
# UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

*La Universidad Católica de Loja*

## Documento de Informe

## Fundamentos de Bases de Datos

**Autor**:

- Flores Valdiviezo Orly André

## Contenido

# 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.

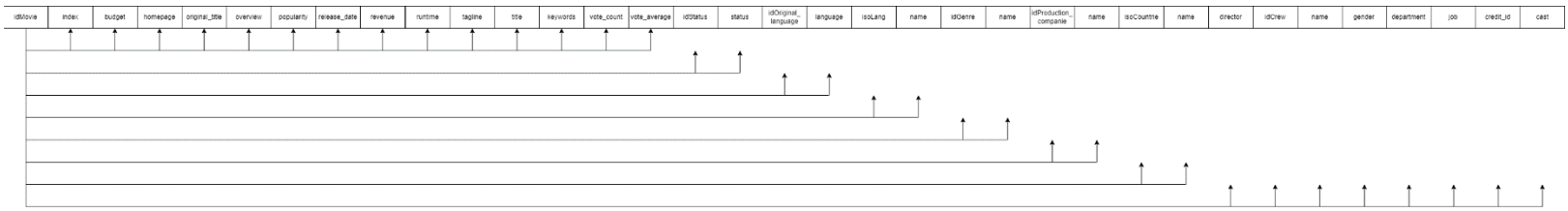


*Figura 1: La imagen muestra una descripción gráfica de las dependencias existentes en las columnas del dataset*

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}

*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.

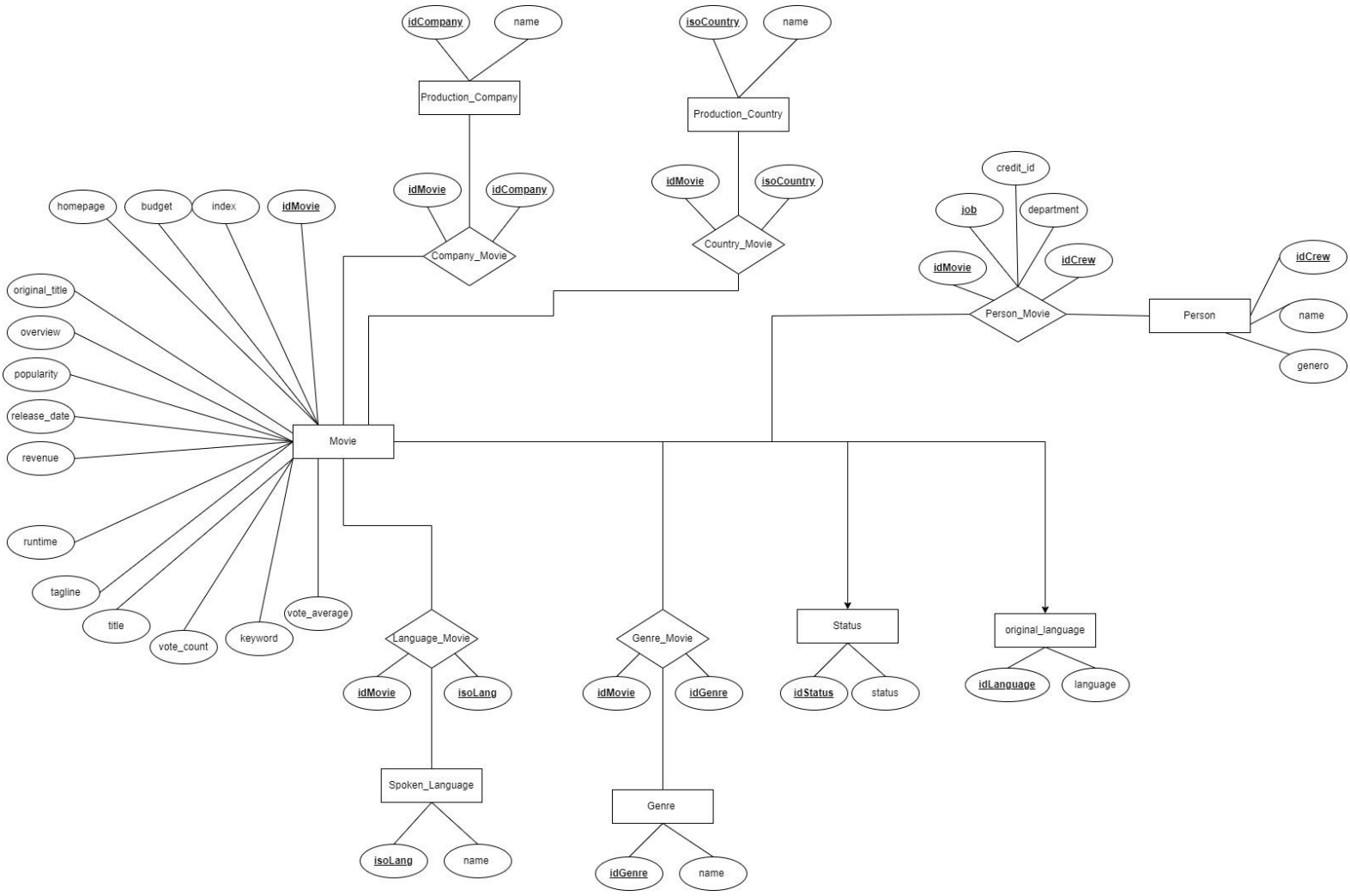


*Figura 3: Representación del Modelo Conceptual*

## 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.

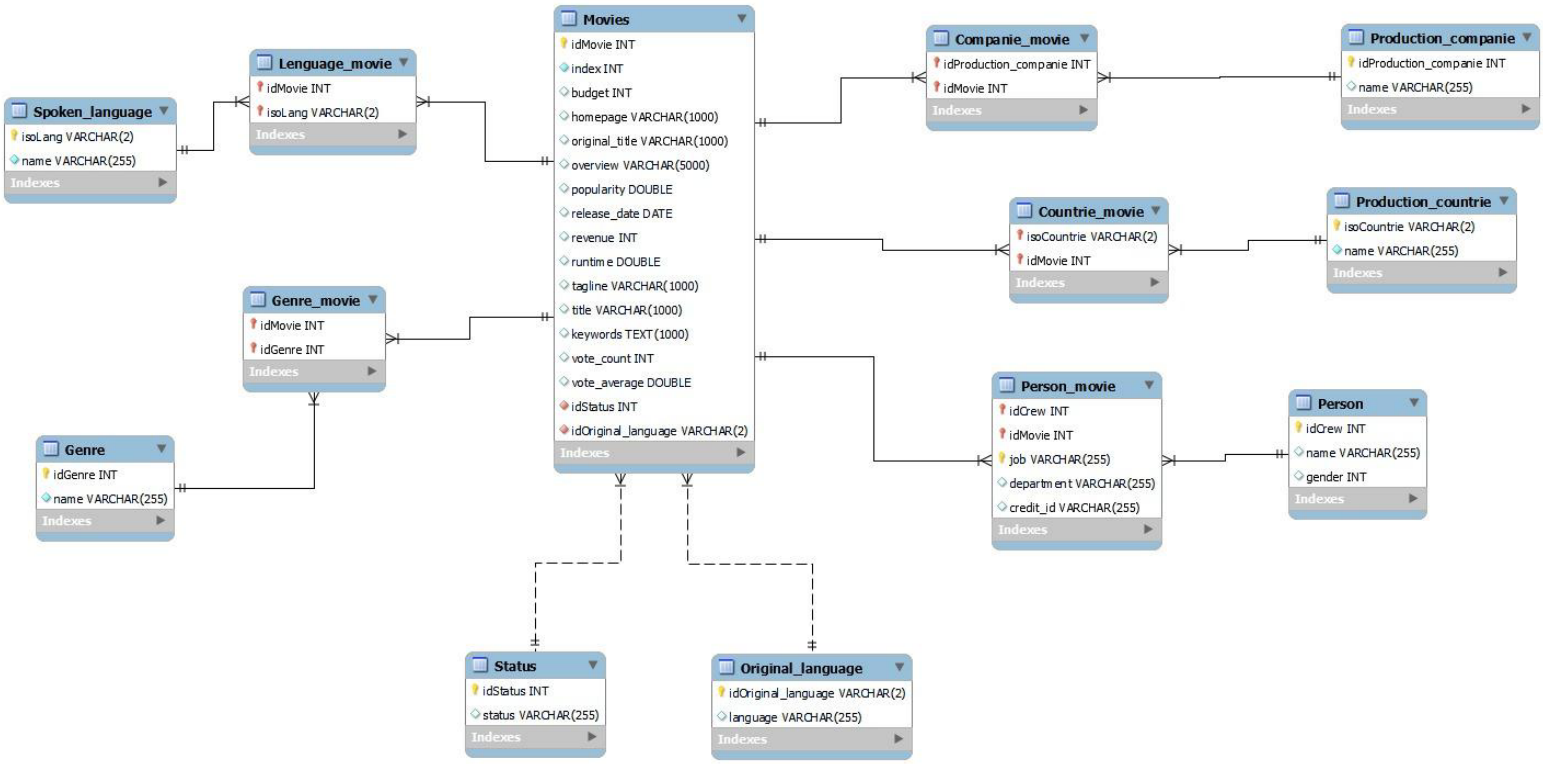


*Figura 4: Representación gráfica del Modelo Lógico*

## 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.

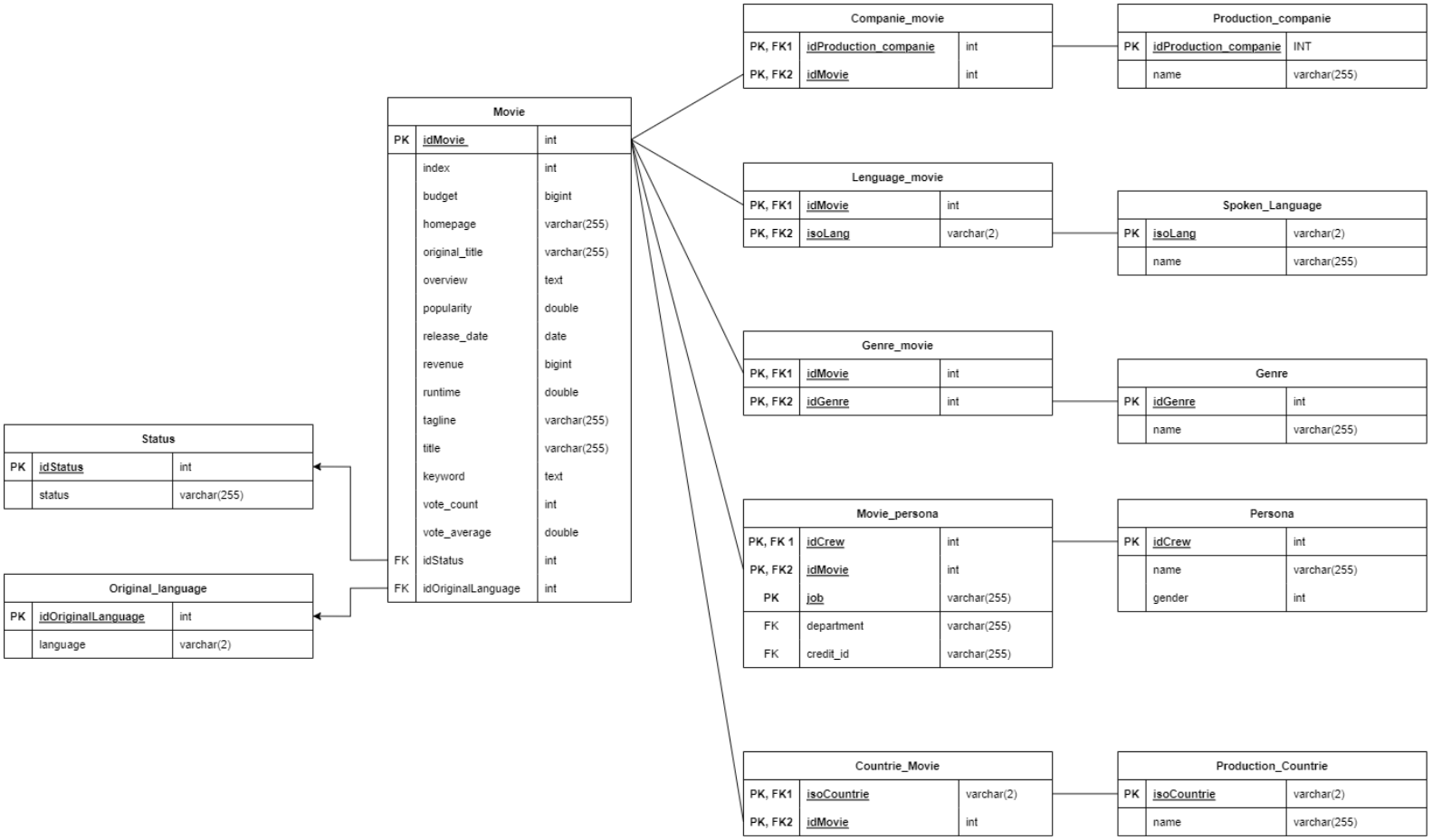


*Figura 5: Representación gráfica del Modelo Físico*

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:

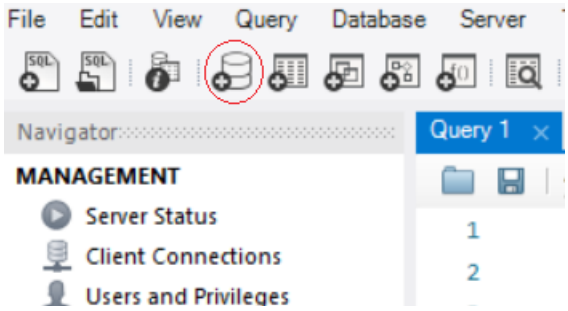


*Figura 6: Representación de la barra de opciones perteneciente a Workbench*

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":

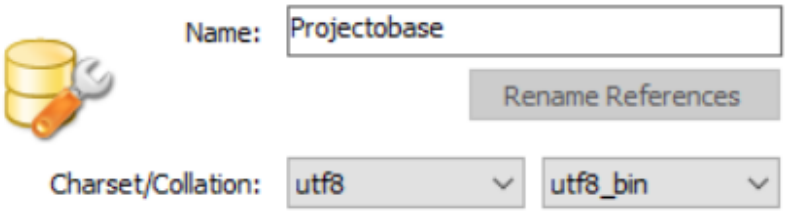


*Figura 7: Representación de la funcionalidad de creación de Schema automáticamente*

**OPCIÓN 2 – MEDIANTE SENTENCIAS SQL**

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

```
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:

```
USE `Projectobase`;
```

## 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:

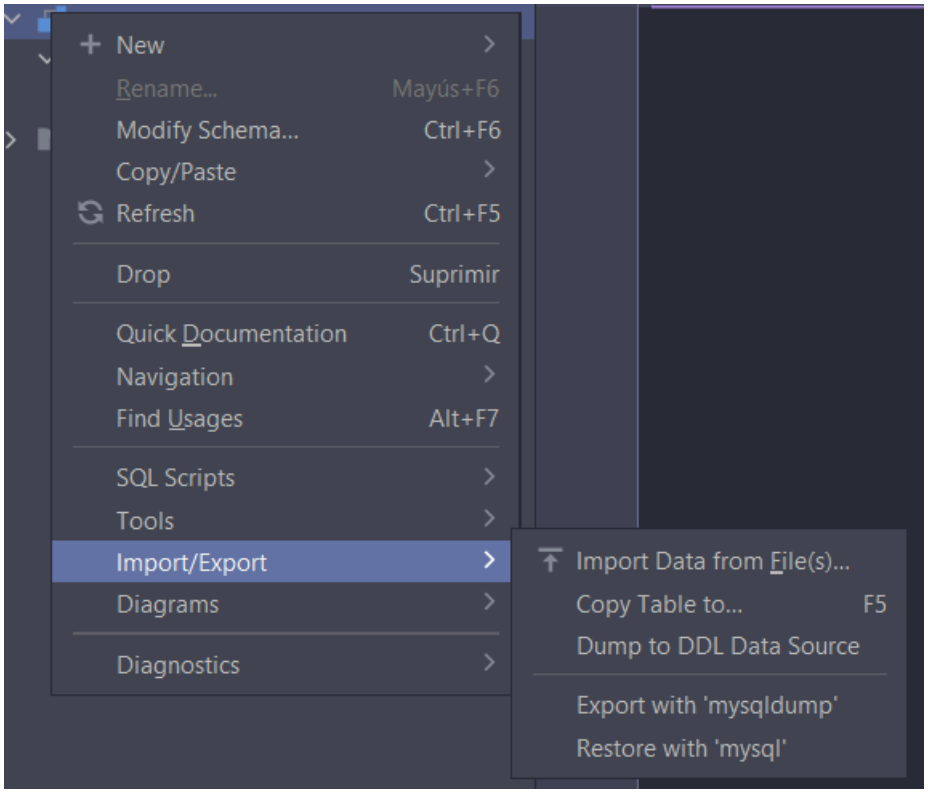


*Figura 8: Representación de la funcionalidad de importación de archivos en DataGrip*

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

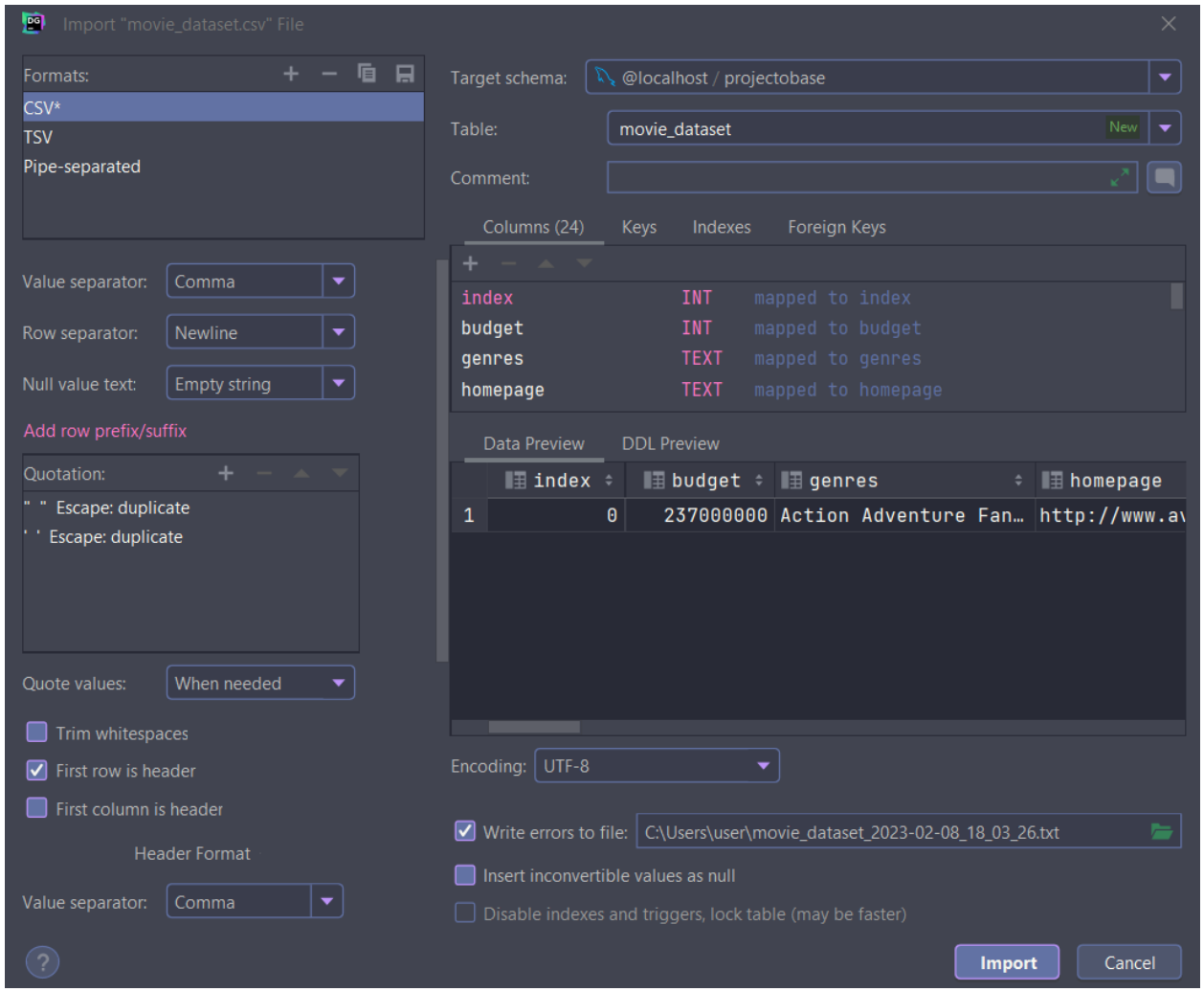


*Figura 8: Representación de la interfaz de importación de archivos en DataGrip*

## 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"**

```
DROP PROCEDURE IF EXISTS TablaOriginal_language;
DELIMITER $$
CREATE PROCEDURE TablaOriginal_language()
BEGIN
    DECLARE done INT DEFAULT FALSE;
    DECLARE namelanguage VARCHAR(100);
    -- Declarar el cursor
    DECLARE Cursorlanguage CURSOR FOR
        SELECT DISTINCT CONVERT(original_language 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 Cursorlanguage;
    CursorDirector_loop: LOOP
        FETCH Cursorlanguage INTO namelanguage;
        -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
        IF done THEN
            LEAVE CursorDirector_loop;
        END IF;
```

```
        IF namelanguage IS NULL THEN
            SET namelanguage = '';
        END IF;
        SET @_oStatement = CONCAT('INSERT INTO original_language (language) VALUES (\'',namelanguage,'\');');
        PREPARE sent1 FROM @_oStatement;
        EXECUTE sent1;
        DEALLOCATE PREPARE sent1;

    END LOOP;
    CLOSE Cursorlanguage;
END $$
DELIMITER ;

CALL TablaOriginal_language ();
```

**Población 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 ;
```

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.

```
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 ;
```

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”**

```
-- Tabla Companie------
DROP PROCEDURE IF EXISTS TablaCompanie ;
DELIMITER $$
CREATE PROCEDURE TablaCompanie ()
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_companies 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_companietem;
SET @sql_text = 'CREATE TABLE production_companieTem ( id int, nameCom VARCHAR(100));';
PREPARE stmt FROM @sql_text;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
cursorLoop: LOOP
    FETCH myCursor INTO jsonData;
    -- Controlador para buscar cada uno de lso arrays
    SET i = 0;
    -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE  cursorLoop ;
    END IF ;
    WHILE(JSON_EXTRACT(jsonData, CONCAT('$[', i, ']')) IS NOT NULL) DO
            SET jsonId = IFNULL(JSON_EXTRACT(jsonData,  CONCAT('$[', i, '].id')), '') ;
            SET jsonLabel = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i,'].name')), '') ;
            SET i = i + 1;
            SET @sql_text = CONCAT('INSERT INTO production_companieTem VALUES (', REPLACE(jsonId,'\'',''), ', ', jsonLabel, '); ');
            PREPARE stmt FROM @sql_text;
            EXECUTE stmt;
            DEALLOCATE PREPARE stmt;
        END WHILE;
END LOOP ;
    select distinct * from production_companieTem;
    INSERT INTO production_companie
    SELECT DISTINCT id, nameCom
    FROM production_companieTem;
    drop table if exists production_companieTem;
    CLOSE myCursor ;
END$$
DELIMITER ;
```

**Población Tabla “Production_countries”**

```
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;
DEALLOCATE PREPARE stmt;
cursorLoop: LOOP
    FETCH myCursor INTO jsonData;

    -- Controlador para buscar cada uno de los arrays
    SET i = 0;

    -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE  cursorLoop ;
    END IF ;
```

```
        WHILE(JSON_EXTRACT(jsonData, CONCAT('$[', i, ']')) IS NOT NULL) DO
                SET jsonId = IFNULL(JSON_EXTRACT(jsonData,  CONCAT('$[', i, '].iso_3166_1')), '') ;
                SET jsonLabel = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i,'].name')), '') ;
                SET i = i + 1;

                SET @sql_text = CONCAT('INSERT INTO production_countriesTem VALUES (', REPLACE(jsonId,'\'',''), ', ', jsonLabel, '); ');
                PREPARE stmt FROM @sql_text;
                EXECUTE stmt;
                DEALLOCATE PREPARE stmt;
            END WHILE;
    END LOOP ;
    select distinct * from production_countriesTem;
    INSERT INTO production_countrie
    SELECT DISTINCT iso_3166_1, nameCountrie
    FROM production_countriesTem;
    drop table if exists production_countriesTem;
    CLOSE myCursor ;
END$$
DELIMITER ;
```

## Población Tabla “Spoken_languages”

```
DROP PROCEDURE IF EXISTS TablaSpokenLen;
DELIMITER $$
CREATE PROCEDURE TablaSpokenLen ()
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(spoken_languages 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_LanguageTem;
SET @sql_text = 'CREATE TABLE production_LanguageTem ( iso_639_1 varchar(2), nameLanguage VARCHAR(100));';
PREPARE stmt FROM @sql_text;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
cursorLoop: LOOP
    FETCH myCursor INTO jsonData;

    -- Controlador para buscar cada uno de los arrays
    SET i = 0;

    -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE  cursorLoop ;
    END IF ;
```

```
    WHILE(JSON_EXTRACT(jsonData, CONCAT('$[', i, ']')) IS NOT NULL) DO
            SET jsonId = IFNULL(JSON_EXTRACT(jsonData,  CONCAT('$[', i, '].iso_639_1')), '') ;
            SET jsonLabel = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i,'].name')), '') ;
            SET i = i + 1;

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

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

**Población 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));
                INSERT INTO temp_genres (genre) VALUES (@_genre);
                SET @_genres = SUBSTRING(@_genres, LENGTH(@_genre) + 2);
            END WHILE;
        -- Insertar los géneros separados en filas individuales
        INSERT INTO temperolgenre (name)
        SELECT genre FROM temp_genres;
    END LOOP CursorDirector_loop;
    select distinct * from temperolgenre;
    INSERT INTO genre (name)
    SELECT DISTINCT name
    FROM temperolgenre;
    drop table if exists temperolgenre;
    CLOSE Cursorgenre;
END $$
DELIMITER ;

CALL TablaGenre();
```

**Población 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;
    CLOSE myCursor ;
END$$
DELIMITER ;

CALL TablaCompaMov ();
```

**Población Tabla “Countrie_movie”**

```sql
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;
    INSERT INTO countrie_movie
    SELECT DISTINCT  idGenre,id
    FROM MovCounTemp;
    CLOSE myCursor ;
END$$
DELIMITER ;

CALL TablaCounMov();
```

**Población Tabla “Lenguage_movie”**

```sql
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 ;
```

```sql
    -- 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();
```

## 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.

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

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.

```sql
SELECT Spoken_language.`name`, COUNT(*) as NumeroApariciones
FROM `Spoken_language`, `Lenguage_movie`
WHERE `Spoken_language`.isoLang = `Lenguage_movie`.isoLang
GROUP BY (`Spoken_language`.`name`)
Order by NumeroApariciones DESC;
```

Output | Result 98

62 rows

| | name | NumeroApariciones |
|---|---|---|
| 1 | English | 4482 |
| 2 | Français | 436 |
| 3 | Español | 351 |
| 4 | Deutsch | 262 |
| 5 | Italiano | 188 |

```sql
SELECT title as 'Título', revenue as 'Ganancias', vote_average as 'Puntaje'
FROM Movies
WHERE revenue >= 1000000
ORDER BY revenue DESC;
```

Output | Result 100

1-500 of 501+

| | Título | Ganancias | Puntaje |
|---|---|---|---|
| 1 | Avatar | 2787965087 | 7.2 |
| 2 | Titanic | 1845034188 | 7.5 |
| 3 | The Avengers | 1519557910 | 7.4 |
| 4 | Jurassic World | 1513528810 | 6.5 |
| 5 | Furious 7 | 1506249360 | 7.3 |

```sql
SELECT Original_language.`language`, COUNT(*) as NumeroApariciones
FROM `Original_language`, `Movies`
WHERE `Original_language`.idOriginal_language = `Movies`.idOriginal_language
GROUP BY (`Original_language`.`language`)
Order by NumeroApariciones DESC;
-- ------------------------------------------------------
```

Output | Result 104

37 rows

| | language | NumeroApariciones |
|---|---|---|
| 1 | en | 4502 |
| 2 | fr | 70 |
| 3 | es | 32 |
| 4 | zh | 27 |
| 5 | de | 27 |

```sql
SELECT title, idMovie
FROM Movies
WHERE title LIKE 'Z%';
-- ------------------------------------------------------
```

Output | projectobase.movies

16 rows | Tx: Auto | DDL

| | title | idMovie |
|---|---|---|
| 1 | Zodiac | 1949 |
| 2 | Zathura: A Space Adventure | 6795 |
| 3 | Zoolander | 9398 |
| 4 | Zack and Miri Make a Porno | 10358 |
| 5 | Zoom | 14113 |

# 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.