UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

Informe Final

Fundamentos de Bases de Datos

Autor:

- Jonathan Anibal Coronel Alulima

Octubre 2022 – Febrero 2023

Índice

Introducción

Este proyecto se centra en la importación y procesamiento de datos desde un archivo CSV llamado "*movie_dataset*" a una base de datos alojada en el motor MySQL.

El objetivo es crear un sistema automatizado para la limpieza y preparación de los datos en el archivo CSV y su posterior carga en la base de datos. Para ello, utilizaremos procesos en MySQL para realizar tareas de limpieza, tales como la eliminación de registros duplicados, la corrección de errores de ortografía y la conversión de valores faltantes en un formato uniforme.

Una vez que los datos estén limpios y estructurados, los cargaremos en la base de datos con el uso de procesos en MySQL para garantizar una carga rápida y eficiente. Además, creamos tablas y relaciones apropiadas para asegurar que la información esté almacenada y organizada de manera adecuada para su uso posterior.

Este proyecto es esencial para proporcionar un sistema eficiente y automatizado para la limpieza y carga de datos en la base de datos MySQL.

Dependencias Funcionales – Tabla Universal

Lo primero que realizamos es determinar las dependencias Funcionales para poder tener una idea claro de como estructurar nuestros modelos.

Figura 1

Dependencia Funcional del csv *movie_dataset*

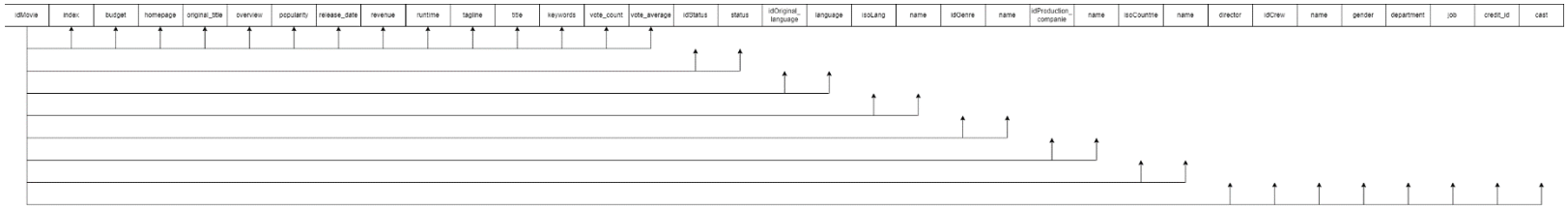


Figura 2

Textualización de las relaciones en la dependencia Funcional

idMovie -> {index, budget, homepage, original_title, overview, popularity, release_date, revenue, runtime, tagline, title, keywords, vote_count, vote_average}

idMovie -> {idStatus, status}

idMovie -> {idOriginal_language, language}

idMovie -> {isoLang, name}

idMovie -> {idGenre, name}

idMovie -> {idProduction_companie, name}

idMovie -> {isoCountrie, name}

idMovie -> {director, idCrew, name, gender, department, job, credit_id, cast}

Con las dependencias funcionales creadas lo que hacemos es verificar si cumplen con la normalización para poder pasar a crear los modelos.

Normalización

First Normal Form (1NF)

Eliminación de atributos repetidos: Cada atributo de la tabla debe ser único. Si se encuentra un atributo repetido, se debe eliminar y crear una tabla nueva para almacenar los

valores repetidos. Identificación de llaves primarias: Se debe identificar y establecer una llave primaria para cada tabla, que permita identificar de manera única cada registro en la tabla.

Para cumplir con esta norma, se verificó que cada columna en la tabla de películas tuviera un valor único y no se repitiera. Por ejemplo, para el título de cada película, se garantizó que no hubiera dos películas con el mismo título. De igual manera, para otros atributos como el presupuesto, la popularidad y la duración, se aseguró que cada valor fuera único para cada película.

Second Normal Form (2NF)

Se debe cumplir con la 1FN.

Cada atributo no primario debe estar completamente dependiente de la llave primaria. Si hay una dependencia parcial, se debe eliminar el atributo dependiente y crear una nueva tabla para almacenarlo.

Para cumplir con esta norma, se verificó que cada atributo dependiera funcionalmente del identificador de la tabla, que en este caso es el ID de la película. Por ejemplo, la información sobre las palabras clave, las compañías de producción y los países de producción, depende únicamente del ID de la película y no de otros atributos en la tabla.

Third Normal Form (3NF)

Se debe cumplir con la 2FN.

Cada atributo no primario no debe estar dependiente de otro atributo no primario. Si existe una dependencia, se debe eliminar el atributo dependiente y crear una nueva tabla para almacenarlo.

Para cumplir con esta norma, se identificaron las dependencias transitorias en la tabla de películas y se separaron en tablas separadas. Por ejemplo, la información sobre los actores

y el equipo de producción se puede separar en tablas separadas para evitar dependencias transitorias en la tabla de películas.

Además de cumplir con las normas de normalización, también se verificó que los valores de los atributos estuvieran correctamente alineados con los tipos de datos correctos, por ejemplo, la fecha de lanzamiento como tipo de fecha, la duración como tipo numérico, etc

Con Todas las normas cumplidas procedemos a crear nuestros Modelos de *movie_daset.*

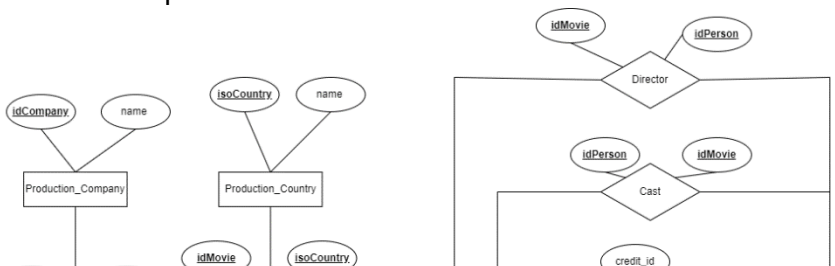
Modelos

Modelo Conceptual

La creación del modelo conceptual nos permite representar la información a través de entidades y relaciones claramente definidas, que permiten una comprensión fácil y eficiente de la estructura de los datos. Este modelo conceptual es esencial para la planificación y diseño de la base de datos movie_dataset, ya que proporciona una visión general clara y concisa de la información almacenada en la base de datos y cómo está organizada. Además, permite la identificación de cualquier posible mejora o ajuste en la estructura de los datos para asegurar una gestión eficiente de la información.

Figura 3

Imagen del Modelo Conceptual Creado



Identificación de entidades:

a. *Movie*: Una película es el objeto principal del modelo y puede ser identificada por su ID único.

b. *Genre*: Un género es una categoría en la que una película puede ser clasificada y puede ser identificada por su nombre.

c. *Production_Companies*: Una compañía de producción es una organización que produce películas y puede ser identificada por su nombre.

d. *Production_Countries*: Un país de producción es un país donde se produjo una película y puede ser identificado por su nombre.

e. *Spoken_Language*: El idioma en los que se tradujo dicha película y puedo tener varios lengujes.

f. *Status*: El estado de la película, en este Dataset dicho estado solo puede tener tres opciones distintas.

g. *Original_language*: El idioma original en el cual se ha grabado la película.

h. *Person*: Son las personas que han trabajado en la producción de la película.

Identificación de atributos:

a. *Movie*: idMovie, index, budget, homepage, original_title, overview, popularity, release_date, revenue, runtime, tagline, title, keywords, vote_count, vote_average, idStatus, idOriginal_language.

b. Genre: idGenre, name.

c. *Production_companies*: idProduction_companie, name.

d. *Production_countries*: isoCountrie, name.

e. *Spoken_language*: isoLang, name.

f. *Status*: idStatus, status.

g. *Original_language*: idOriginal_language, language

h. *Person*: idCrew, name, gender.

Identificación de relaciones:

a. La relación entre *Movie* y *Genre* es una relación de muchos a muchos, donde una película puede pertenecer a uno o más géneros, y un género puede ser asignado a una o más películas.

b. La relación entre *Movie* y *Production_companie* es una relación de muchos a muchos, donde una película puede ser producida por una o más compañías de producción y una compañía puede producir una o más películas.

c. La relación entre *Movie* y *Production_countrie* es una relación de muchos a muchos, donde una película puede ser producida en uno o más países de producción, y cada país de producción produce una o más películas.

d. La relación entre *Movie* y *Spoken_language* es una relación de muchos a muchos, donde cada película puede tener uno o más lenguajes y cada lenguaje puede ser asignado a uno o más películas.

e. La relación entre *Movie* y *Person* es una relación de muchos a muchos, donde una película puede incluir uno o más personas, y cada persona puede haber participado en uno o más películas.

f. La relación entre *Movie* y *Status* es una relación de uno a muchos, donde una película puede tener solo un estado, pero un estado puede estar en una o más películas.

g. La relación entre *Movie* y *Original_language* es una relación de uno a muchos, donde una película puede tener solo un idioma original, pero el idioma original puede estar en una o más películas.

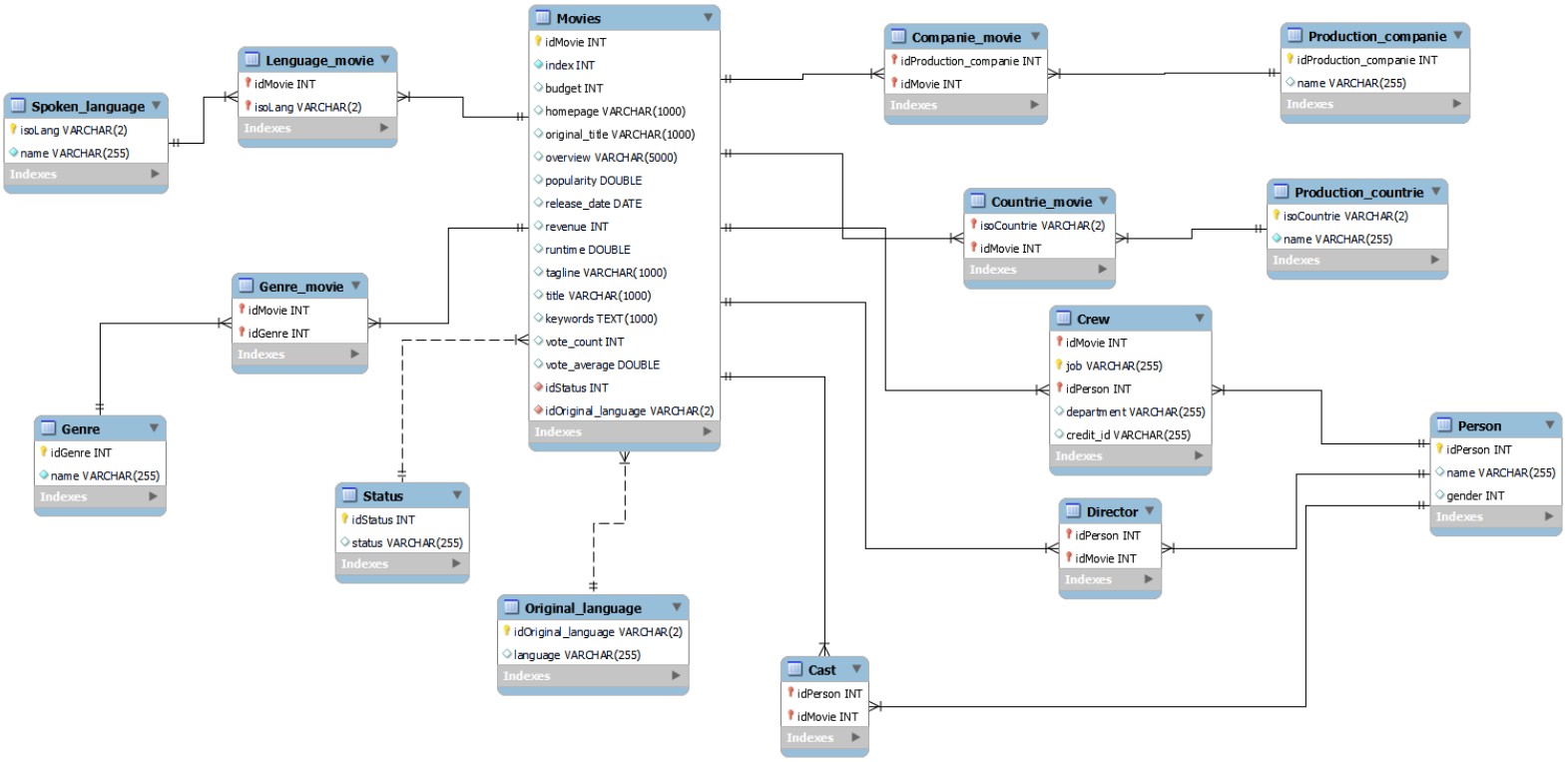
Una vez definido estos parámetros continuamos a la creación del Modelo Lógico.

Modelo Lógico

Utilizamos la herramienta *Workbench* para diseñar un modelo claro y eficiente que captura la estructura de los datos y las relaciones entre ellos. Este modelo nos permitirá acceder a la información de manera organizada y fácil de entender. La imagen del modelo lógico nos permitirá entender de mejor manera todas nuestras relaciones.

Figura 4

Modelo lógico generado en *Workbench*

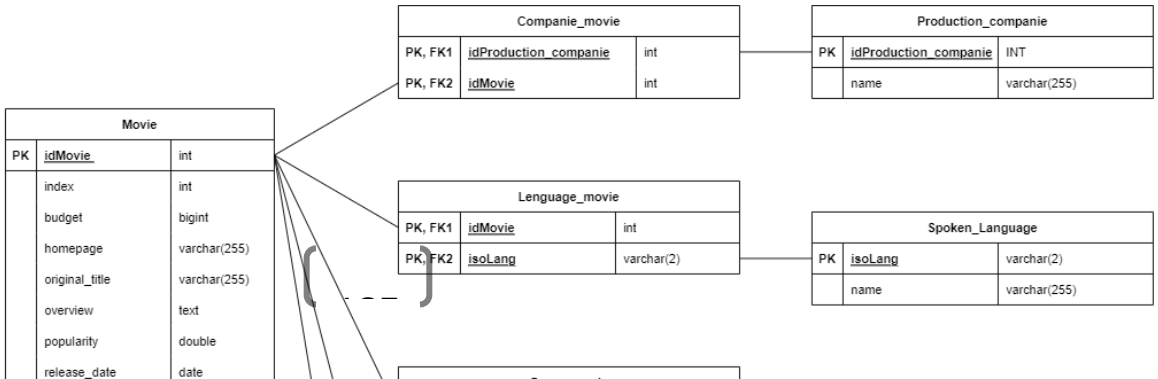


Modelo Físico

Este modelo representa de manera detallada cómo se almacenarán los datos en la base de datos, incluyendo la estructura de las tablas, las relaciones entre ellas y los tipos de datos que se utilizarán. Es importante tener un modelo físico preciso ya que es la base para el correcto funcionamiento de la base de datos y permite garantizar la integridad y la seguridad de los datos.

Figura 5

Imagen del Modelo Físico Obtenido



Con nuestro modelo físico generado podemos seguir a la creación de nuestra base de datos con la cual vamos a trabajar.

```
-- -----------------------------------------------------
-- 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`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Original_language`
-- -----------------------------------------------------
```

```sql
DROP TABLE IF EXISTS `Original_language` ;
CREATE TABLE IF NOT EXISTS `Original_language` (
  `idOriginal_language` VARCHAR(2) NOT NULL,
  `language` VARCHAR(255) NULL,
  PRIMARY KEY (`idOriginal_language`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- 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` INT NULL,
  `runtime` DOUBLE NULL,
  `tagline` VARCHAR(1000) NULL,
  `title` VARCHAR(1000) NULL,
  `keywords` TEXT(1000) NULL,
  `vote_count` INT NULL,
  `vote_average` DOUBLE NULL,
  `idStatus` INT NOT NULL,
  `idOriginal_language` VARCHAR(2) NOT NULL,
  PRIMARY KEY (`idMovie`),
    FOREIGN KEY (`idStatus`)
    REFERENCES `Status` (`idStatus`),
    FOREIGN KEY (`idOriginal_language`)
    REFERENCES `Original_language` (`idOriginal_language`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- 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`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- 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`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- 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`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- 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`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- 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`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- 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`))
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- 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`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- 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`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Person`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Person` ;
CREATE TABLE IF NOT EXISTS `Person` (
  `idPerson` INT NOT NULL,
  `name` VARCHAR(255) NULL,
  `gender` INT NULL,
  PRIMARY KEY (`idPerson`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `Crew`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Crew` ;
CREATE TABLE IF NOT EXISTS `Crew` (
  `idMovie` INT NOT NULL,
  `job` VARCHAR(255) NOT NULL,
  `idPerson` INT NOT NULL,
  `department` VARCHAR(255) NULL,
  `credit_id` VARCHAR(255) NULL,
  PRIMARY KEY (`idMovie`, `job`, `idPerson`),
    FOREIGN KEY (`idMovie`)
    REFERENCES `Movies` (`idMovie`),
    FOREIGN KEY (`idPerson`)
    REFERENCES `Person` (`idPerson`))
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `Cast`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Cast` ;
CREATE TABLE IF NOT EXISTS `Cast` (
  `idPerson` INT NOT NULL,
  `idMovie` INT NOT NULL,
  PRIMARY KEY (`idPerson`, `idMovie`),
    FOREIGN KEY (`idPerson`)
    REFERENCES `Person` (`idPerson`),
    FOREIGN KEY (`idMovie`)
    REFERENCES `Movies` (`idMovie`))
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `Director`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `Director` ;
CREATE TABLE IF NOT EXISTS `Director` (
  `idPerson` INT NOT NULL,
  `idMovie` INT NOT NULL,
  PRIMARY KEY (`idPerson`, `idMovie`),
    FOREIGN KEY (`idPerson`)
    REFERENCES `Person` (`idPerson`),
    FOREIGN KEY (`idMovie`)
    REFERENCES `Movies` (`idMovie`))
ENGINE = InnoDB;
```

Pasos para Realizar el Proyecto

Creación de un “Schema”.

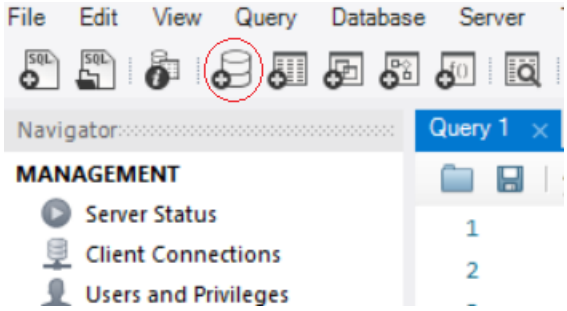
Para la creación del “*Schema*” se podía realizar de dos distintas maneras, la creación automática o por sentencias SQL.

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

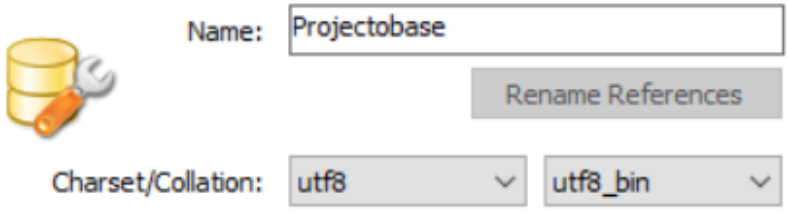
Representación de la barra de opciones perteneciente a Workbench



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:

Figura 8

Sentencia para crear un Schema

```
CREATE DATABASE IF NOT EXISTS `Projectobase` 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:

Figura 9

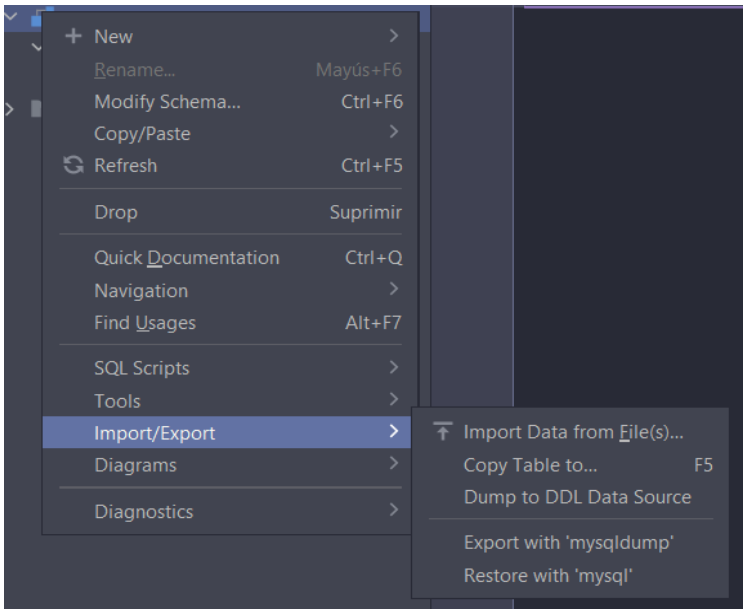
Usar el Schema creado

Importación del CSV.

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

Figura 10

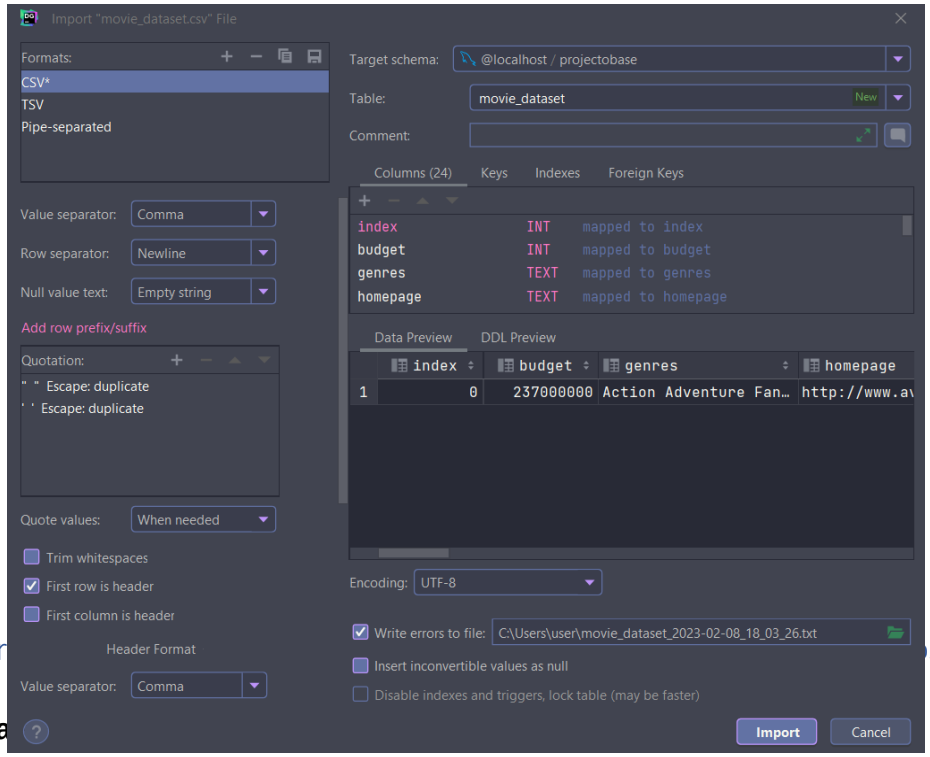
Representación de la funcionalidad de importación de archivos en DataGrip



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 11

Representación de la interfaz de importación de archivos en DataGrip



Cr s.

Ta

Figura 11

Procedimiento que permite extraer y Limpiar la tabla Original_language

```
-- Tabla Original_language--
DROP PROCEDURE IF EXISTS TablaOriginal_language;
DELIMITER $$
CREATE PROCEDURE TablaOriginal_language()
BEGIN
```

Este es un procedimiento que crea una tabla "original_language" y llena sus valores desde la tabla "movie_dataset". En primer lugar, verifica si ya existe un procedimiento con el nombre "TablaOriginal_language" y en caso de existir, la borra. Luego, define un delimitador personalizado "$$" que se usará como delimitador en lugar del delimitador estándar ";".

A continuación, crea la procedura "TablaOriginal_language". Se declara una variable "namelanguage" de tipo VARCHAR (100) que se usará para almacenar los valores obtenidos del cursor. Se crea un cursor llamado "Cursorlanguage" que selecciona los valores distintos de la columna "original_language" de la tabla "movie_dataset".

También se crea un controlador de continuación llamado "NOT FOUND" que se activará cuando el cursor haya alcanzado su final. Se abre el cursor y se entra en un bucle "CursorDirector_loop". Dentro de este bucle, se extrae la siguiente fila del cursor y se almacena en la variable "namelanguage".

Si se alcanzó el final del cursor, se sale del bucle. Se verifica si la variable "namelanguage" es NULL y en caso de ser así, la establece en una cadena vacía. Concatena una consulta SQL para insertar el valor de "namelanguage" en la tabla "original_language". Prepara y ejecuta la consulta SQL y dealloca la consulta preparada.

Finalmente, se cierra el cursor, se termina la procedura y se establece de nuevo el delimitador estándar ";". La línea "CALL TablaOriginal_language();" ejecuta la procedura.

Tabla Status

Figura 12

Procedimiento que permite extraer y Limpiar la tabla Status

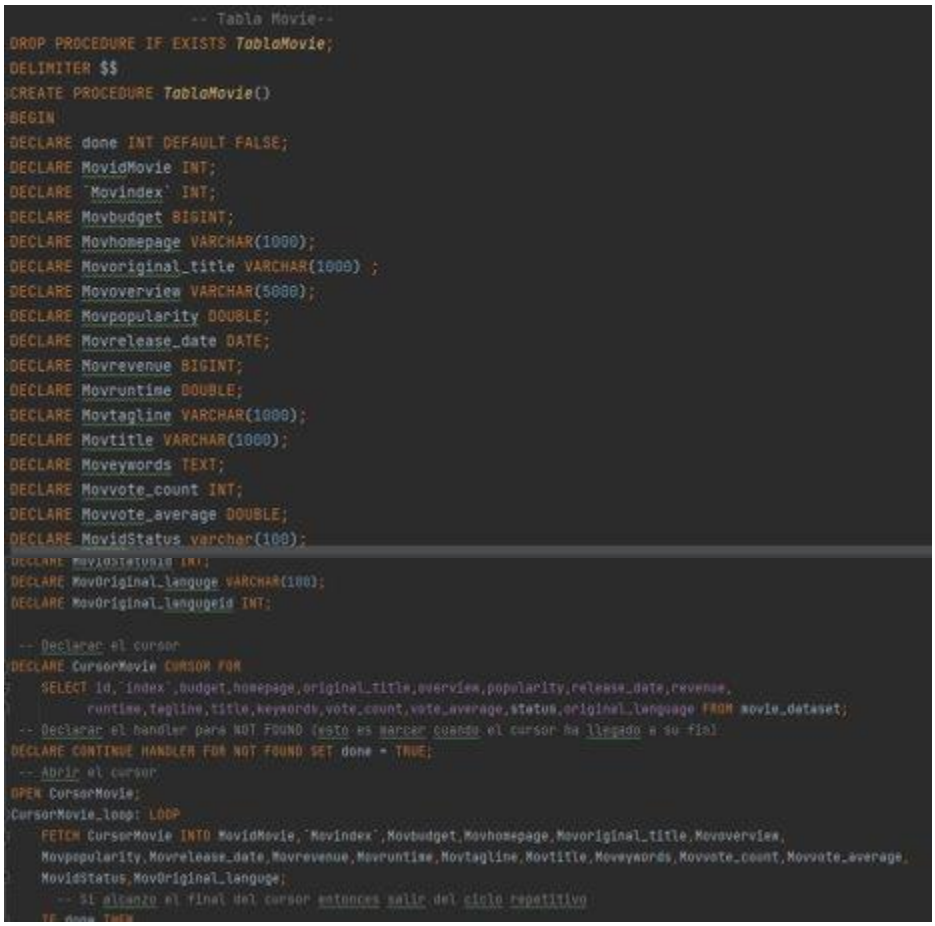
```
-- Tabla Status--
DROP PROCEDURE IF EXISTS TablaStatus;
DELIMITER $$
CREATE PROCEDURE TablaStatus()
BEGIN
DECLARE done INT DEFAULT FALSE;
DECLARE nameStatus VARCHAR(100);
 -- Declarar el cursor
DECLARE CursorStatus CURSOR FOR
    SELECT DISTINCT CONVERT(status USING UTF8MB4) from movie_dataset;
 -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
 -- Abrir el cursor
OPEN CursorStatus;
CursorDirector_loop: LOOP
    FETCH CursorStatus INTO nameStatus;
      -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE CursorDirector_loop;
    END IF;
    IF nameStatus IS NULL THEN
        SET nameStatus = '';
    END IF;
    SET @_oStatement = CONCAT('INSERT INTO status (status) VALUES (\'',nameStatus,'\');');
    PREPARE sent1 FROM @_oStatement;
    EXECUTE sent1;
    DEALLOCATE PREPARE sent1;
END LOOP;
CLOSE CursorStatus;
END $$
DELIMITER ;
CALL TablaStatus ();
```

En este procedimiento se hace lo mismo que lo que explico en el procedimiento de original_language simplemente se están cambiando variables.

Tabla Movie

Figura 13

Procedimiento que permite extraer y Limpiar la tabla Movie



```
                    -- Tabla Movie--
DROP PROCEDURE IF EXISTS TablaMovie;
DELIMITER $$
CREATE PROCEDURE TablaMovie()
BEGIN
DECLARE done INT DEFAULT FALSE;
DECLARE MovidMovie INT;
DECLARE `Movindex` INT;
DECLARE Movbudget BIGINT;
DECLARE Movhomepage VARCHAR(1000);
DECLARE Movoriginal_title VARCHAR(1000) ;
DECLARE Movoverview VARCHAR(5000);
DECLARE Movpopularity DOUBLE;
DECLARE Movrelease_date DATE;
DECLARE Movrevenue BIGINT;
DECLARE Movruntime DOUBLE;
DECLARE Movtagline VARCHAR(1000);
DECLARE Movtitle VARCHAR(1000);
DECLARE Moveywords TEXT;
DECLARE Movvote_count INT;
DECLARE Movvote_average DOUBLE;
DECLARE MovidStatus varchar(100);
DECLARE MovidStatusid INT;
DECLARE MovOriginal_languge VARCHAR(100);
DECLARE MovOriginal_langugeid INT;

 -- Declarar el cursor
DECLARE CursorMovie CURSOR FOR
    SELECT id,`index`,budget,homepage,original_title,overview,popularity,release_date,revenue,
           runtime,tagline,title,keywords,vote_count,vote_average,status,original_language FROM movie_dataset;
 -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
 -- Abrir el cursor
OPEN CursorMovie;
CursorMovie_loop: LOOP
    FETCH CursorMovie INTO MovidMovie,`Movindex`,Movbudget,Movhomepage,Movoriginal_title,Movoverview,
    Movpopularity,Movrelease_date,Movrevenue,Movruntime,Movtagline,Movtitle,Moveywords,Movvote_count,Movvote_average,
    MovidStatus,MovOriginal_languge;
    -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE CursorMovie_loop;
    END IF;
    SELECT idStatus INTO MovidStatusid FROM status WHERE status.status=MovidStatus;
    SELECT idOriginal_language INTO MovOriginal_langugeid FROM original_language
                                                WHERE original_language.language=MovOriginal_languge;
    INSERT INTO movies
    VALUES (MovidMovie,`Movindex`, Movbudget, Movhomepage, Movoriginal_title, Movoverview, Movpopularity,
            Movrelease_date, Movrevenue, Movruntime, Movtagline, Movtitle,Moveywords,Movvote_count,
            Movvote_average,MovidStatusid,MovOriginal_langugeid);
    END LOOP;
CLOSE CursorMovie;
END $$
DELIMITER ;
CALL TablaMovie ();
```

El procedimiento comienza declarando varias variables para almacenar los valores de los diferentes atributos de la película. Estos atributos incluyen el ID de la película, título original, popularidad, fecha de lanzamiento, ganancias, tiempo de ejecución, resumen, etc.

Luego, se declara un cursor llamado "CursorMovie" para seleccionar todos los atributos de las películas desde la tabla "movie_dataset". Un "handler" para "NOT FOUND" se declara también para marcar cuando el cursor ha llegado al final.

Después, el cursor se abre y se inicia un ciclo repetitivo para recorrer todas las películas. En cada iteración, se utiliza el comando "FETCH" para obtener los valores de los atributos de la película actual y almacenarlos en las variables correspondientes.

El procedimiento también incluye consultas para obtener el ID de la película en la tabla de estado y la tabla de idioma original. Una vez que se tienen todos los valores, se inserta una nueva fila en la tabla "movies" con todos los atributos.

El ciclo repetitivo continúa hasta que se alcance el final del cursor y se cierra. Finalmente, el procedimiento termina.

Tabla Production_comapnie

Figura 13

Procedimiento que permite extraer y Limpiar la tabla Production_companie

```
-- Tabla Companie------
DROP PROCEDURE IF EXISTS TablaCompanie ;
DELIMITER $$
CREATE PROCEDURE TablaCompanie ()
BEGIN
 DECLARE done INT DEFAULT FALSE ;
 DECLARE jsonData json ;
```

Este Procedimiento crea una tabla llamada "production_companie" a partir de datos en una columna llamada "production_companies" en la tabla "movie_dataset".

El procedimiento comienza por definir una serie de variables, incluyendo un cursor llamado "myCursor" que se utiliza para recorrer los registros en la columna "production_companies". También se define un handler "continue" para indicar cuándo se ha llegado al final del cursor.

A continuación, se crea una tabla temporal "production_companieTem" para almacenar los datos extraídos de la columna "production_companies". Luego, se abre el cursor y se inicia un loop para recorrer cada registro en la columna "production_companies". Para cada registro, se utiliza la función JSON_EXTRACT para extraer los valores de "id" y "name" de la columna "production_companies". Estos valores se insertan en la tabla temporal "production_companieTem".

Después de recorrer todos los registros en la columna "production_companies", se cierra el cursor y se insertan los valores distintos en la tabla "production_companie". Finalmente, se elimina la tabla temporal "production_companieTem".

Tabla Production_countrie

Figura 14

Procedimiento que permite extraer y Limpiar la tabla Production_countrie

```
DROP PROCEDURE IF EXISTS TablaContries;
DELIMITER $$
CREATE PROCEDURE TablaContries ()
BEGIN
 DECLARE done INT DEFAULT FALSE ;
 DECLARE jsonData json ;
 DECLARE jsonId varchar(250) ;
 DECLARE jsonLabel varchar(250) ;
 DECLARE resultSTR LONGTEXT DEFAULT '';
 DECLARE i INT;
 -- Declarar el cursor
 DECLARE myCursor
  CURSOR FOR
   SELECT JSON_EXTRACT(CONVERT(production_countries USING UTF8MB4), '$[*]') FROM movie_dataset ;
```

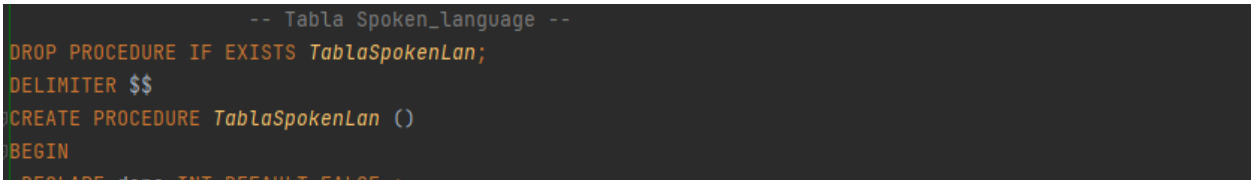
```
-- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
DECLARE CONTINUE HANDLER
 FOR NOT FOUND SET done = TRUE ;
-- Abrir el cursor
OPEN myCursor  ;
drop table if exists production_countriesTem;
   SET @sql_text = 'CREATE TABLE production_countriesTem ( iso_3166_1 varchar(2),
   nameCountrie VARCHAR(100));';
   PREPARE stmt FROM @sql_text;
   EXECUTE stmt;
```

Este procedimiento sigue la misma estructura de Production_countries ya que simplemente se esta cambiando variables.

Tabla Spoken_languge

Figura 15

Procedimiento que permite extraer y Limpiar la tabla Spoken_language



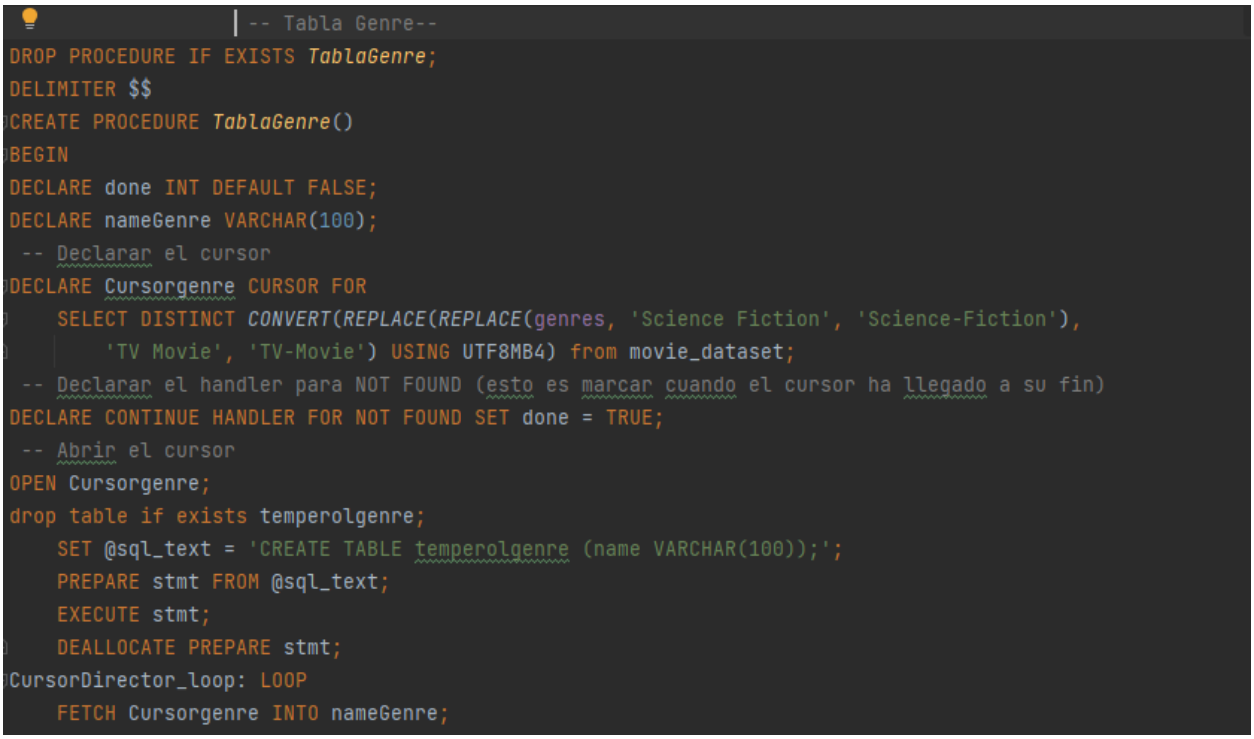
```
    DEALLOCATE PREPARE stmt;
  END WHILE;
 END LOOP ;
 select distinct * from production_LanguageTem;
    INSERT INTO spoken_language
    SELECT DISTINCT iso_639_1, nameLanguage
    FROM production_LanguageTem;
    drop table if exists production_LanguageTem;
 CLOSE myCursor ;
END$$
DELIMITER ;
```

Este procedimiento sigue la misma estructura de Production_countries ya que simplemente se está cambiando variables.

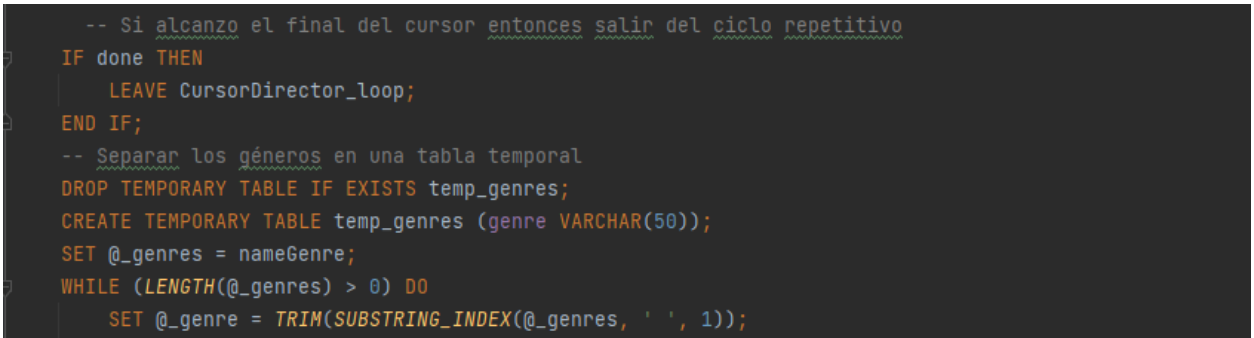
Tabla Genre

Figura 16

Procedimiento que permite extraer y Limpiar la tabla genre



```
-- 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 movie_dataset;
 -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
 -- Abrir el cursor
OPEN Cursorgenre;
drop table if exists temperolgenre;
    SET @sql_text = 'CREATE TABLE temperolgenre (name VARCHAR(100));';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
CursorDirector_loop: LOOP
    FETCH Cursorgenre INTO nameGenre;
```



```
      -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE CursorDirector_loop;
    END IF;
    -- Separar los géneros en una tabla temporal
    DROP TEMPORARY TABLE IF EXISTS temp_genres;
    CREATE TEMPORARY TABLE temp_genres (genre VARCHAR(50));
    SET @_genres = nameGenre;
    WHILE (LENGTH(@_genres) > 0) DO
        SET @_genre = TRIM(SUBSTRING_INDEX(@_genres, ' ', 1));
```

El procedimiento comienza verificando si la tabla "TablaGenre" ya existe y, en caso afirmativo, la elimina. Luego, se establece el delimitador para separar el código SQL y el procedimiento almacenado.

A continuación, se define el procedimiento TablaGenre(). Dentro del procedimiento, se declaran una variable "done" para marcar cuando el cursor haya llegado a su fin, una variable "nameGenre" para almacenar el nombre del género y un cursor "Cursorgenre" para recuperar los géneros distintos del conjunto de datos de películas.

El cursor se abre y se crea una tabla temporal "temperolgenre" para almacenar temporalmente los géneros. Luego, se inicia un ciclo repetitivo "CursorDirector_loop" que recupera cada nombre de género del cursor y lo procesa. Si el cursor ha alcanzado el final, se sale del ciclo repetitivo.

Dentro del ciclo repetitivo, se crea una tabla temporal "temp_genres" para separar los géneros en filas individuales y almacenarlos. Luego, se insertan los géneros separados en la tabla temporal "temperolgenre".

Finalmente, se cierra el cursor y se insertan los géneros distintos en la tabla "genre". La tabla temporal "temperolgenre" se elimina después de su uso.

Para ejecutar el procedimiento, se usa la instrucción "CALL TablaGenre();".

Tabla Person

Figura 17

Procedimiento que permite extraer y Limpiar la tabla person

```
                    -- Tabla Person --
DROP PROCEDURE IF EXISTS TablaPerson;
DELIMITER $$
CREATE PROCEDURE TablaPerson ()
BEGIN
 DECLARE done INT DEFAULT FALSE ;
 DECLARE jsonData json ;
 DECLARE jsonId INT ;
 DECLARE jsonLabel varchar(250) ;
 DECLARE resultSTR LONGTEXT DEFAULT '';
 DECLARE jsongenre VARCHAR(250);
 DECLARE i INT;
 -- Declarar el cursor
 DECLARE CursorPerson
  CURSOR FOR
   SELECT JSON_EXTRACT(CONVERT(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE
       (REPLACE(crew, '"', '\''), '{\'', '{"'),
    '\': \'', '": "'),'\', \'', '", "'),'\': ', '": '),', \'', ', "')
    USING UTF8mb4 ), '$[*]') FROM movie_dataset;
 -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
 DECLARE CONTINUE HANDLER
  FOR NOT FOUND SET done = TRUE ;
 -- Abrir el cursor
 OPEN CursorPerson  ;
 drop table if exists personTem;
    SET @sql_text = 'CREATE TABLE personTem ( idcrew int,
    name VARCHAR(100),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 ;
```

El procedimiento se llama "TablaPerson", Las sentencias DECLARE se usan para declarar variables y el cursor "CursorPerson". La función JSON_EXTRACT se utiliza para extraer información de un objeto JSON.

El cuerpo del procedimiento comienza con un bucle que se ejecutará hasta que se hayan procesado todas las filas de la tabla "movie_dataset".

Dentro del bucle, se utiliza la función FETCH para recuperar una fila de la tabla "movie_dataset" en una variable llamada "jsonData". Luego, hay otro bucle que se ejecutará hasta que se hayan procesado todos los arrays dentro de "jsonData".

Dentro de este bucle, se utiliza la función JSON_EXTRACT para extraer el id, nombre y género de cada objeto JSON en el array. Esta información se asigna a variables locales llamadas "jsonId", "jsonLabel" y "jsongenre", respectivamente.

Luego, se crea una tabla temporal llamada "personTem" que almacenará esta información extraída. La información se inserta en esta tabla temporal mediante una sentencia preparada y una sentencia EXECUTE.

Finalmente, el cursor se cierra y el procedimiento termina. La información almacenada en la tabla temporal se inserta en la tabla "person" con una sentencia INSERT.

Tabla Companie_Movie

Figura 18

Procedimiento que permite extraer y Limpiar la tabla Companie_movie

```
DROP PROCEDURE IF EXISTS TablaCompaMov;
DELIMITER $$
CREATE PROCEDURE TablaCompaMov ()
BEGIN
 DECLARE done INT DEFAULT FALSE;
 DECLARE idMovie int;
 DECLARE idProdComp JSON;
 DECLARE idJSON text;
 DECLARE i INT;
 -- Declarar el cursor
 DECLARE myCursor
  CURSOR FOR
   SELECT id, production_companies FROM movie_dataset;
 -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
 DECLARE CONTINUE HANDLER
  FOR NOT FOUND SET done = TRUE ;
 -- Abrir el cursor
 OPEN myCursor  ;
 drop table if exists CompaMovTem;
```

```
    SET @sql_text = 'CREATE TABLE CompaMovTem ( 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 CompaMovTem VALUES (', idMovie, ', ',
      REPLACE(idJSON,'\'',''), '); ');
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
  END WHILE;
 END LOOP ;
 select distinct * from CompaMovTem;
    INSERT INTO companie_movie
    SELECT DISTINCT id, idGenre
    FROM CompaMovTem;
     DROP TABLE CompaMovTem;
 CLOSE myCursor ;
END$$
DELIMITER ;

-- CALL TablaCompaMov ();
```

El procedimiento comienza con la creación de un cursor llamado "myCursor" que selecciona las columnas "id" y "production_companies" de la tabla "movie_dataset". Luego, se crea una tabla temporal llamada "CompaMovTem" que almacenará los datos de salida de la consulta.

Luego, el procedimiento entra en un bucle "cursorLoop" que se ejecutará mientras haya filas disponibles en el cursor. En cada iteración, se extrae la siguiente fila del cursor y se almacena en las variables "idMovie" y "idProdComp".

La columna "idProdComp" contiene información en formato JSON, por lo que se utiliza el comando "JSON_EXTRACT" para extraer los valores de cada uno de los elementos del array. El valor extraído se almacena en la variable "idJSON". Luego, se construye una consulta SQL dinámica utilizando la función "CONCAT" para insertar los valores en la tabla temporal "CompaMovTem".

Una vez que se han extraído todos los datos de todas las filas, se cierra el cursor y se realiza una consulta SELECT DISTINCT para obtener valores únicos de la tabla temporal y almacenarlos en la tabla "companie_movie". Finalmente, se elimina la tabla temporal.

El procedimiento se puede ejecutar llamándolo en la consola de MySQL o desde un script, y se garantiza que se elimine antes de ser creado en caso de que ya exista.

Tabla CountriesMovies

Figura 19

Procedimiento que permite extraer y Limpiar la tabla CountriesMovies

```
DROP PROCEDURE IF EXISTS TablaCounMov;
DELIMITER $$
CREATE PROCEDURE TablaCounMov ()
BEGIN
 DECLARE done INT DEFAULT FALSE;
 DECLARE idMovie int;
 DECLARE idProdCoun text;
 DECLARE idJSON text;
 DECLARE i INT;
 -- Declarar el cursor
 DECLARE myCursor
  CURSOR FOR
   SELECT id, production_countries FROM movie_dataset;
 -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
 DECLARE CONTINUE HANDLER
  FOR NOT FOUND SET done = TRUE ;
 -- Abrir el cursor
 OPEN myCursor  ;
 drop table if exists MovCounTemp;
    SET @sql_text = 'CREATE TABLE MovCounTemp ( id int, idGenre varchar(255) );';
    PREPARE stmt FROM @sql_text;
```

```
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
cursorLoop: LOOP
    FETCH myCursor INTO idMovie, idProdCoun;
  -- Controlador para buscar cada uno de los arrays
    SET i = 0;
  -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
  IF done THEN
   LEAVE  cursorLoop ;
  END IF ;
  WHILE(JSON_EXTRACT(idProdCoun, CONCAT('$[', i, '].iso_3166_1')) IS NOT NULL) DO
  SET idJSON = JSON_EXTRACT(idProdCoun,  CONCAT('$[', i, '].iso_3166_1')) ;
  SET i = i + 1;
  SET @sql_text = CONCAT('INSERT INTO MovCounTemp VALUES (', idMovie, ', ',
      REPLACE(idJSON,'\'',''), '); ');
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
  END WHILE;
 END LOOP ;
 select distinct * from MovCounTemp;
 select distinct * from MovCounTemp;
    INSERT INTO countrie_movie
    SELECT DISTINCT  idGenre,id
    FROM MovCounTemp;
    DROP TABLE MovCounTemp;
 CLOSE myCursor ;
END$$
DELIMITER ;
-- call TablaCounMov();
```

Se utiliza el mismo Proceso anterior solo se van cambiando variables.

Tabla LanguageMovie

Figura 19

Procedimiento que permite extraer y Limpiar la tabla LanguageMovie

```
DROP PROCEDURE IF EXISTS LanguageMovie;
DELIMITER $$
CREATE PROCEDURE LanguageMovie ()
BEGIN
 DECLARE done INT DEFAULT FALSE;
 DECLARE idMovie int;
 DECLARE idSpokLang text;
 DECLARE idJSON text;
 DECLARE i INT;
 -- Declarar el cursor
 DECLARE myCursor
  CURSOR FOR
```

```
  SELECT id, spoken_languages FROM movie_dataset;
 -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
DECLARE CONTINUE HANDLER
 FOR NOT FOUND SET done = TRUE ;
 -- Abrir el cursor
OPEN myCursor  ;
cursorLoop: LOOP
    FETCH myCursor INTO idMovie, idSpokLang;
 -- Controlador para buscar cada uno de los arrays
   SET i = 0;
 -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
 IF done THEN
  LEAVE  cursorLoop ;
 END IF ;
 WHILE(JSON_EXTRACT(idSpokLang, CONCAT('$[', i, '].iso_639_1')) IS NOT NULL) DO
 SET idJSON = JSON_EXTRACT(idSpokLang,  CONCAT('$[', i, '].iso_639_1')) ;
 SET i = i + 1;
 SET @sql_text = CONCAT('INSERT INTO lenguage_movie VALUES (', idMovie, ', ',
    REPLACE(idJSON,'\'',''), '); ');
   PREPARE stmt FROM @sql_text;
   EXECUTE stmt;
   DEALLOCATE PREPARE stmt;
 END WHILE;
END LOOP ;
CLOSE myCursor ;
END$$
DELIMITER ;
-- call LanguageMovie();
```

Tabla Director

Figura 20

Procedimiento que permite extraer y Limpiar la tabla director

```
-- Tabla Directior ---

DROP PROCEDURE IF EXISTS Director;
DELIMITER $$
CREATE PROCEDURE Director()
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 personTem;
     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 idPerson INTO idPersonas FROM person WHERE person.name=MovDirector;
     INSERT INTO directorTemp VALUES (idPersonas,Movid);
     END LOOP;
 CLOSE CursorDirector;
 select distinct * from directorTemp;
 INSERT  INTO director
     SELECT DISTINCT  idPer,idMov
     FROM directorTemp;
 drop table if exists directorTemp;
 END $$
 DELIMITER ;
 -- CALL Director();
```

Tabla Crew

Figura 21

Procedimiento que permite extraer y Limpiar la tabla crew

```
-- 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('$[', i, '].id')) IS NOT NULL) DO
 SET jobJSON = JSON_EXTRACT(idCrew,  CONCAT('$[', i, '].job')) ;
 SET idJSON = JSON_EXTRACT(idCrew,  CONCAT('$[', i, '].id')) ;
 SET departmentJSON = JSON_EXTRACT(idCrew,  CONCAT('$[', i, '].department')) ;
 SET credit_idJSON = JSON_EXTRACT(idCrew,  CONCAT('$[', i, '].credit_id')) ;
 SET i = i + 1;
 SET @sql_text = CONCAT('INSERT Ignore INTO Crew VALUES (', idMovie, ', ',
     REPLACE(idJSON,'\'',''), ', ', REPLACE(idJSON,'\'',''), ', ',
     REPLACE(departmentJSON,'\'',''), ', ', REPLACE(credit_idJSON,'\'',''), '); ');
   PREPARE stmt FROM @sql_text;
   EXECUTE stmt;
   DEALLOCATE PREPARE stmt;
 END WHILE;
END LOOP ;
CLOSE myCursor ;
END$$
DELIMITER ;
call TablaCrew();  5 m 36 s
```

El procedimiento tiene como objetivo insertar datos en una tabla llamada "Crew" desde una tabla llamada "movie_dataset".

Antes de crear el procedimiento, se verifica si ya existe un procedimiento con el mismo nombre con la sentencia "DROP PROCEDURE IF EXISTS TablaCrew;". Luego, se cambia el delimitador a "$$" para que MySQL pueda procesar correctamente el código que está dentro de la definición del procedimiento.

El procedimiento comienza declarando algunas variables: "idMovie", "idCrew", "idJSON", "jobJSON", "departmentJSON" y "credit_idJSON", todas ellas son utilizadas para almacenar valores temporales durante la ejecución del procedimiento.

Luego, se define un cursor llamado "myCursor" que selecciona todos los registros de la tabla "movie_dataset" y convierte una columna "crew" en un formato que sea compatible con JSON.

Se define un handler llamado "CONTINUE HANDLER" que se ejecutará cuando el cursor llegue al final de los registros y marcará la variable "done" como "TRUE".

El cursor se abre y se inicia un loop infinito que se repetirá hasta que se alcance el final del cursor o se salga explícitamente del loop. Dentro del loop, se usa la función "FETCH" para obtener un registro a la vez y se guardan los valores en las variables correspondientes.

Se usa un segundo loop "WHILE" para procesar los datos en formato JSON en el campo "idCrew". Se extraen los valores de "job", "id", "department" y "credit_id" utilizando la función "JSON_EXTRACT" y se guardan en las variables correspondientes.

Finalmente, se construye una sentencia SQL dinámica usando la función "CONCAT" y se ejecuta usando las funciones "PREPARE", "EXECUTE" y "DEALLOCATE PREPARE".

Una vez que se han procesado todos los registros, se cierra el cursor y se finaliza el procedimiento. Para ejecutar el procedimiento, se puede llamar "call TablaCrew();".

Conclusiones

En este proyecto, se siguieron las normas de modelado de base de datos, como la normalización y el diseño conceptual y lógico, y se lograron cumplir con los objetivos establecidos al inicio. A pesar de los desafíos que surgieron durante el proceso, se superaron con la ayuda de los docentes y la creatividad del equipo. Al finalizar, se pudo apreciar un gran progreso en la toma de decisiones en cuanto a la modelación y población de una base de datos, y se adquirieron nuevas habilidades en programación para casos de la vida real. Este proyecto nos acercó más a ser profesionales confiables en futuros proyectos.

Se aprendió la importancia de tratar los datos con precisión antes de procesarlos para evitar errores y consistencia. También se comprendió la importancia de las herramientas para interactuar con la base de datos. En resumen, todo lo aprendido en el ciclo de estudios fue crucial para el éxito de este proyecto integrador, que fue un reto para los estudiantes debido a la cantidad de datos con los que se trabajó por primera vez.