombre de director es nulo, se asigna un valor vacío.

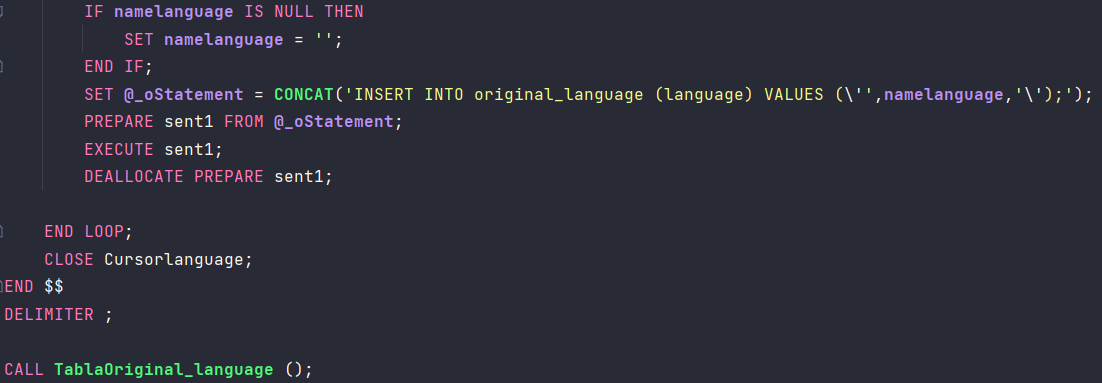
Luego, se crea una consulta dinámica que inserta el nombre del idioma en la tabla "original\_language". Se prepara y ejecuta la consulta dinámica y se libera la memoria de la consulta preparada. Se repite este proceso para cada nombre del idioma que está dentro del cursor.

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

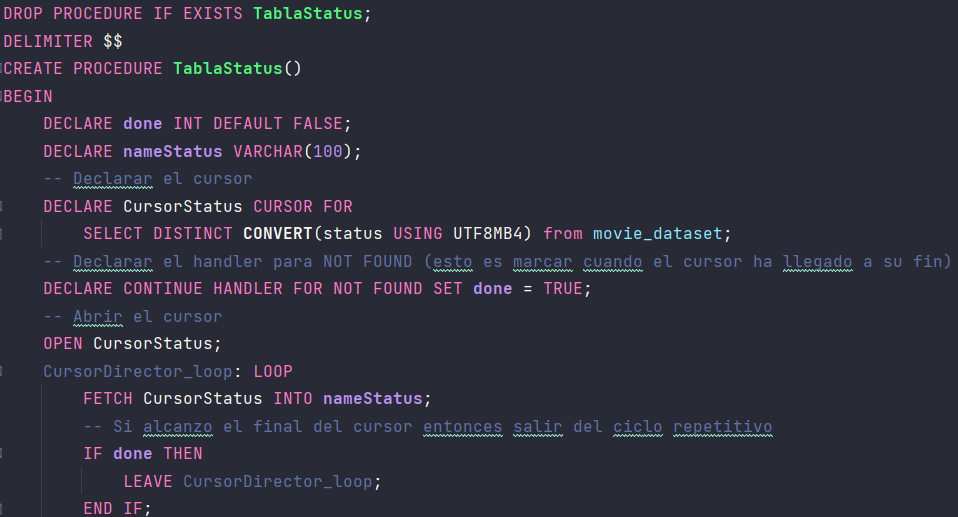
Este código nos servirá para poblar las tablas mencionadas al inicio.

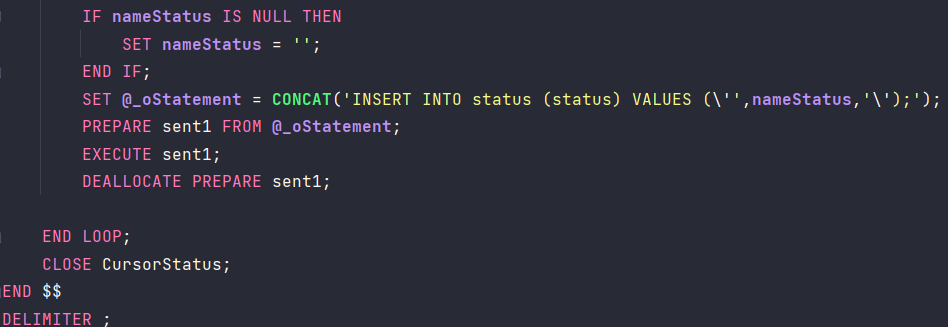
**Población Tabla “Original\_language”**





**Población Tabla “Status”**





Debido a que la tabla Movie utiliza llaves foráneas, era necesario crear las tablas Status y Original\_language para utilizar sus llaves primarias como foráneas en Movie.

**Población Tabla “Movie”**

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

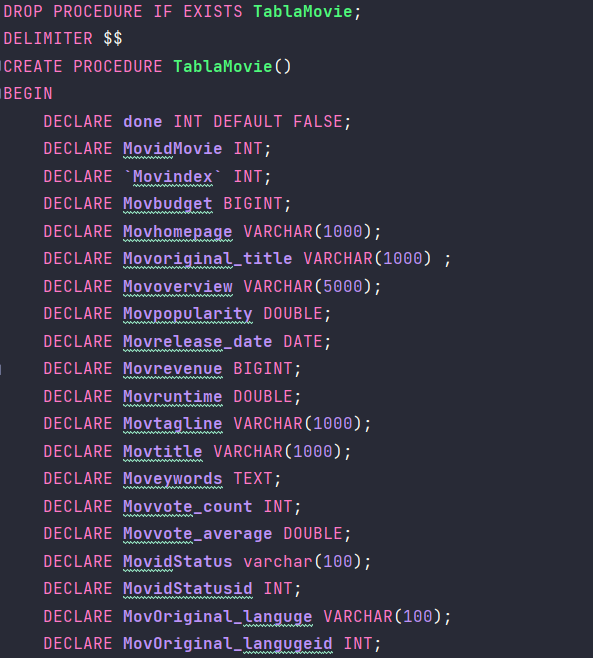
Un procedimiento almacenado primero elimina todos los procedimientos existentes con el mismo nombre. Luego se declaran varias variables para almacenar los valores de cada columna en la tabla "movie\_dataset".

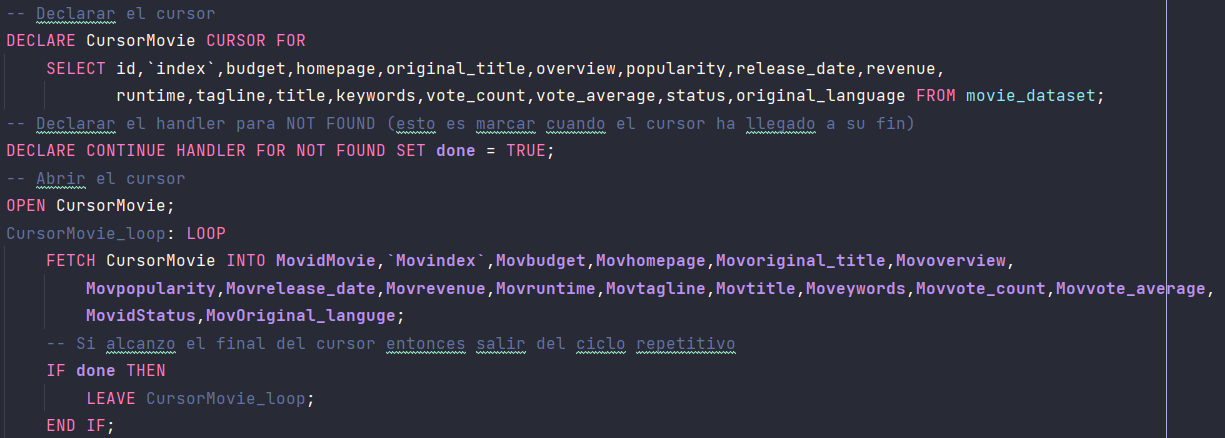
A continuación, se crea un cursor llamado "CursorMovie" que selecciona datos de la tabla "movie\_dataset". Los cursores se utilizan para recorrer cada fila de una tabla y extraer valores de cada columna. Se Declara un "manejador" para manejar el caso cuando se alcance el final del puntero.

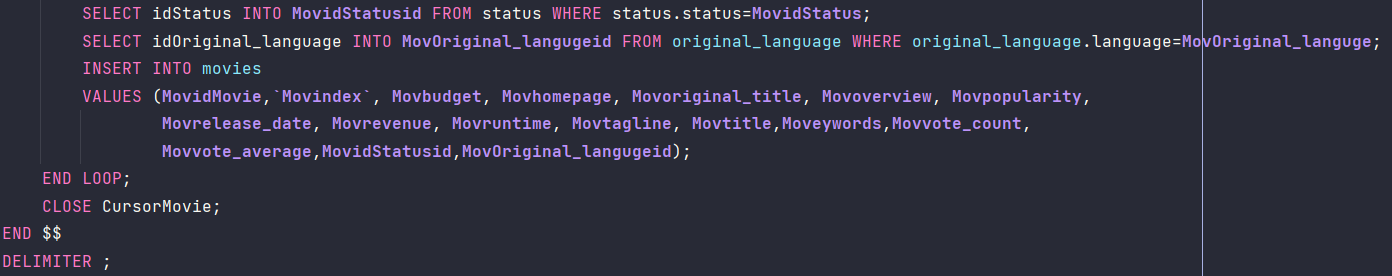
El cursor se abre y "CursorMovie\_loop" comienza a recorrer cada fila de la tabla "movie\_dataset". Dentro del ciclo, la declaración FETCH se usa para recuperar valores de la fila actual y almacenarlos en variables previamente declaradas.

Si se alcanza el final del puntero, el ciclo termina. De lo contrario, comprueba si el nombre del administrador está vacío. Si es así, se le asigna un valor de cero. Ejecuta una consulta para obtener la identificación de las tablas con relación uno a muchos y se guardan en una variable declarada anteriormente.

Finalmente, se inserta una nueva fila en la tabla "movies" con el valor obtenido de la tabla "movie\_dataset" y las claves primarias de las tablas con relación uno a muchos. Cuando se completa el ciclo, el cursor se cierra y el procedimiento almacenado finaliza.

****

****

****

Tenemos varias columnas en formato JSON, para las cuales seguiremos el mismo procedimiento de población.

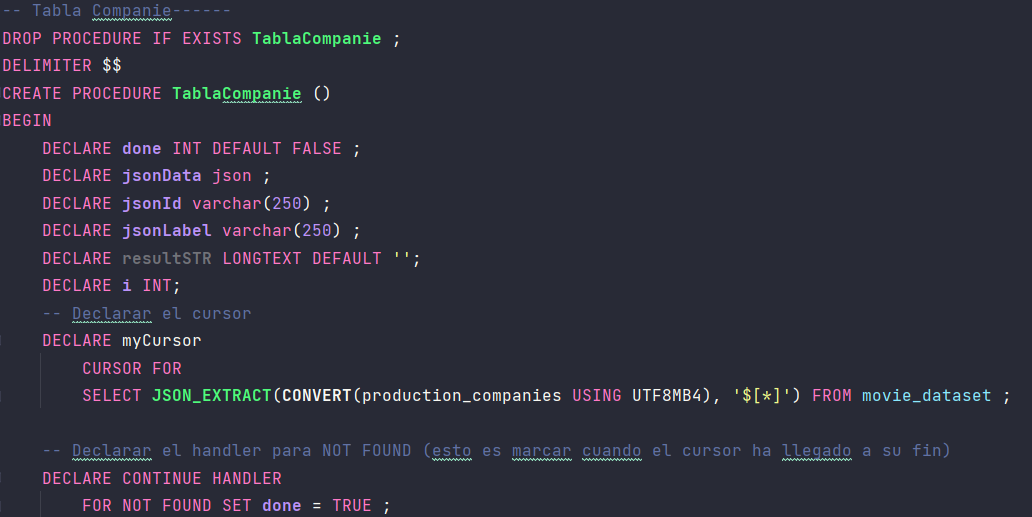
Este código es un procedimiento que se encarga de extraer información de una tabla llamada "movie\_dataset", en particular la columna "production\_companies", que contiene información en formato JSON sobre las compañías productoras de películas.

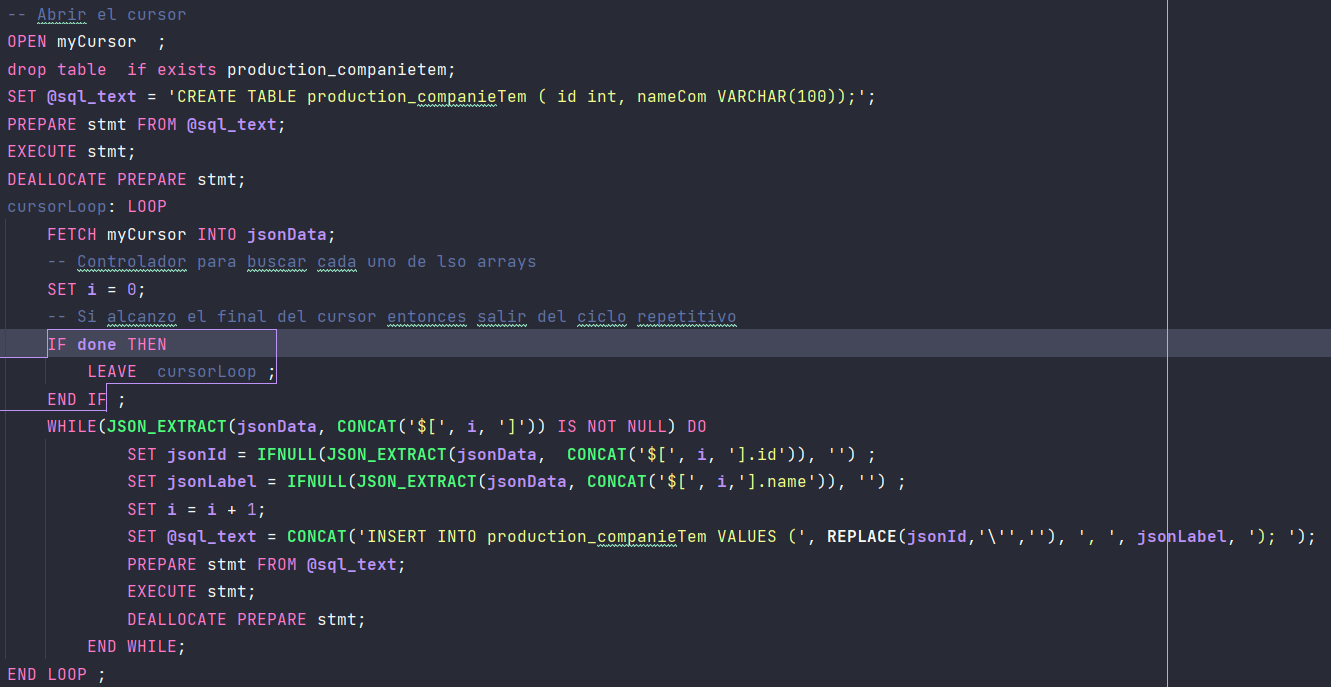
El procedimiento comienza declarando variables, un cursor que se encarga de recorrer las filas de la tabla "movie\_dataset" y un controlador de eventos "CONTINUE HANDLER" para determinar cuándo se ha alcanzado el final de los datos. Luego, se crea una tabla temporal llamada "production\_companieTem"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

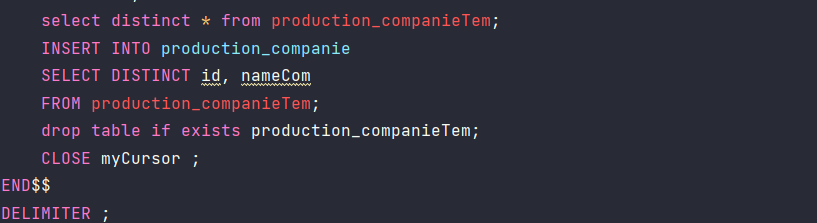
El cursor se usa para recorrer las filas de la tabla "movie\_dataset" y, para cada fila, se extrae la información de las compañías productoras de las películas utilizando la función "JSON\_EXTRACT". Los datos extraídos se insertan en la tabla "production\_companieTem".

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

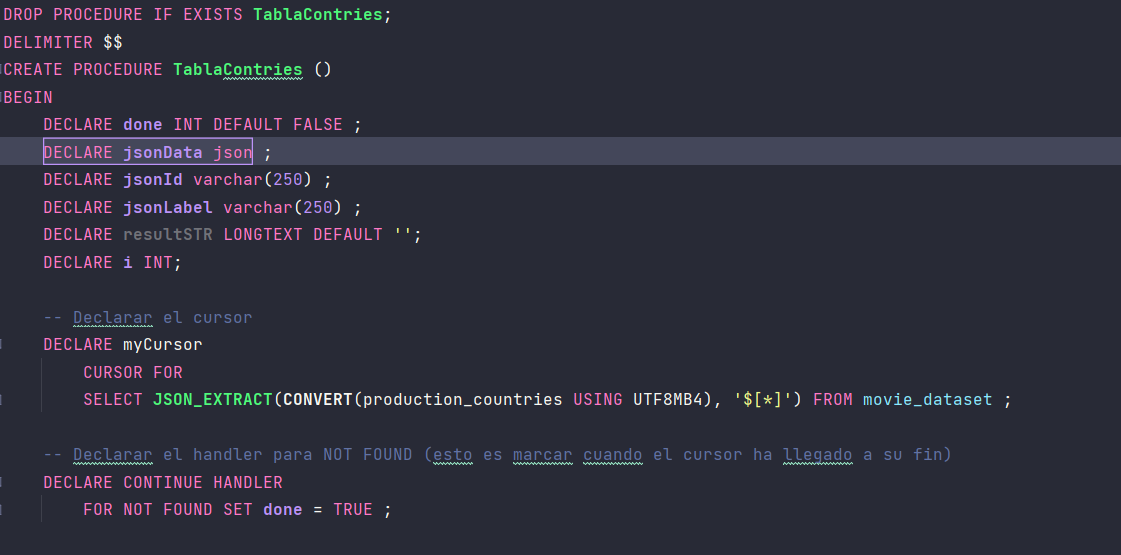
**Población Tabla “Production\_companies”**

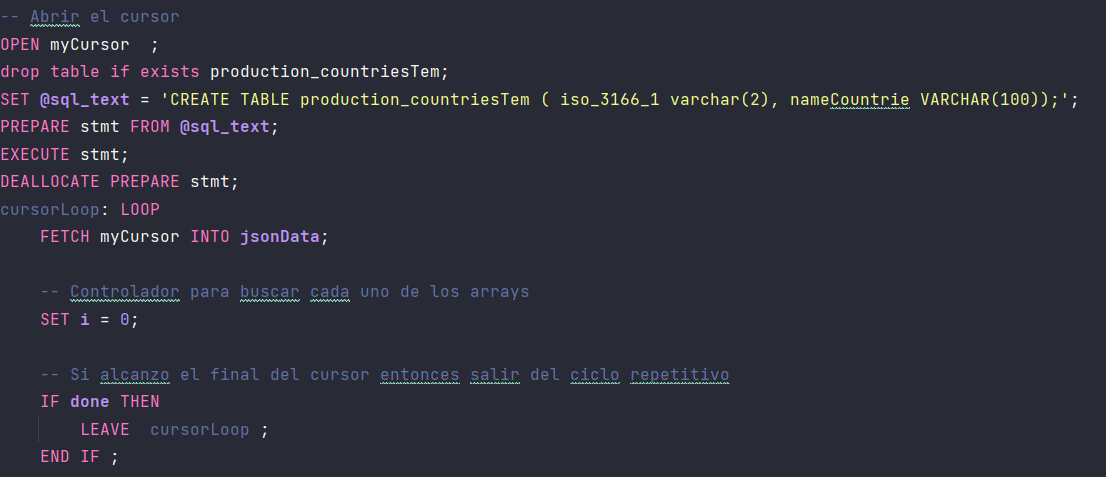


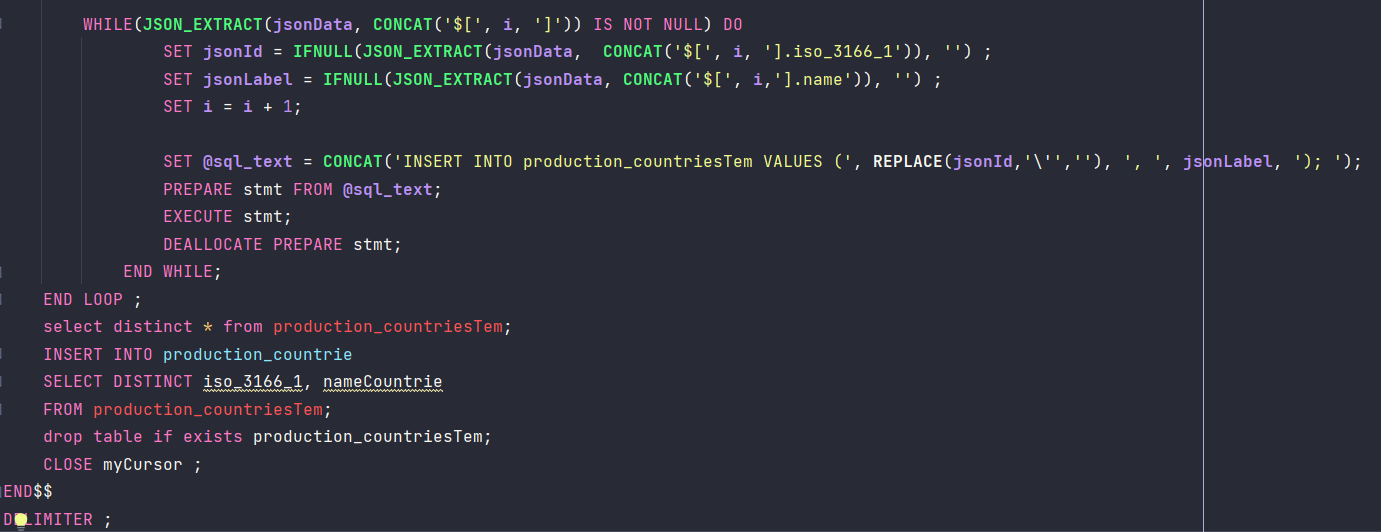




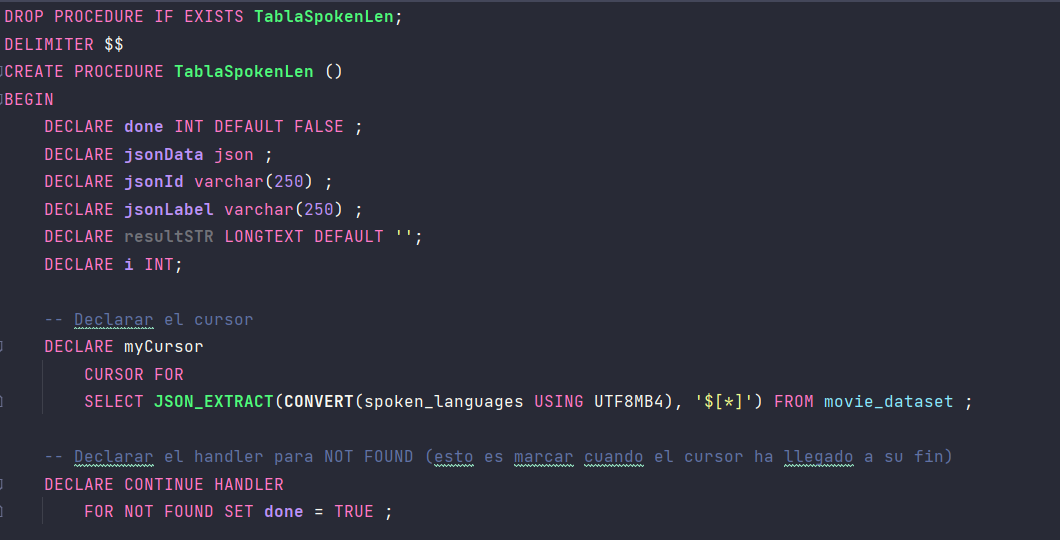
**Población Tabla “Production\_countries”**

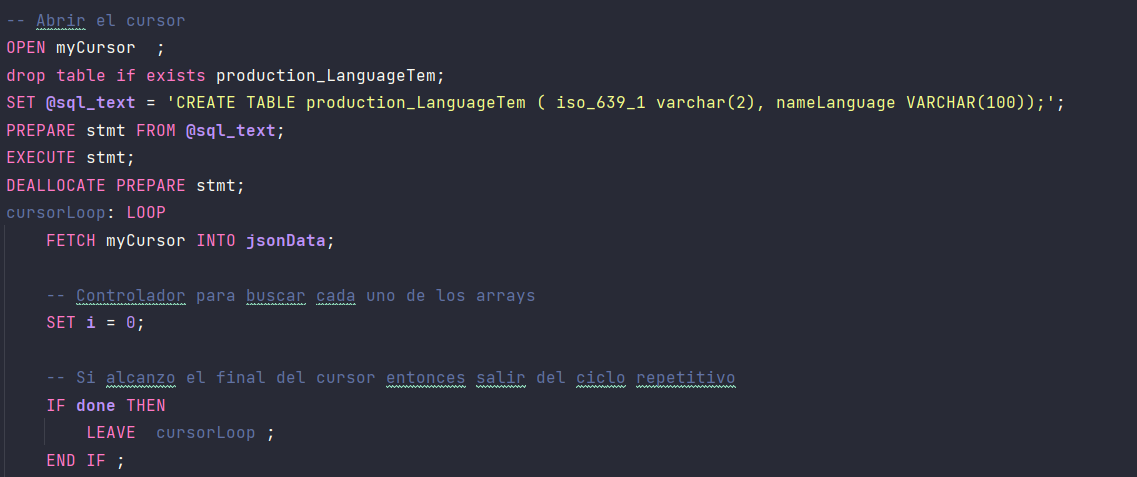


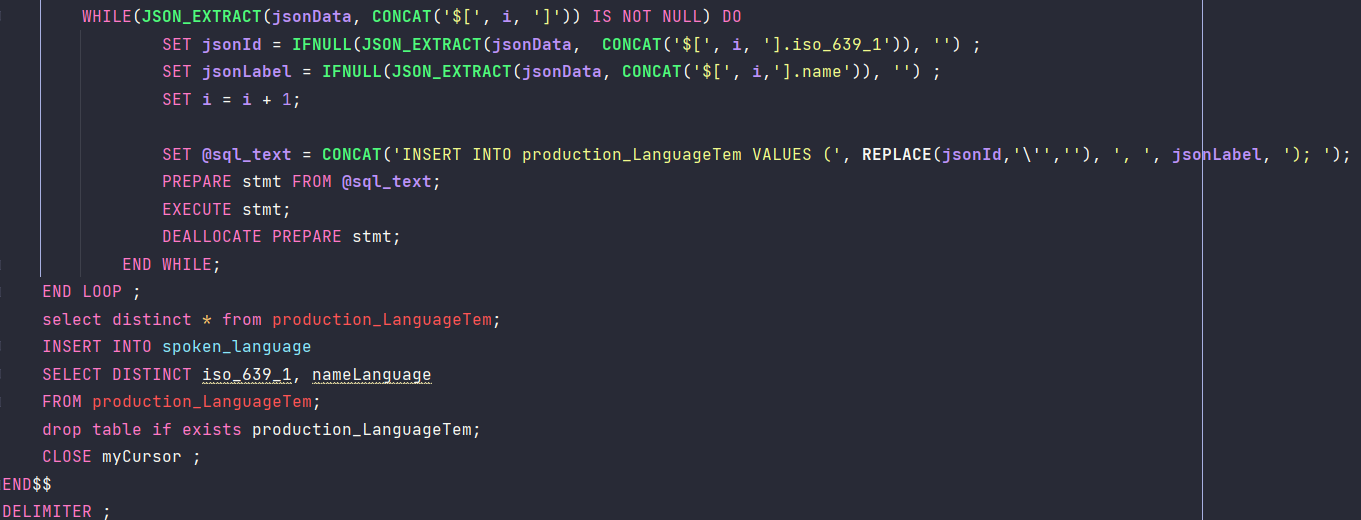




**Población Tabla “Spoken\_languages”**

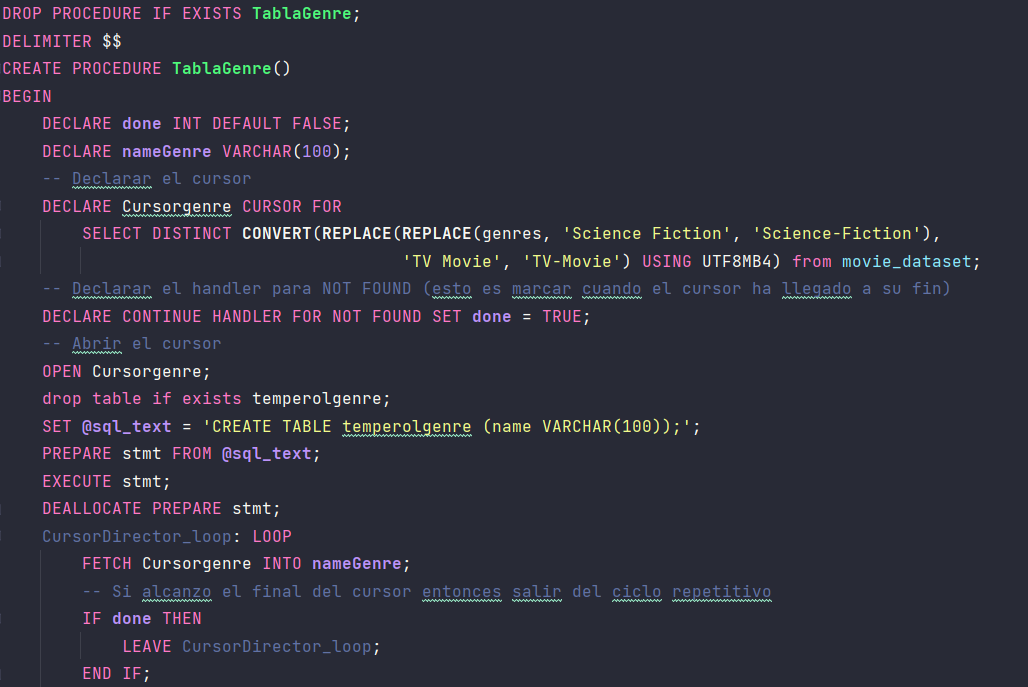






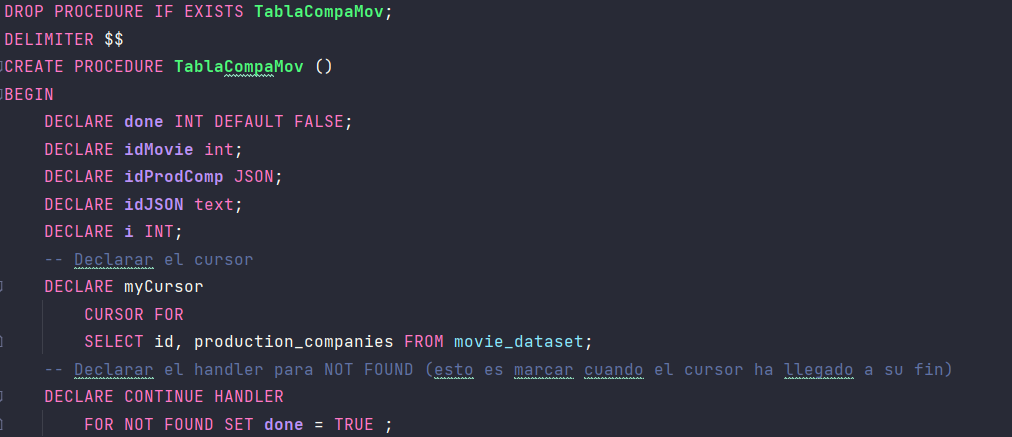
Por último, tenemos la población mediante procedimientos para las tablas con relación muchos a muchos.

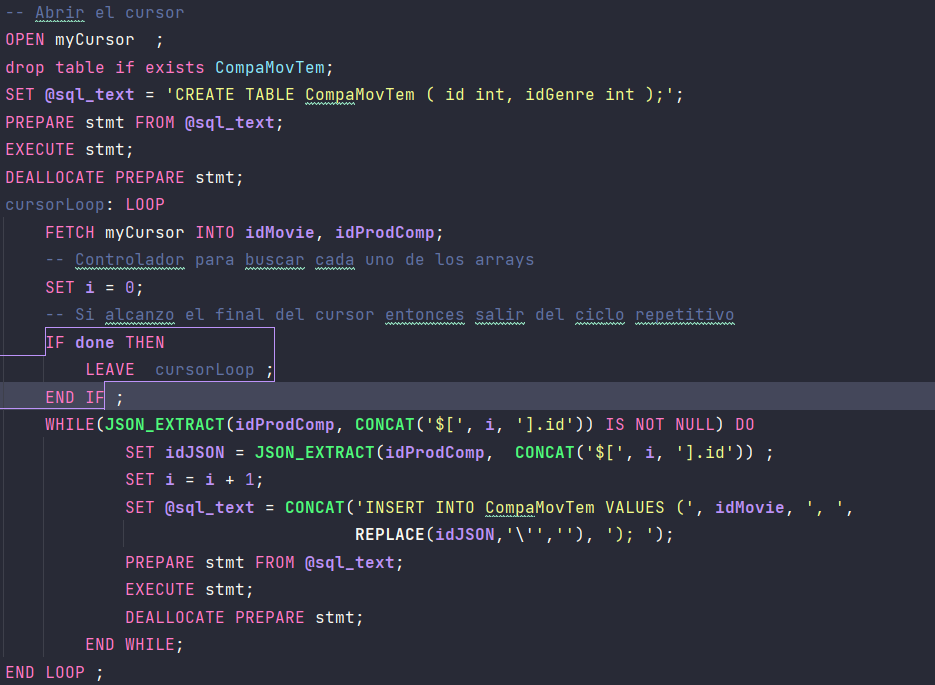
**Población Tabla “Genre”**

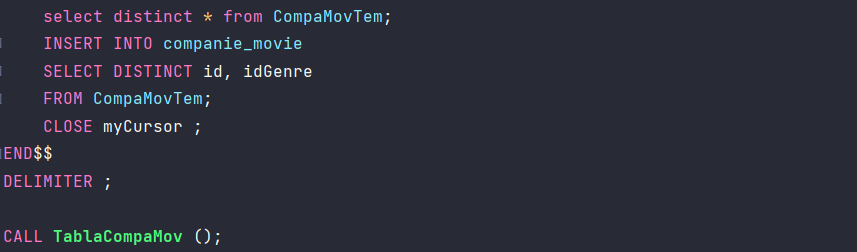
****

****

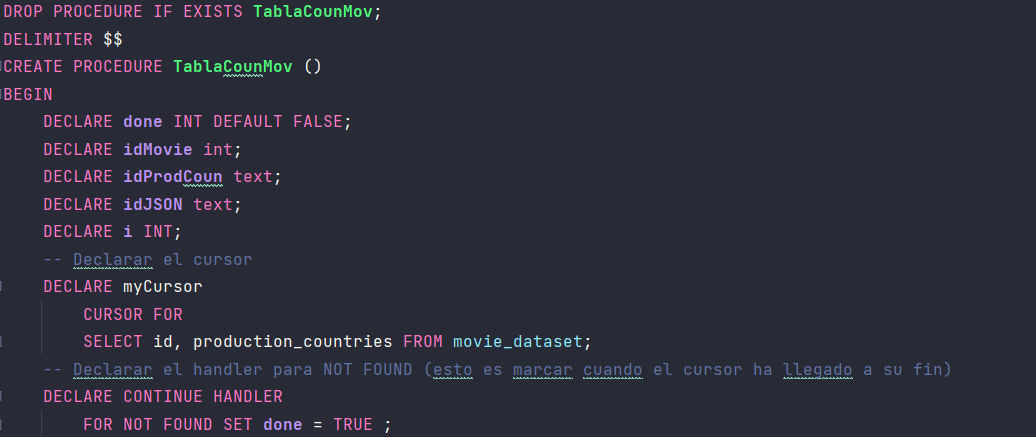
**Población Tabla “Companie\_movie”**

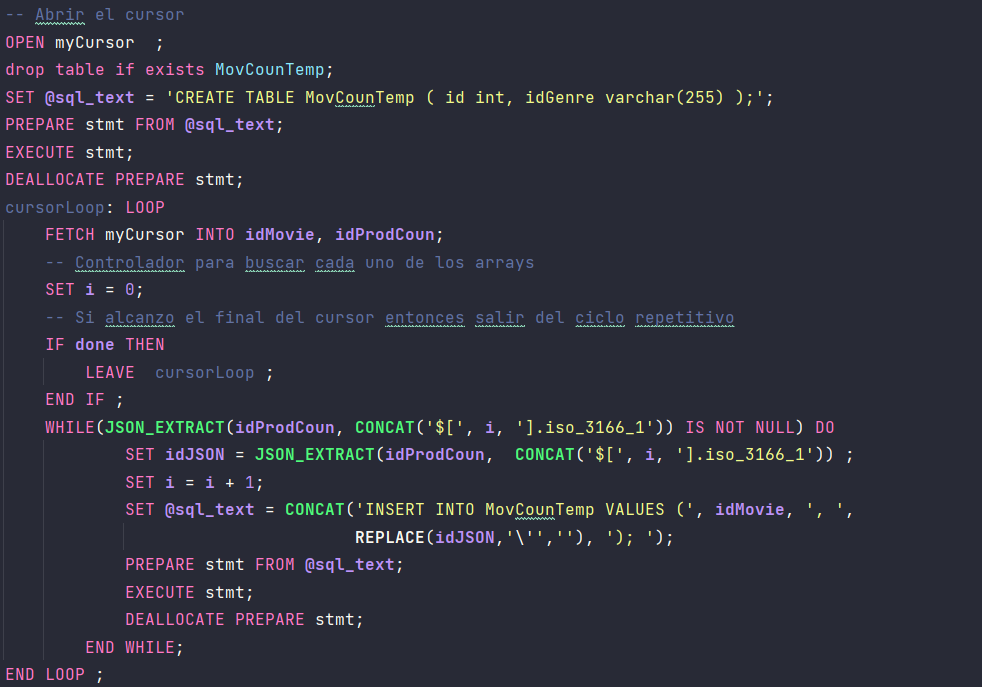
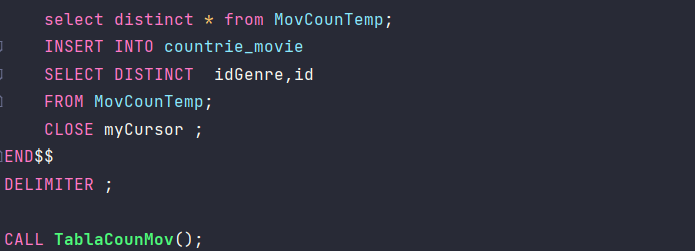
****

****

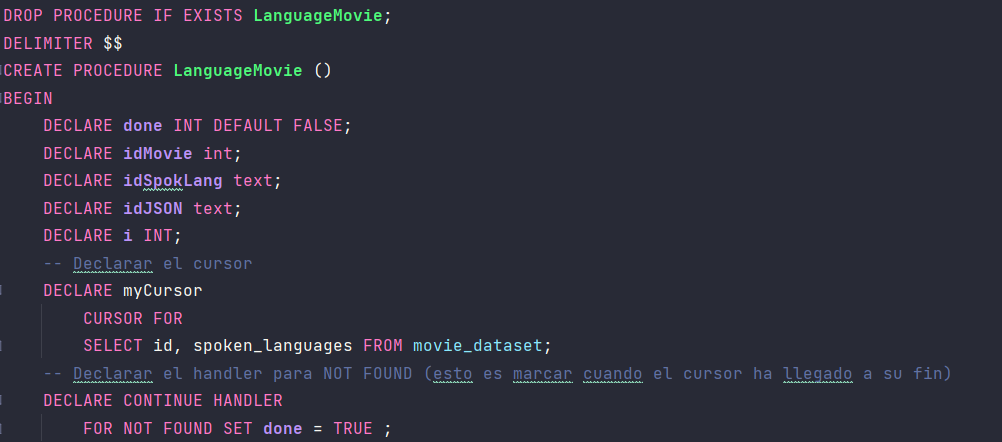
****

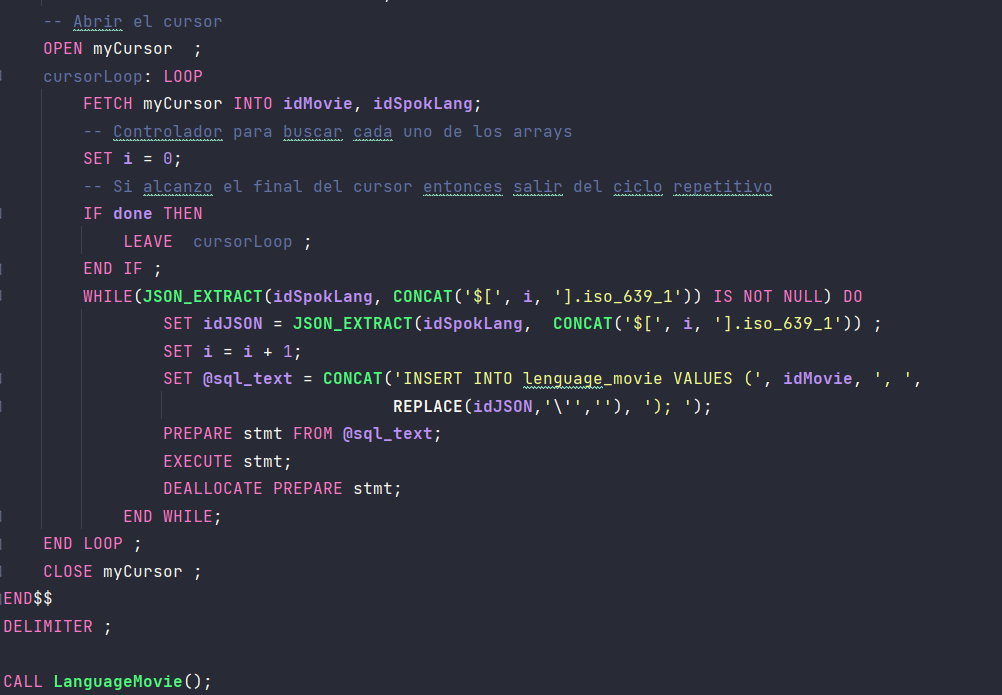
**Población Tabla “Countrie\_movie”**

****

**** ****

**Población Tabla “Lenguage\_movie”**

****



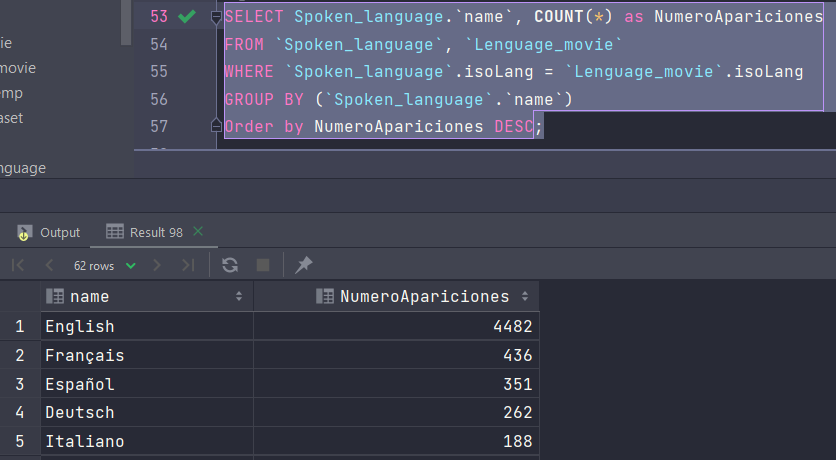
# Limpieza columna crew

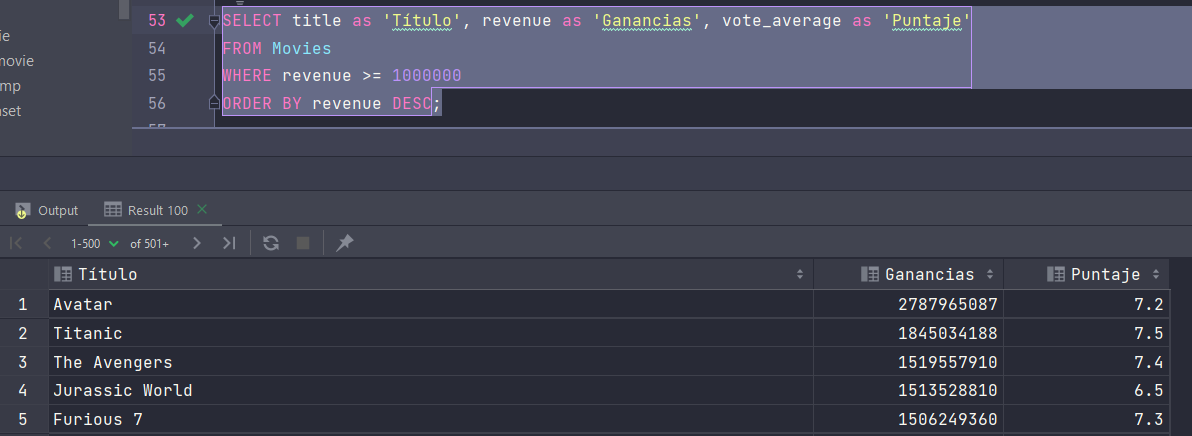
La columna crew es una columna tipo JSON, el problema con esta columna es que tiene un JSON no válido, los key del JSON están delimitados por comillas simples en lugar de comillas dobles, además, dentro de ella se encuentran los nombres de la gente que ha participado en las películas, dichos nombres pueden contener dentro tanto comillas simples como comillas dobles, es por esta razón que la solución de este problema no es hacer un único REPLACE. Para limpiar la columna crew se han usado distintos REPLACE los cuales hay que poner en un orden en específico para que la limpieza se haga de manera exitosa, el primer REPLACE debe de ser convertir comillas dobles a comillas simples, luego de esto reemplazaremos el inicio del JSON, allí se reemplazará la primera comilla simple por una comilla doble, el siguiente REPLACE reemplazará los key del JSON que tengan un dato tipo cadena en su interior, seguido de esto se reemplaza el final de un key que contenga un dato tipo cadena junto con el inicio de otro key del JSON, después se reemplazará el inició de una key con un dato numérico, y por último se hará un REPLACE para el final de un dato tipo numérico que marca el comienzo de otra key del JSON, el código descrito anteriormente se muestra a continuación.

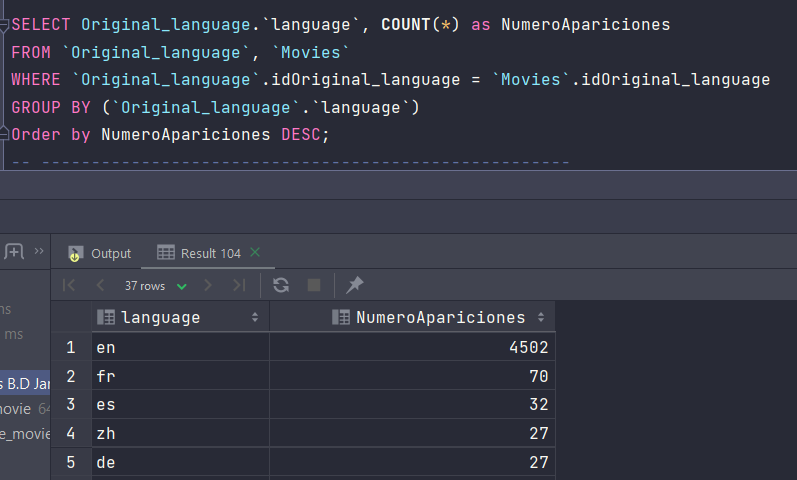


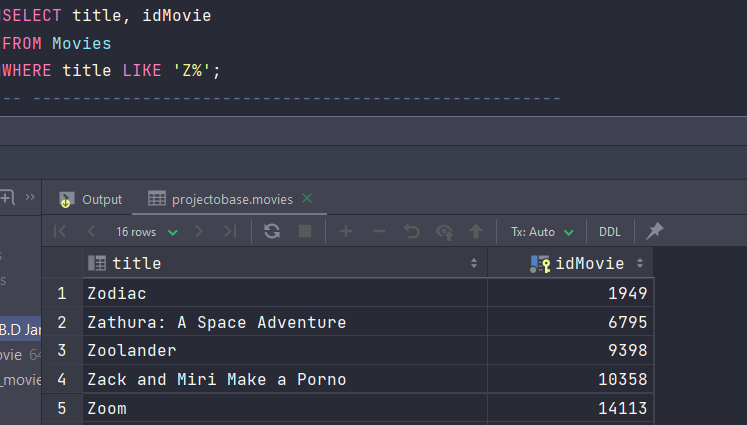
Acerca de la columna cast, no se ha podido resolverla debido a que se han experimentado muchos problemas en dicha columna, entre ellos tenemos que existen muchos nombres distintos y muchas opciones distintas de REPLACE que tenemos que hacer para realizar la limpieza de la columna cast.

# Consultas.









# Conclusiones

Este proyecto fue realizado bajo unas normas de modelado de base de datos, como normalización, diseño conceptual, lógico, etc. Y luego de una ardua tarea por parte de todos los integrantes del grupo, hemos podido cumplir con todos los objetivos planteados al inicio del desarrollo. Cabe resaltar que hemos tenido que afrontar varios desafíos a medida que íbamos progresando en el proyecto, sin embargo, con las pautas de nuestros docentes guías y nuestra creatividad de resolución de problemas, pudimos satisfacer los requerimientos dados.

Una vez realizado todos los pasos correspondientes para culminar nuestro proyecto, hemos podido evidenciar un gran progreso en nuestra toma de decisiones referentes al modelado y población de una base de datos, además de aprender nuevas estrategias de programación aplicables a casos de la vida real. El desarrollo de este proyecto sin duda nos acerca un paso más a ser unos profesionales seguros a la hora de trabajar en proyectos futuros.

Aprendimos que datos, antes de sus respectivos tratamientos, deben de hacer de manera prolija para evitar consistencia y errores. Además, entendemos la importancia de las herramientas al momento de interactuar con el base de datos. Y se puede culminar que todo lo visto este ciclo fue muy importante ya que eso fue lo utilizado en este proyecto integrador cabe recalcar que este proyecto integrador fue un reto para nosotros como estudiantes ya que es nuestra primera vez trabajando con tantos datos.