



UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA
La Universidad Católica de Loja

Informe Preliminar
Fundamentos de Bases de Datos

Autor: Carlos Enrique Morocho Carrión

Octubre 2022 – Febrero 202

Tabla de contenido

1. Introducción	3
2. Normalización.....	4
3. Modelos.....	6
4. Tabla Uno a Muchos (1 : N).....	9
5. Tabla Muchos a Muchos (N : N)	10
6. Pasos para Realizar el Proyecto	12
7. LIMPIEZA COLUMNA CREW	34
8. Consultas.	36
9. Conclusiones.....	39

1. Introducción

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

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

2. Normalización

First Normal Form (1NF)

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

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

Ejemplo:

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

La segunda Forma Normal (2FN)

Se debe seguir los siguientes pasos:

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

Ejemplo:

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

La tercera Forma Normal (3FN)

Se debe considerar los siguientes puntos:

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

Ejemplo:

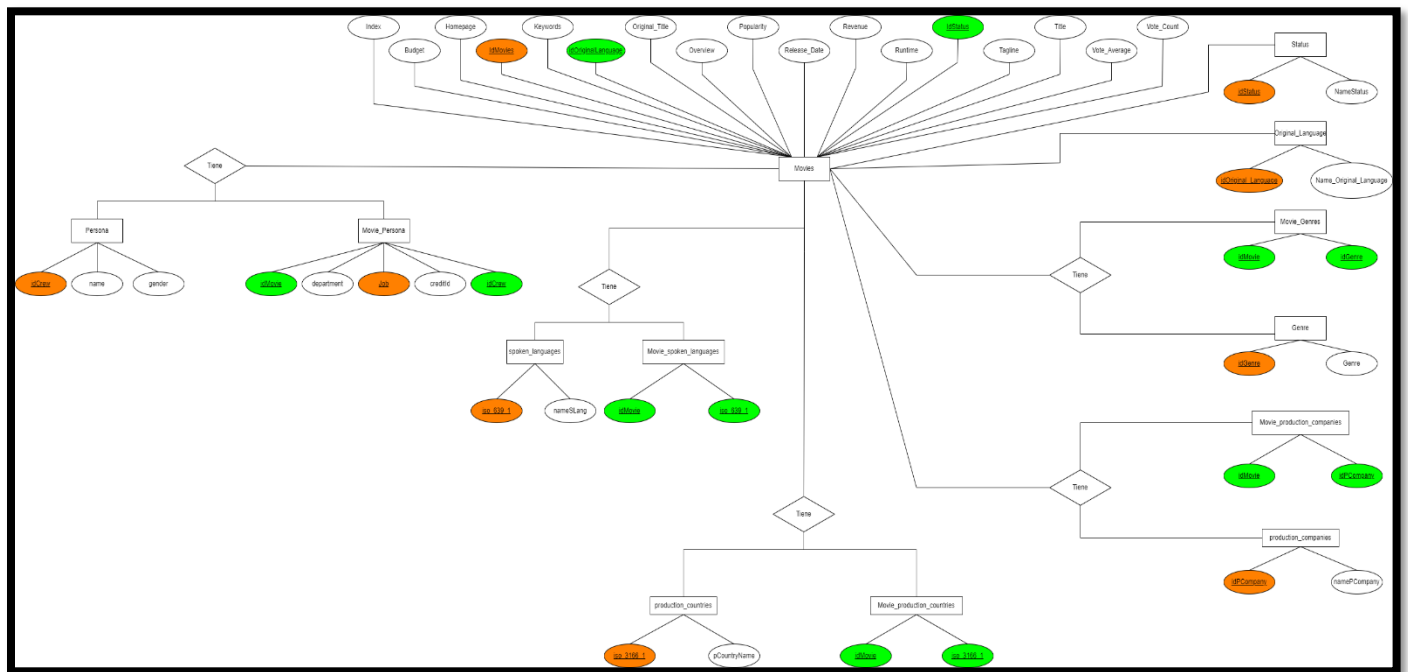
El más claro ejemplo para esta tercera forma normal es la tabla universal “Movies”, ya que es la única tabla que no depende de otras tablas.

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

3. Modelos

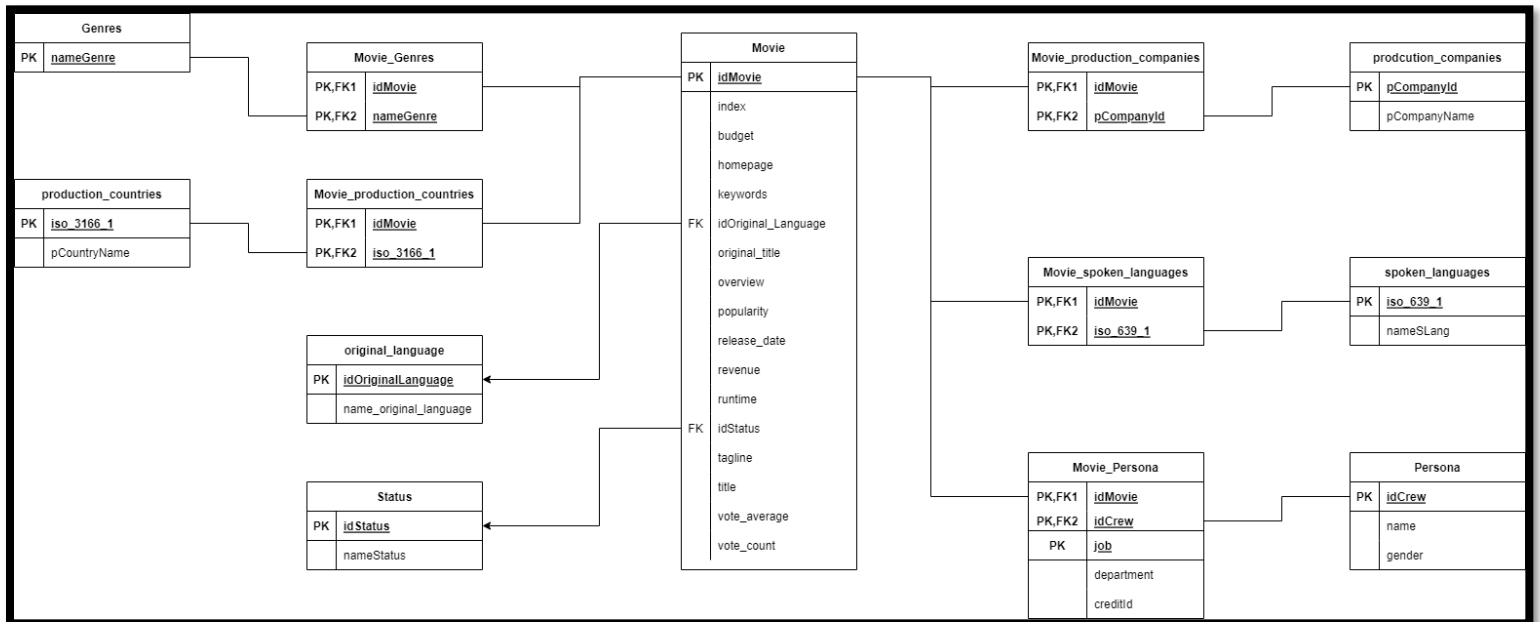
Modelo Conceptual

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



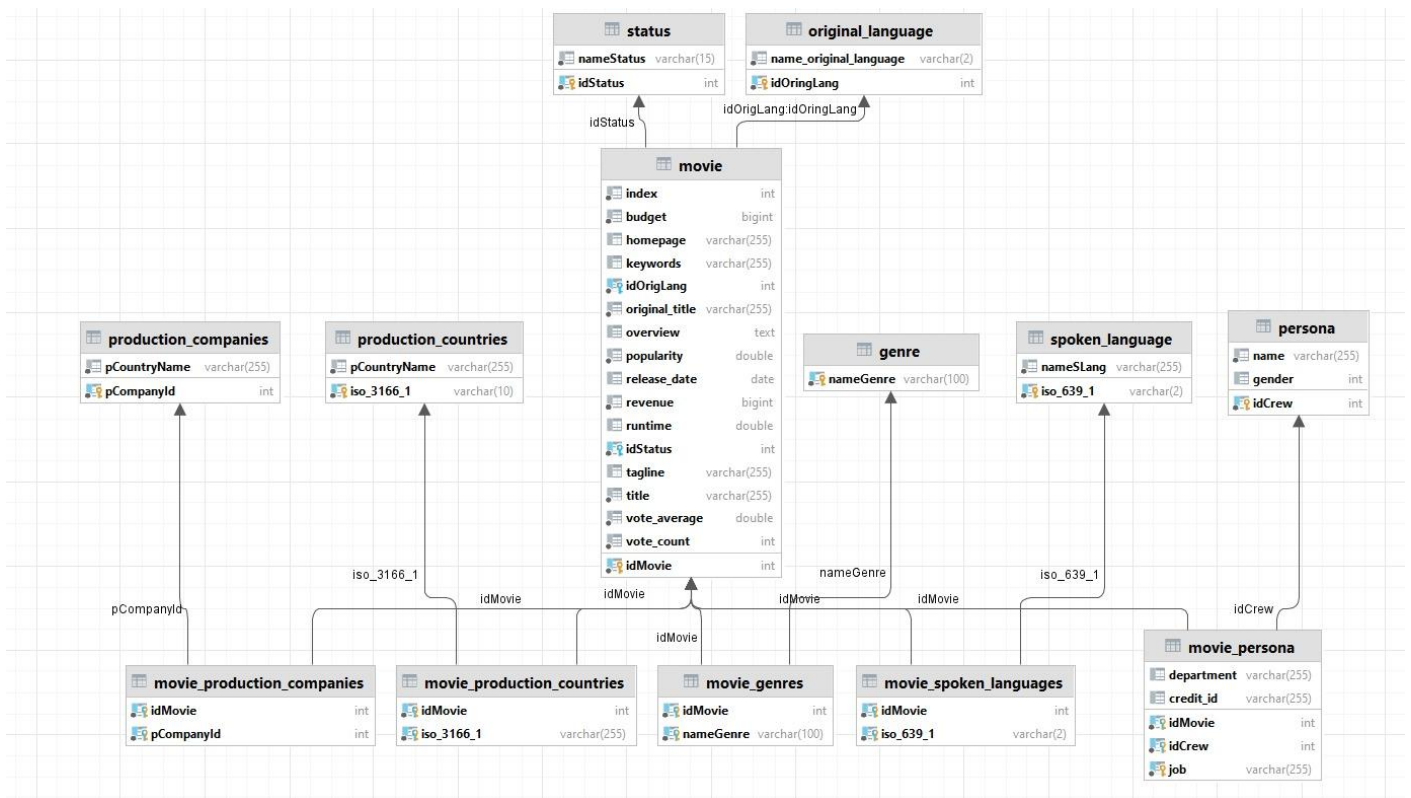
Modelo Lógico

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



Modelo Físico

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



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

4. Tabla Uno a Muchos (1 : N)

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

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

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

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

Como solución, utilizamos el original_languageName como llave foránea en la tabla original_language.

5. Tabla Muchos a Muchos (N : N)

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

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

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

Movie.

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

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

muchas Movie.

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

Production_companies contiene un String de JSON que contiene un name y un id que puede contener 0 a n production_companies.

Para solucionar, cada columna debe contener un solo valor.

- En este caso tenemos una lista de production_companies. Existen múltiples valores.

- La solución es crear una tabla separada que se llame Production_companies.
- Tiene dos atributos, name e id los cuales los nombramos

prodCompName y prodCompId el cual es la primary key.

- Una Movie puede tener muchos production_companies, y un production_companies puede aparecer en muchas Movie.

- La relación (muchos a muchos) se soluciona con una tabla llamada
- Movie_ Production_companies utilizando la llave primaria de cada uno.
- Production_countries contiene un String de JSON que contiene un

iso_3166_1 y un name que puede contener 0 a n production_countries.

- Para solucionar, cada columna debe contener un solo valor.

- En este caso tenemos una lista de production_countries. Existen múltiples valores.
- La solución es crear una tabla separada que se llame Production_countries. Tiene dos atributos, iso_3166_1 y name los cuales los nombramos, iso_3166_1 el cual es la primary key y prodCompName.
- Una Movie puede tener muchos production_countries, y un production_countries puede aparecer en muchas Movie.
- La relación (muchos a muchos) se soluciona con una tabla llamada
- Movie_Production_countries utilizando la llave primaria de cada uno.

Cada Movie tiene uno de los siguientes:

- Production_countries
- Production_companies
- Spoken_languages
- Crew

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

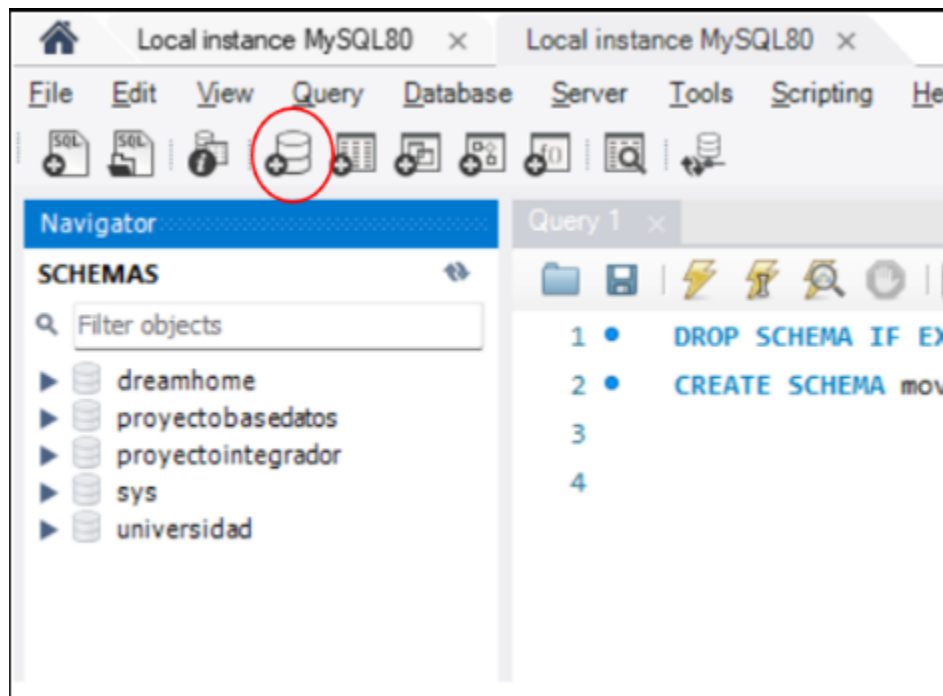
6. Pasos para Realizar el Proyecto

Creación de un "Schema".

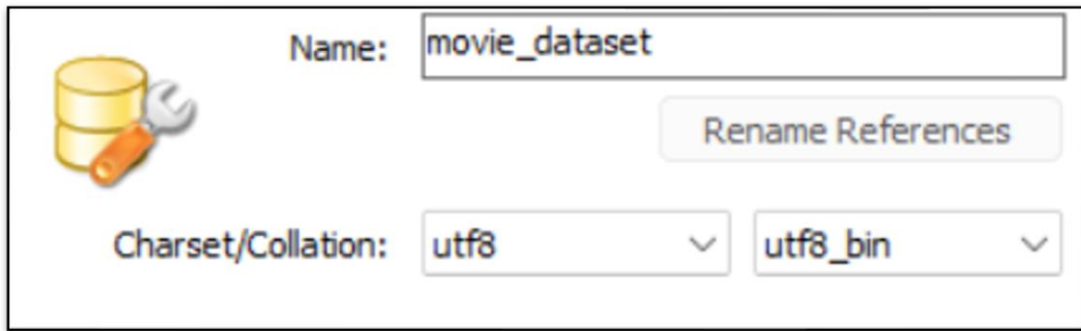
Para la creación del “Schema” se podía realizar de dos distintas maneras, la creación automática o por sentencias SQL.

OPCIÓN 1 – MEDIANTE LA CREACIÓN AUTOMÁTICA

Para la creación automática del Schema dentro del lenguaje de Base de Datos MySQL, primero 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:



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”:



Name:

Charset/Collation:

[Rename References](#)

OPCIÓN 2 – MEDIANTE SENTENCIAS SQL

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

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

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

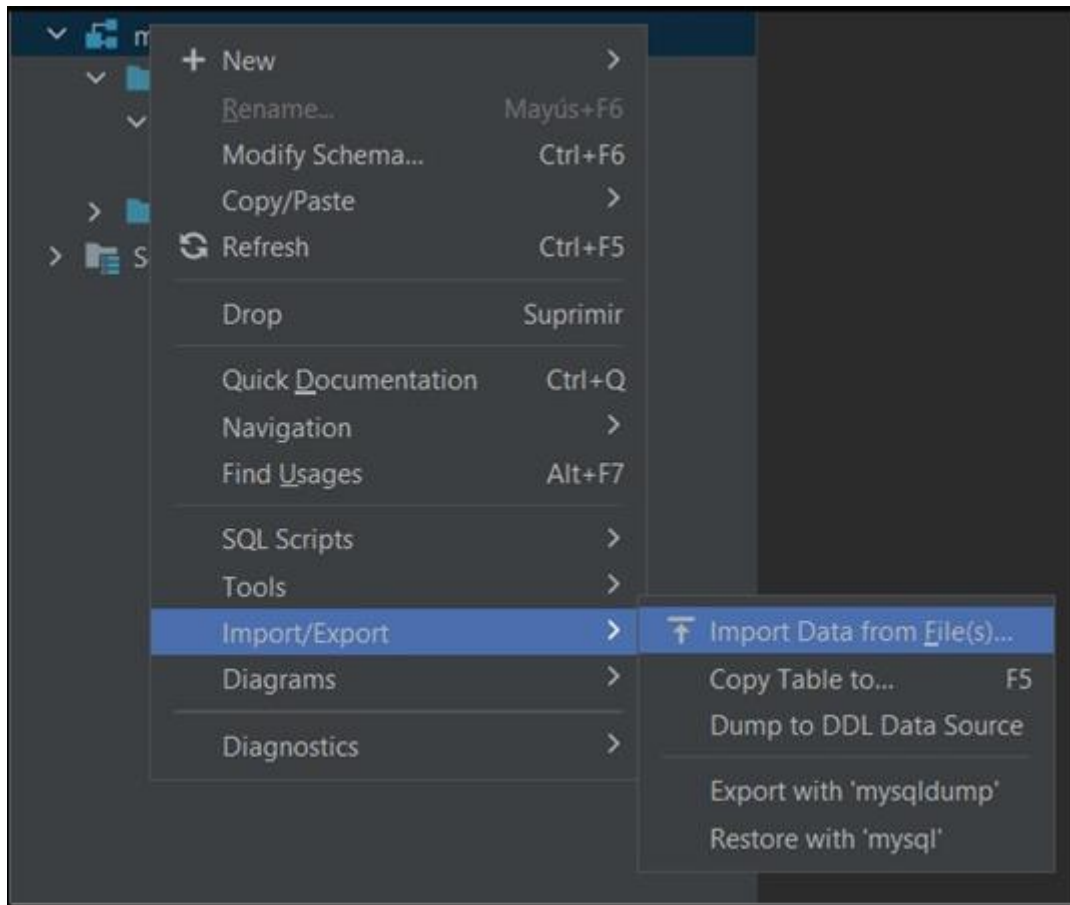
Conexión del “Schema” a la Base De Datos.

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

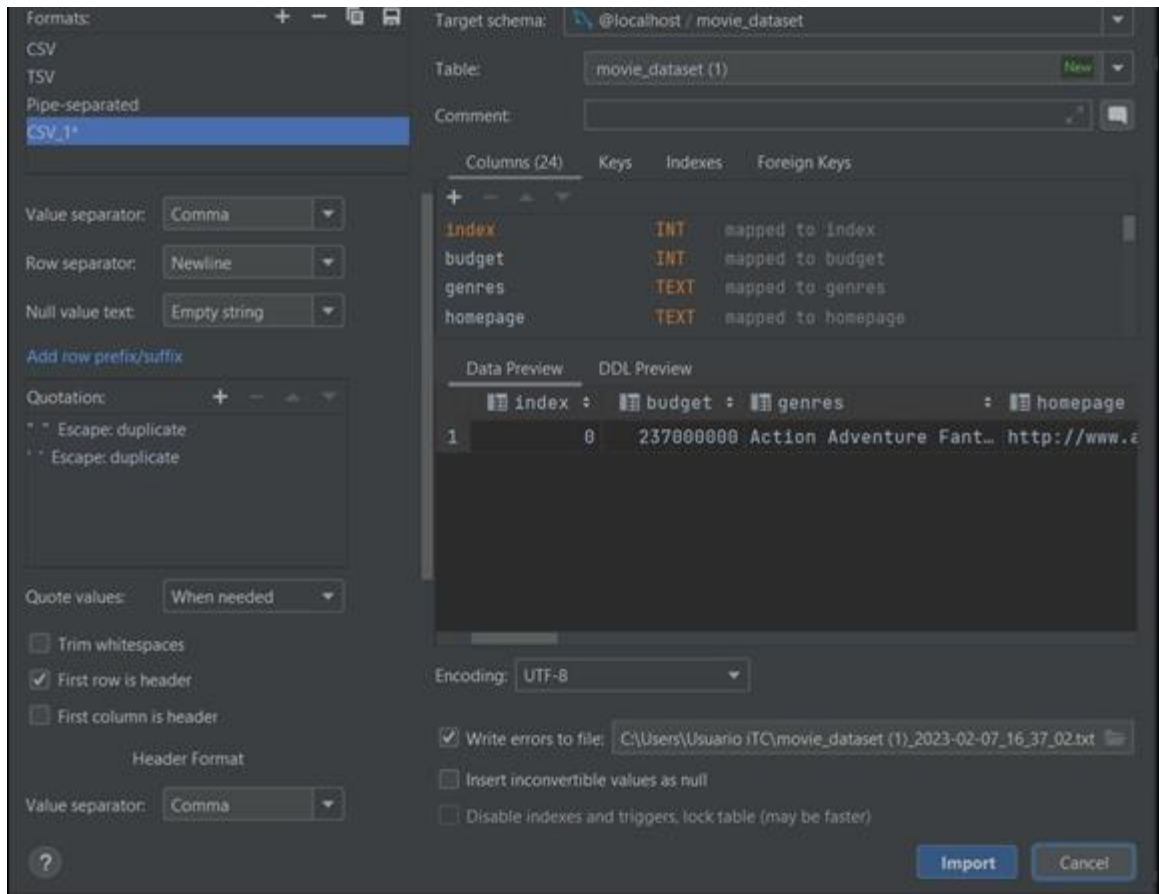
```
USE `Movie_Dataset` ;
```

Importación del CSV.

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



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



Creación de funciones que Permitan Extraer y Limpiar los Datos.

Descripción de las tablas Director, Status y original_langauge

(Tablas uno a muchas)

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

Después, se declara un cursor "CursorDirector" que selecciona los nombres únicos de los directores en la tabla "movie_dataset". Se establece un manejador de continuación "CONTINUE

HANDLER" para detectar cuando se ha alcanzado el final del cursor. Se abre el cursor y se inicia un ciclo repetitivo que recorre cada uno de los nombres de los directores en el cursor.

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

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

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

Este código nos servirá para poblar estas tres tablas.

Población Tabla “Director”

```
1 • USE Movies2023;
2
3 • DROP PROCEDURE IF EXISTS TablaDirector;
4
5 DELIMITER $$
6 • CREATE PROCEDURE TablaDirector()
7   BEGIN
8
9     DECLARE done INT DEFAULT FALSE;
10    DECLARE nameDirector VARCHAR(100);
11
12    -- Declarar el cursor
13    DECLARE CursorDirector CURSOR FOR
14      SELECT DISTINCT CONVERT(director USING UTF8MB4) AS names from movie_dataset;
15
16    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
17    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
18
19    -- Abrir el cursor
20    OPEN CursorDirector;
21    CursorDirector_loop: LOOP
22      FETCH CursorDirector INTO nameDirector;
23
24      -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
25      IF done THEN
```

```

        LEAVE CursorDirector_loop;
27     END IF;
28     IF nameDirector IS NULL THEN
29         SET nameDirector = '';
30     END IF;
31     SET @_oStatement = CONCAT('INSERT INTO directorCURSOR (name) VALUES (\'',
32     REPLACE(REPLACE(nameDirector, '\', '\\'), '\u', '\\u')
33     ,'\');');
34     PREPARE sent1 FROM @_oStatement;
35     EXECUTE sent1;
36     DEALLOCATE PREPARE sent1;
37
38 END LOOP;
39 CLOSE CursorDirector;
40 END $$
41 DELIMITER ;
42
43 CALL TablaDirector();
44
45 DROP TABLE IF EXISTS directorCURSOR;
46
47 CREATE TABLE directorCURSOR (
48     name varchar(255) PRIMARY KEY
49 );

```

Población Tabla “Status”

```

1  USE Movies2023;
2
3  DROP PROCEDURE IF EXISTS TablaStatus;
4
5  DELIMITER $$
6  CREATE PROCEDURE TablaStatus()
7  BEGIN
8
9      DECLARE done INT DEFAULT FALSE;
10     DECLARE nameStatus VARCHAR(100);
11
12     -- Declarar el cursor
13     DECLARE CursorStatus CURSOR FOR
14         SELECT DISTINCT CONVERT(status USING UTF8MB4) AS names from movie_dataset;
15
16     -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
17     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
18
19     -- Abrir el cursor
20     OPEN CursorStatus;
21     CursorStatus_loop: LOOP
22         FETCH CursorStatus INTO nameStatus;
23
24         -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
25     IF done THEN

```

```

        LEAVE CursorStatus_loop;
27     END IF;
28     IF nameStatus IS NULL THEN
29         SET nameStatus = '';
30     END IF;
31     SET @_oStatement = CONCAT('INSERT INTO statusCURSOR (name) VALUES (\'',
32     nameStatus, '\');');
33     PREPARE sent1 FROM @_oStatement;
34     EXECUTE sent1;
35     DEALLOCATE PREPARE sent1;
36
37 END LOOP;
38 CLOSE CursorStatus;
39 END $$
40 DELIMITER ;
41
42 CALL TablaStatus();
43
44 DROP TABLE IF EXISTS statusCURSOR;
45
46 CREATE TABLE statusCURSOR (
47     name varchar(255) PRIMARY KEY
48 );
49
50 SELECT * FROM statusCURSOR;

```

Población Tabla “Original_language”

```

1  USE Movies2023;
2
3  DROP PROCEDURE IF EXISTS TablaOriginalLanguage;
4
5  DELIMITER $$
6  CREATE PROCEDURE TablaOriginalLanguage()
7  BEGIN
8
9      DECLARE done INT DEFAULT FALSE;
10     DECLARE nameOL VARCHAR(100);
11
12     -- Declarar el cursor
13     DECLARE CursorOL CURSOR FOR
14         SELECT DISTINCT original_language AS names from movie_dataset;
15
16     -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
17     DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
18
19     -- Abrir el cursor
20     OPEN CursorOL;
21     CursorOL_loop: LOOP
22         FETCH CursorOL INTO nameOL;
23
24         -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
25     IF done THEN

```

```

26      LEAVE Cursor0L_loop;
27    END IF;
28  IF name0L IS NULL THEN
29    SET name0L = '';
30  END IF;
31  SET @oStatement = CONCAT('INSERT INTO original_languageCURSOR (name) VALUES (\'',
32    name0L, '\');');
33  PREPARE sent1 FROM @oStatement;
34  EXECUTE sent1;
35  DEALLOCATE PREPARE sent1;
36
37  END LOOP;
38  CLOSE Cursor0L;
39  END $$
DELIMITER ;

42  CALL TablaOriginalLanguage();
43
44  DROP TABLE IF EXISTS original_languageCURSOR;
45
46  CREATE TABLE original_languageCURSOR (
47    name varchar(255) PRIMARY KEY
48  );
49
50  SELECT * FROM original_languageCURSOR;

```

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

(REFERENCES)

Población Tabla “Movie”

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

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

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

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

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

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

```

1 DROP PROCEDURE IF EXISTS TablaMovie;
2
3 DELIMITER $$
4 CREATE PROCEDURE TablaMovie()
5 BEGIN
6
7 DECLARE done INT DEFAULT FALSE;
8 DECLARE Movindex INT;
9 DECLARE Movbudget BIGINT;
10 DECLARE Movhomepage VARCHAR(1000);
11 DECLARE MovidMovie INT;
12 DECLARE Movkeywords TEXT;
13 DECLARE Movoriginal_language VARCHAR(255);
14 DECLARE Movoriginal_title VARCHAR(255);
15 DECLARE Movoverview TEXT;
16 DECLARE Movpopularity DOUBLE;
17 DECLARE Movrelease_date VARCHAR(255);
18 DECLARE Movrevenue BIGINT;
19 DECLARE Movruntime DOUBLE;
20 DECLARE Movstatus VARCHAR(255);
21 DECLARE Movtagline VARCHAR(255);
22 DECLARE Movtitle VARCHAR(255);
23 DECLARE Movvote_average DOUBLE;
24 DECLARE Movvote_count INT;
25 DECLARE nameDirector VARCHAR(255);
26
27 DECLARE Director_nameDirector varchar(255);
28 DECLARE Director_nameStatus varchar(255);
29 DECLARE Director_nameOriginal_language varchar(255);
30
31 -- Declarar el cursor
32 DECLARE CursorMovie CURSOR FOR
33 SELECT 'index', budget, homepage, id, keywords, original_language, original_title, overview, popularity, release_date, revenue,
34 tagline, title, vote_average, vote_count, director FROM movie_dataset;
35
36 -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
37 DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
38
39 -- Abrir el cursor
40 OPEN CursorMovie;
41 CursorMovie_loop: LOOP
42 FETCH CursorMovie INTO Movindex, Movbudget, Movhomepage, MovidMovie, Movkeywords, Movoriginal_language, Movoriginal_title, M
43 Movpopularity, Movrelease_date, Movrevenue, Movruntime, Movstatus, Movtagline, Movtitle, Movvote_average, Movvote_count, nameD
44
45 -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
46 IF done THEN
47 LEAVE CursorMovie_loop;
48 END IF;
49 IF nameDirector IS NULL THEN
50 SET nameDirector = '';

```

```

51 END IF;
52
53 SELECT 'name' INTO Director_nameDirector FROM directorCURSOR WHERE directorCURSOR.name = nameDirector;
54 SELECT 'name' INTO Director_nameStatus FROM statusCURSOR WHERE statusCURSOR.name = Movstatus;
55 SELECT 'name' INTO Director_nameOriginal_language FROM original_languageCURSOR WHERE original_languageCURSOR.name = M
56
57 INSERT INTO MovieCURSOR ('index', budget, homepage, id, keywords, original_language, original_title, overview, popularity, rel
58 tagline, title, vote_average, vote_count, director)
59 VALUES (Movindex, Movbudget, Movhomepage, MovidMovie, Movkeywords, Director_nameOriginal_language, Movoriginal_title, Movove
60 Movpopularity, Movrelease_date, Movrevenue, Movruntime, Director_nameStatus, Movtagline, Movtitle, Movvote_average, Movvote_c
61
62 END LOOP;
63 CLOSE CursorMovie;
64 END $$
65 DELIMITER ;
66
67 CALL TablaMovie ();
68
69 DROP TABLE IF EXISTS MovieCURSOR;
70
71 CREATE TABLE MovieCURSOR (
72 'index' int,
73 budget bigint,
74 homepage varchar(1000),
75 id int PRIMARY KEY,
76 keywords TEXT,
77 original_language varchar(255),
78 original_title varchar(255),
79 overview TEXT,
80 popularity double,
81 release_date varchar(255),
82 revenue bigint,
83 runtime double,
84 'status' varchar(255),
85 tagline varchar(255),
86 title varchar(255),
87 vote_average double,
88 vote_count int,
89 director varchar(255),
90 FOREIGN KEY (original_language) REFERENCES original_languageCURSOR(name),
91 FOREIGN KEY (status) REFERENCES statusCURSOR(name),
92 FOREIGN KEY (director) REFERENCES directorCURSOR(name)
93 );
94
95 DROP TABLE IF EXISTS MovieCURSOR;
96
97 SELECT COUNT(*) FROM MovieCursor;
98 SELECT * FROM MovieCursor;

```

Tenemos tres columnas con valores JSON, para las cuales seguiremos el mismo procedimiento para las tres que son spoken_languages, production_companies y production_countries

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

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

El cursor se usa para recorrer las filas de la tabla "movie_dataset" y, para cada fila, se extrae la información de las compañías productoras de las películas utilizando la función "JSON_EXTRACT". Los datos extraídos se insertan en la tabla "production_companieTem".

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

Población Tabla “Production_companies”

```
1 * USE Movies2023;
2
3 * DROP PROCEDURE IF EXISTS TablaProduction_companies;
4
5 DELIMITER $$
6 * CREATE PROCEDURE TablaProduction_companies ()
7
8 BEGIN
9
10 DECLARE done INT DEFAULT FALSE ;
11 DECLARE jsonData json ;
12 DECLARE jsonId varchar(250) ;
13 DECLARE jsonLabel varchar(250) ;
14 DECLARE resultSTR LONGTEXT DEFAULT '';
15 DECLARE i INT;
16
17 -- Declarar el cursor
18 DECLARE myCursor
19 CURSOR FOR
20 SELECT JSON_EXTRACT(CONVERT(production_companies USING UTF8MB4), '$[*]') FROM movie_dataset ;
21
22 -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
23 DECLARE CONTINUE HANDLER
24 FOR NOT FOUND SET done = TRUE ;
25
26 -- Abrir el cursor
27 OPEN myCursor ;
28 drop table if exists production_companietem;
29 SET @sql_text = 'CREATE TABLE production_companieTem ( id int, nameCom VARCHAR(100));';
30 PREPARE stmt FROM @sql_text;
31 EXECUTE stmt;
32 DEALLOCATE PREPARE stmt;
33 cursorLoop: LOOP
34 FETCH myCursor INTO jsonData;
35
36 -- Controlador para buscar cada uno de los arrays
37 SET i = 0;
38
39 -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
40 IF done THEN
41 LEAVE cursorLoop ;
42 END IF ;
43
44 WHILE(JSON_EXTRACT(jsonData, CONCAT('$[', i, ']')) IS NOT NULL) DO
45 SET jsonId = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].id')), '');
46 SET jsonLabel = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].name')), '');
47 SET i = i + 1;
48
49 SET @sql_text = CONCAT('INSERT INTO production_companieTem VALUES (', REPLACE(jsonId, ',', ''), ', ', jsonLabel, '); ');
50 PREPARE stmt FROM @sql_text;
```



```

51     EXECUTE stmt;
52     DEALLOCATE PREPARE stmt;
53
54     END WHILE;
55
56     END LOOP ;
57
58     select distinct * from production_companieTem;
59     INSERT INTO production_companiesCURSOR
60     SELECT DISTINCT id, nameCom
61     FROM production_companieTem;
62     drop table if exists production_companieTem;
63     CLOSE myCursor ;
64
65     END$$
66     DELIMITER ;
67
68     call TablaProduction_companies();
69
70     CREATE TABLE production_companiesCURSOR (
71         id INT PRIMARY KEY,
72         name varchar(100)
73     );
74
75     DROP TABLE production_companiesCURSOR;

```

Población Tabla “Production_countries”

```

1  USE Movies2023;
2
3  DROP PROCEDURE IF EXISTS TablaProduction_countries;
4
5  DELIMITER $$
6  CREATE PROCEDURE TablaProduction_countries ()
7
8  BEGIN
9
10     DECLARE done INT DEFAULT FALSE ;
11     DECLARE jsonData json ;
12     DECLARE jsonId varchar(250) ;
13     DECLARE jsonLabel varchar(250) ;
14     DECLARE resultSTR LONGTEXT DEFAULT '';
15     DECLARE i INT;
16
17     -- Declarar el cursor
18     DECLARE myCursor
19     CURSOR FOR
20         SELECT JSON_EXTRACT(CONVERT(production_countries USING UTF8MB4), '$[*]') FROM movie_dataset ;
21
22     -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
23     DECLARE CONTINUE HANDLER
24     FOR NOT FOUND SET done = TRUE ;
25

```

```

26 -- Abrir el cursor
27 OPEN myCursor ;
28 drop table if exists production_countriesTem;
29 SET @sql_text = 'CREATE TABLE production_countriesTem ( iso_3166_1 varchar(2), nameCountry VARCHAR(100));';
30 PREPARE stmt FROM @sql_text;
31 EXECUTE stmt;
32 DEALLOCATE PREPARE stmt;
33 cursorLoop: LOOP
34   FETCH myCursor INTO jsonData;
35
36   -- Controlador para buscar cada uno de los arrays
37   SET i = 0;
38
39   -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
40   IF done THEN
41     LEAVE cursorLoop ;
42   END IF ;
43
44   WHILE(JSON_EXTRACT(jsonData, CONCAT('$[', i, ']')) IS NOT NULL) DO
45     SET jsonId = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].iso_3166_1')), '');
46     SET jsonLabel = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].name')), '');
47     SET i = i + 1;
48
49     SET @sql_text = CONCAT('INSERT INTO production_countriesTem VALUES (' , REPLACE(jsonId, '\'', ''), ', ', jsonLabel, '); ');
50     PREPARE stmt FROM @sql_text;
51     EXECUTE stmt;
52     DEALLOCATE PREPARE stmt;
53
54   END WHILE;
55
56 END LOOP ;
57
58 select distinct * from production_countriesTem;
59 INSERT INTO production_countriesCURSOR
60 SELECT DISTINCT iso_3166_1, nameCountry
61 FROM production_countriesTem;
62 drop table if exists production_countriesTem;
63 CLOSE myCursor ;
64
65 END$$
66 DELIMITER ;
67
68 call TablaProduction_countries();
69
70 - SELECT * FROM production_countriesCURSOR;
71
72 CREATE TABLE production_countriesCURSOR (
73   iso_3166_1 varchar(2) PRIMARY KEY,
74   name varchar(100)
75 );

```

Población Tabla “Spoken_languages”

```
1 USE Movies2023;
2
3 DROP PROCEDURE IF EXISTS TablaSpokenLanguages;
4
5 DELIMITER $$
6 CREATE PROCEDURE TablaSpokenLanguages ()
7
8 BEGIN
9
10  DECLARE done INT DEFAULT FALSE ;
11  DECLARE jsonData json ;
12  DECLARE jsonId varchar(250) ;
13  DECLARE jsonLabel varchar(250) ;
14  DECLARE resultSTR LONGTEXT DEFAULT '';
15  DECLARE i INT;
16
17  -- Declarar el cursor
18  DECLARE myCursor
19  CURSOR FOR
20    SELECT JSON_EXTRACT(CONVERT(spoken_languages USING UTF8MB4), '$[*]') FROM movie_dataset ;
21
22  -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
23  DECLARE CONTINUE HANDLER
24  FOR NOT FOUND SET done = TRUE ;
25
26  -- Abrir el cursor
27  OPEN myCursor ;
28  drop table if exists spokenLanguagesTem;
29  SET @sql_text = 'CREATE TABLE spokenLanguagesTem ( iso_639_1 varchar(2), nameLang VARCHAR(100));';
30  PREPARE stmt FROM @sql_text;
31  EXECUTE stmt;
32  DEALLOCATE PREPARE stmt;
33  cursorLoop: LOOP
34    FETCH myCursor INTO jsonData;
35
36    -- Controlador para buscar cada uno de los arrays
37    SET i = 0;
38
39    -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
40    IF done THEN
41      LEAVE cursorLoop ;
42    END IF ;
43
44    WHILE(JSON_EXTRACT(jsonData, CONCAT('$[', i, ']')) IS NOT NULL) DO
45      SET jsonId = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].iso_639_1')), '');
46      SET jsonLabel = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i, '].name')), '');
47      SET i = i + 1;
48
49      SET @sql_text = CONCAT('INSERT INTO spokenLanguagesTem VALUES (' , REPLACE(jsonId, '\'', ''), ', ' , jsonLabel, '); ');
50      PREPARE stmt FROM @sql_text;
```

```

51     EXECUTE stmt;
52     DEALLOCATE PREPARE stmt;
53
54     END WHILE;
55
56     END LOOP ;
57
58     select distinct * from spokenLanguagesTem;
59     INSERT INTO spoken_languagesCURSOR
60     SELECT DISTINCT iso_639_1, nameLang
61     FROM spokenLanguagesTem;
62     drop table if exists spokenLanguagesTem;
63     CLOSE myCursor ;
64
65     END$$
66     DELIMITER ;
67
68     call TablaSpokenLanguages();
69
70     SELECT * FROM spoken_languagesCURSOR;
71
72     CREATE TABLE spoken_languagesCURSOR (
73         iso_639_1 varchar(2) PRIMARY KEY,
74         name varchar(100)
75     );

```

7. LIMPIEZA COLUMNA CREW

1. Primero, usemos IntelliJ y el lenguaje de programación Scala para leer el archivo CSV y comenzar a analizar las columnas de Crew. Encontré algunos casos en los que el valor de la clave "Nombre" contenía comillas simples.
2. Analice diferentes patrones en los datos para encontrar cuándo se usan comillas simples, como en O'Brian, o cuándo se usan comillas dobles para encerrar apodos como "D.J." I was. Las comillas son hasta \', por lo que se consideran parte de la cadena de nombre, no JSON.
3. El análisis de los casos muestra que hay casos como e', t' o d' y estos patrones coinciden con las claves 'nombre', 'departamento', 'id_crédito' o 'id'. Por lo tanto, no era posible reemplazar el valor e', t' o d' con e\', t\' o d\', ya que provocaría un error en la clave.

4. Para las intercalaciones 0, 1 y 2, tuve que convertir según la posición y el carácter al lado.
Como tal, simplemente ponlo entre comillas.
5. El valor clave de "id" también tenía que estar entre comillas simples, así que lo puse de acuerdo con el valor antes y después de "\"id\":" , pero se convirtió en "\"id\":" puse ' entre comillas antes del valor, reemplazó '}',' con '\"}',' y colocó una comilla simple al final del valor.
6. En el caso del último valor de "id" reemplazamos '}]' al colocar '\"}]' en su lugar. A diferencia del último reemplazo en el punto 5, el } no está seguido por un , debido a que es el último valor del arreglo. Termina con una llave] así que por eso tomamos este caso en consideración para que todos los últimos valores de key tengan comillas cerradas.
7. Tras realizar todos los REPLACE, realizamos un JSON_EXTRACT JSON_EXTRACT (CONVERT(*SENTENCIA_DE_TODOS_LOS REPLACE* USING UTF8MB4), '\$[*]') para extraer todos los valores JSON.
8. Hemos realizado exitosamente la limpieza de la columna "crew". El siguiente paso es realizar la carga de estos datos a su respectiva tabla/s.

8. Consultas.

Ejemplo 1

```
SELECT C.id AS "IdCrew", C.name AS "Nombre", C.departament AS "Departamento", C.job AS "TRABAJO/ROL"  
FROM Crew C, Movie O  
WHERE C.idMovie = (SELECT id FROM Movie WHERE original_title = "Avatar");
```

	IdCrew	Nombre	Departamento	TRABAJO/ROL
►	1621932	Min Windle	Crew	Stunts
	1483234	Ben White	Crew	CG Supervisor
	1483233	David Weitzberg	Crew	CG Supervisor
	1483232	Michael Takarangi	Crew	CG Supervisor
	1483231	Philippe Rebours	Crew	CG Supervisor
	1483230	Sergei Nevshupov	Crew	CG Supervisor
	1483229	Matthias Menz	Crew	CG Supervisor
	1483228	Sebastian Marino	Crew	CG Supervisor
	1483227	Andy Lomas	Crew	CG Supervisor
	1483226	Jerry Kung	Crew	CG Supervisor
	1483225	Mitch Gates	Crew	CG Supervisor
	1483224	Adrian Fernandes	Crew	CG Supervisor
	1483223	Graeme Demmocks	Crew	CG Supervisor
	1483222	Simon Clutterbuck	Crew	CG Supervisor
	1483221	Shadi Almassizadeh	Crew	CG Supervisor
	1483220	Brad Alexander	Crew	CG Supervisor
	1466035	Thrain Shadbolt	Crew	CG Supervisor
	1457305	Georgia Lockhart...	Costume & M...	Makeup Artist
	1453938	Arun Ram-Mohan	Lighting	Lighting Artist
	1452643	Roxane Griffin	Costume & M...	Hairstylist

Ejemplo 2

```
SELECT DISTINCT M.title AS "Titulo", S.name as "Idioma Original"  
FROM Movie M JOIN spoken_language S ON M.revenue > M.budget  
WHERE S.name != "English";
```

	Titulo	Idioma Original
►	Puss in Boots	Español
	The Monkey King 2	Español
	Bad Moms	Español
	Airlift	Español
	Ip Man 3	Español
	Fifty Shades of Black	Español
	God's Not Dead 2	Español
	Lights Out	Español
	Keanu	Español
	Miracles from Heaven	Español
	The Birth of a Nation	Español
	Risen	Español
	10 Cloverfield Lane	Español
	Love the Coopers	Español
	The Shallows	Español
	Captive	Español
	Zoolander 2	Español
	The Forest	Español

Ejemplo 3

```
SELECT DISTINCT S.name AS "Doblado", M.original_title as "Titulo Original"  
FROM spoken_language S, movie M  
WHERE (SELECT COUNT(*) FROM production_companies) > 3;
```

	Doblado	Titulo Original
	English	American Beauty
	English	Forrest Gump
	English	Finding Nemo
	English	Star Wars
	English	Four Rooms
	Español	To Be Frank, Sinatra at 100
	Español	Running Forever
	Español	8 Days
	Español	Growing Up Smith
	Español	Midnight Cabaret
	Español	Puss in Boots
	Español	Perfect Cowboy
	Español	인천상륙작전
	Español	Solitude
	Español	Restoration
	Español	The Secret
	Español	The Dog Lover
	Español	Code of Honor
	Español	A Beginner's Guide to Snuff

9. Conclusiones

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

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

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