UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

Informe Final

Fundamentos de Bases de Datos

Autor: Castillo Paladines Byron Josué

Octubre 2022 – Febrero 2023

Contenido

Introducción

Las bases de datos han demostrado a lo largo del tiempo que son esenciales en muchos sistemas informáticos ya que nos da la opción de ingresar, manipular, almacenar y/o eliminar grandes cantidades de información, este presente informe tiene como objetivo aplicar todos los conocimientos obtenidos en todo este tiempo de estudio con base a la materia de Base de Datos, aplicando la importación del CSV desde una base de datos usando el sistema de gestión de base de datos MySQL, además de pasar por distintas fases, las cuales incluyen la inserción de datos, la limpieza de dichos datos, la carga y sobre todo la explotación de datos. Igualmente, podemos diferenciar los métodos empleados al realizar todas las fases propuestas en este trabajo.

Este informe busca producir resultados orientados a la gestión y uso eficiente de la información relevante y de interés, a partir del conjunto de datos de trabajo. Además, nos permitió recordar varios temas estudiados en la materia tales como diagramas de Entidad/Relación, modelos conceptuales, lógicos y físicos, la creación de instructivos SQL, entre muchos más temas que se muestran a lo largo del informe.

Modelos

Modelo Conceptual

Después de organizar los datos en una tabla universal, los normalizamos haciendo uso del Modelado Entidad/Relación, además nos permite representar gráficamente la estructura de los datos y las relaciones entre ellos tomando en cuenta en como estarían organizados los datos del archivo CSV y las características de cada columna, identificando los atributos correspondientes a cada relación.

La normalización implica separar los datos repetidos y asegurar su integridad mediante creación de tablas individuales para cada entidad y relación.

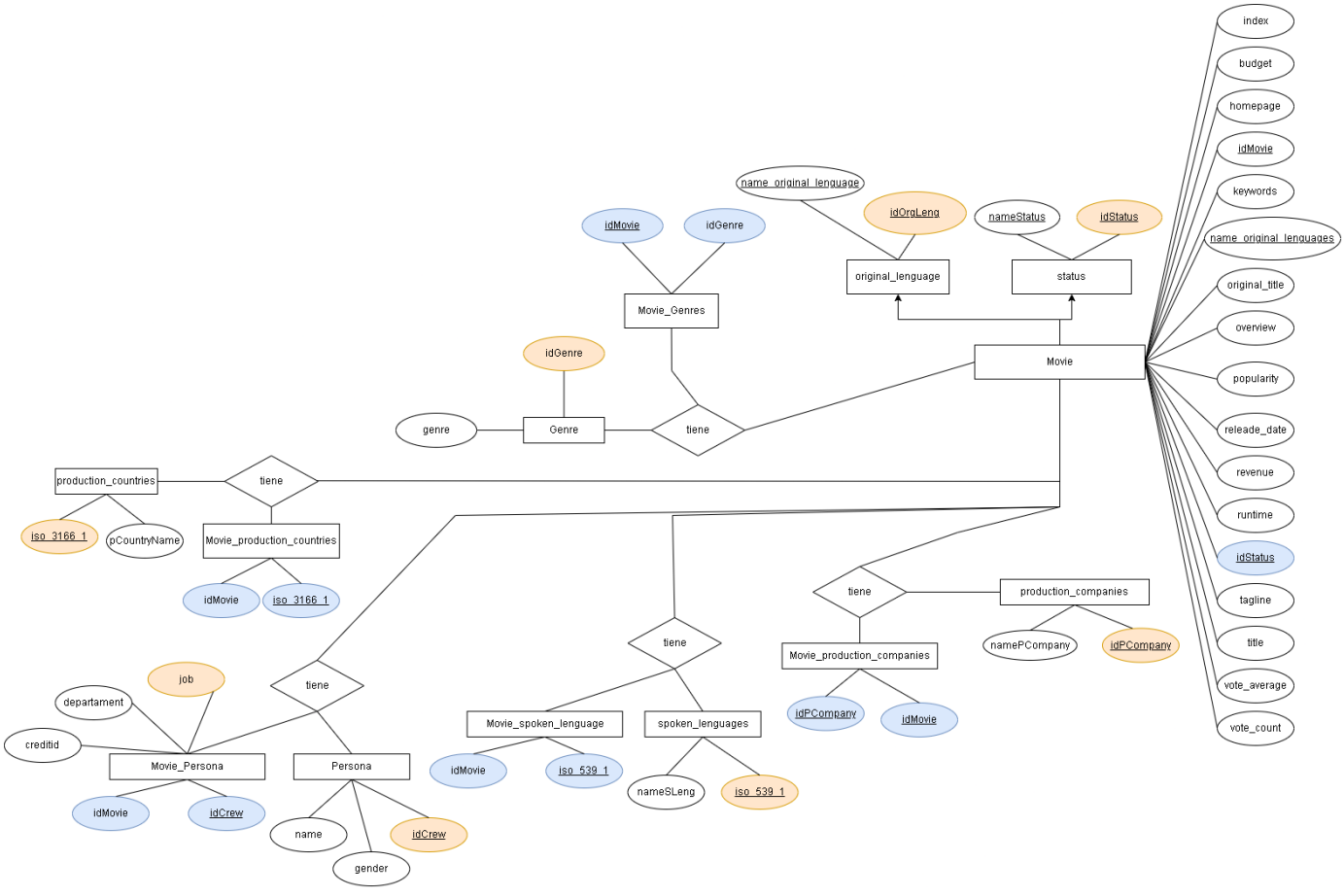


Figura1: Representación del Modelo Conceptual

Modelo Lógico

Permite especificar las relaciones entre las tablas mediante claves foráneas y claves primarias. Una clave primaria es una columna o conjunto de columnas que identifica de manera única a cada registro en una tabla. Por otro lado, una clave foránea se refiere a una columna en una tabla que hace referencia a una clave primaria en otra tabla. De esta forma, el Modelo Lógico SQL permite representar las relaciones entre las entidades de manera clara y estructurada.

Con el modelo conceptual ya realizado, se seleccionan los atributos apropiados para cada tabla y se crea un prototipo de estas basado en las entidades y relaciones identificadas. En cada tabla, se declara una clave primaria y, en su caso, una clave foránea, esto con el objetivo de tener una base sólida para desarrollar el esquema de la base de datos.

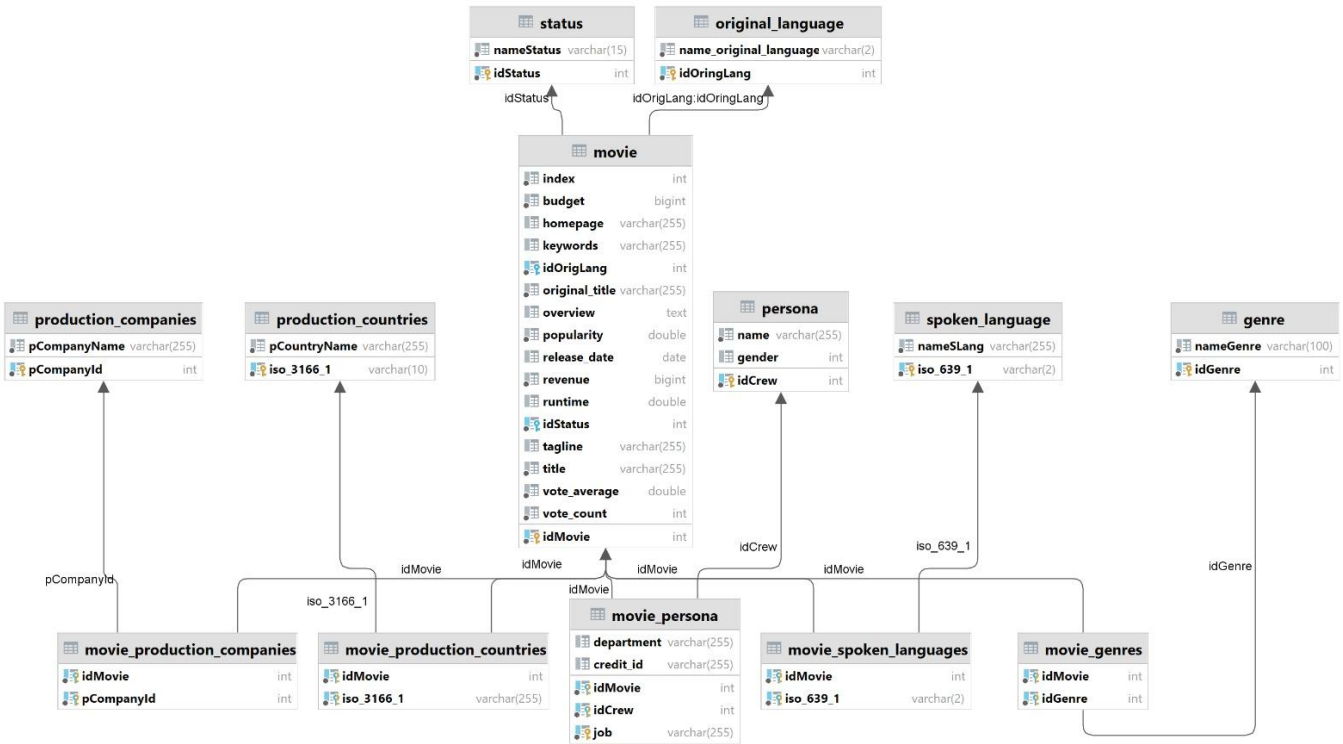
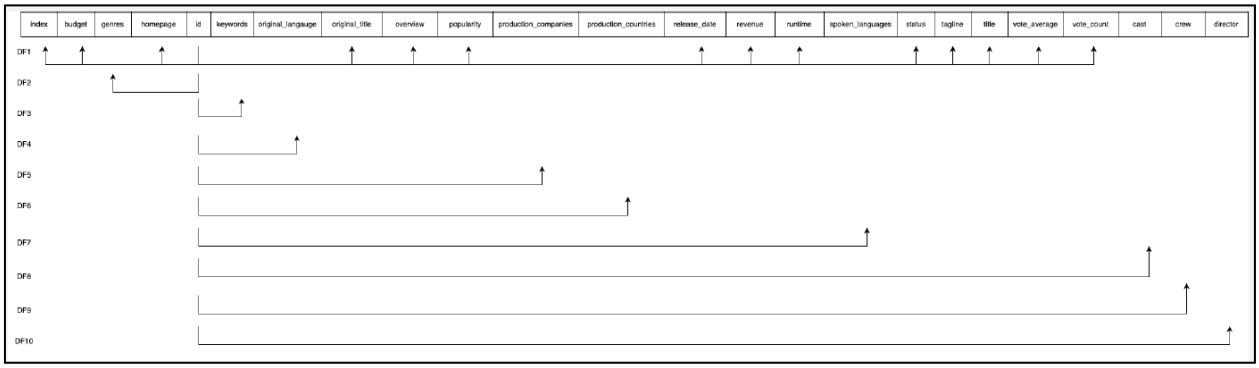


Figura 2: Representación gráfica del Modelo Lógico

Dependencias Funcionales – Tabla Universal



Se refiere a las relaciones que existen entre los atributos de una entidad y como un cambio en un atributo puede afectar a otros atributos, es importante para garantizar la integridad de los datos en una base de datos y para optimizar el rendimiento de las consultas. Una vez establecido el modelo lógico y conceptual se creó las dependencias funcionales existentes en la tabla universal.

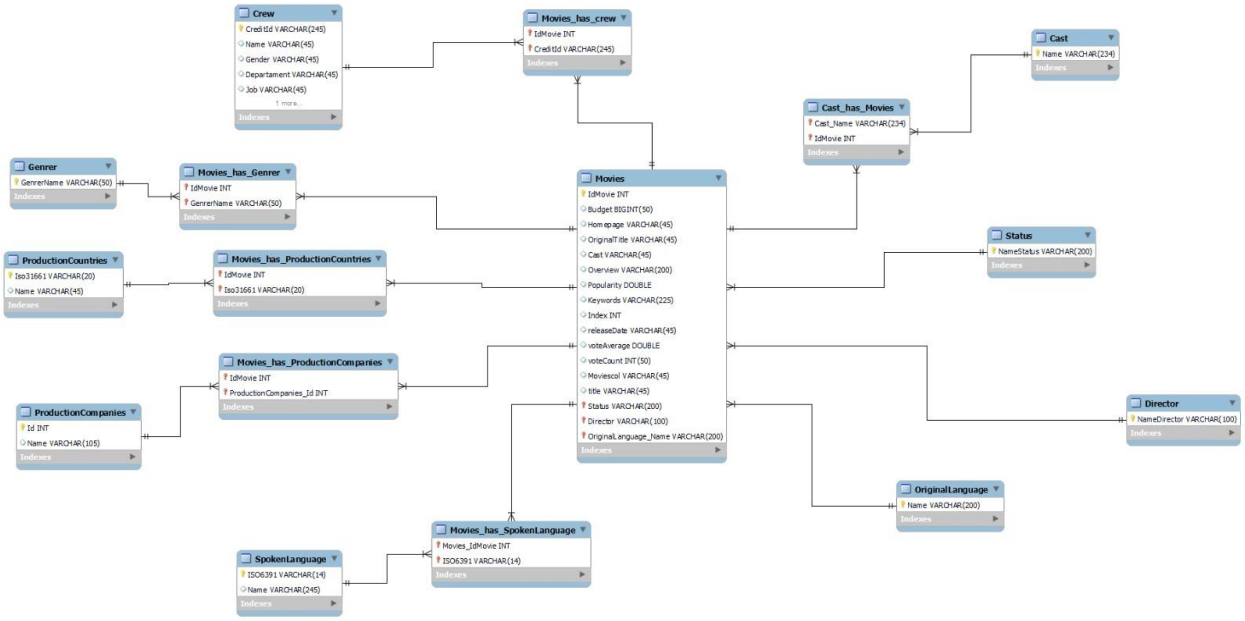
Figura 3: La imagen muestra una descripción gráfica de las dependencias existentes en las columnas del dataset

	id --> { index, budget, homepage, original_language, original_title, overview, popularity, release_date, revenue, runtime, status, tagline, title, vote_average, vote_count }	
	genres --> { id }	
	keywords --> { id }	
	original_language --> { id }	
json	production_companies --> { id }	idProductionCompany --> { id , name}
json	production_countries --> { id }	iso_3611_1 --> { id , name}
json	spoken_languages --> { id }	iso_639_1 --> { id, name }
	cast --> { id }	
json	crew --> { id }	idCrew --> { id , name, gender, department, job, credit_id }
	director --> { id }	

Figura 4: Dependencias Funcionales textualizadas para un mejor entendimiento

Modelo Físico

Este modelo describe cómo se almacenan los datos en el sistema de almacenamiento y cómo se accede a ellos, con este modelo podemos pasar a generar las respectivas sentencias DDL, incluye información detallada sobre la implementación de las tablas, columnas, índices, claves primarias y foráneas, y otras características de la base de datos, tales como la configuración de la memoria caché, el almacenamiento de datos, los tamaños de página, etc.



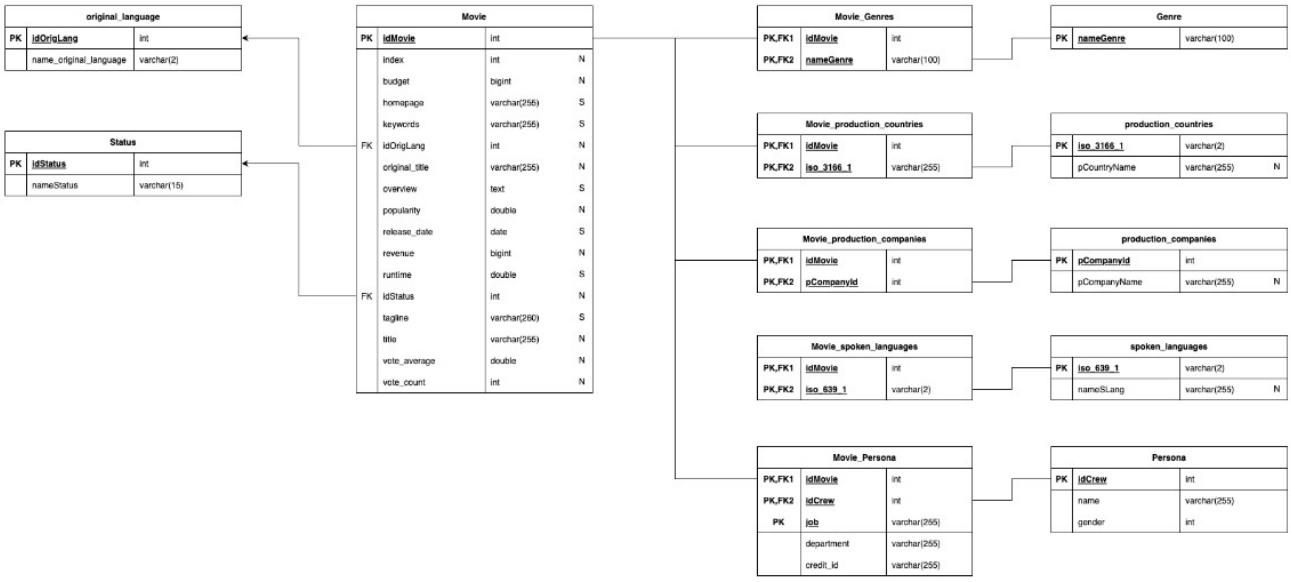


Figura 5: Representación gráfica del Modelo Físico

Normalización

La normalización se la puede describir como un proceso en el cual se organizan los datos por medio de creación de elementos como tablas y relaciones las cuales pueden ser modelos de bases relacionales, tomando como sentido el funcionamiento de las bases de datos relacionales la normalización busca una mejor organización de los datos y a la vez eliminar la redundancia

First Normal Form (1NF)

- Una relación en donde la intersección de cada fila y columna contiene uny solo un valor.
- No existen filas repetidas

- La fase antes de 1NF es Unnormalized Form (UNF) la cual es una tablaque 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.
- Todos los atributos son atómicos, simples e individuales.
- 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.
- Identificar cada conjunto de relacionados con la clave principal.
- Elimina grupos repetidos en tablas individuales.

Second Normal Form (2NF)

- Una relación que está en la primera forma normal y cada atributo que noes de clave principal depende funcionalmente de la clave principal.
- Crear tablas separadas para el conjunto de valores y los registros múltiples, estas tablas se deben relacionar con una clave externa.
- Los registros no deben depender de otra cosa que la clave principal de la tabla, incluida la clave compuesta si es necesario.
- 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)

- Comprueba las dependencias transitivas, eliminando campos que no dependen de la clave principal.
- Una relación que está en primera y segunda forma normal y en la que

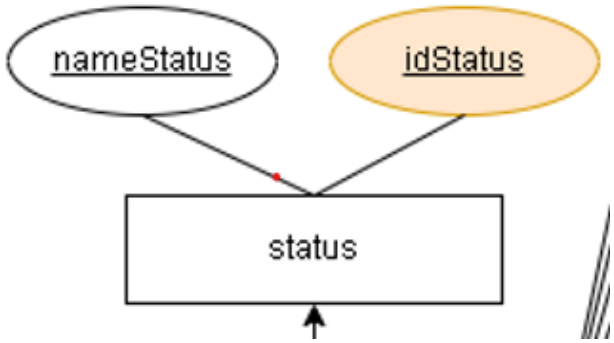
ningún atributo que no sea de clave principal depende transitivamente de la clave principal.

- Los valores que no dependen de la clave principal no pertenecen a la tabla.
- Los campos que no pertenecen a la clave principal colóquelos en una tabla aparte y relacionen ambas tablas por medio de una clave externa.
- La normalización de las relaciones 2NF a 3NF implica la eliminación delas dependencias transitivas.
- Los campos que no pertenecen a la clave principal colóquelos en una tabla aparte y relacionen ambas tablas por medio de una clave externa.

- 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.

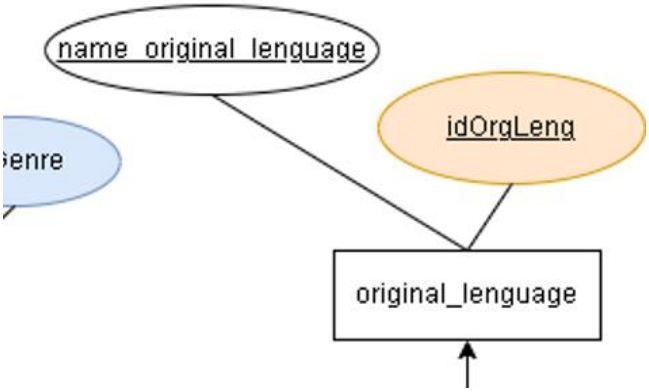
Tabla Uno a Muchos (1 : N)



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

- Una película puede tener un Status, pero un Status puede representar a muchas películas.
- Para solucionar, se crea una tabla separada que se llame Status.
- La llave primaria es nameStatus.
- Como solución, utilizamos el nameStatus como llave foránea en

Original_language es una columna con el nombre del lenguaje original de la Movie.



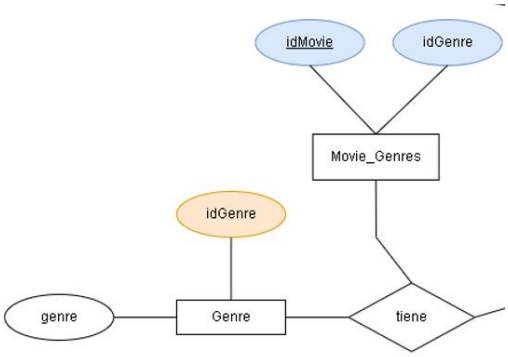
- Una película tiene un original_language, pero un original_language puede tener muchas películas.
- Para solucionar, se crea una tabla separada que se llame original_language.

  La llave primaria es idOrgLeng.
- Como solución, utilizamos el idOrgLeng como llave foránea en original_language.

Tabla Muchos a Muchos (N : N)

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

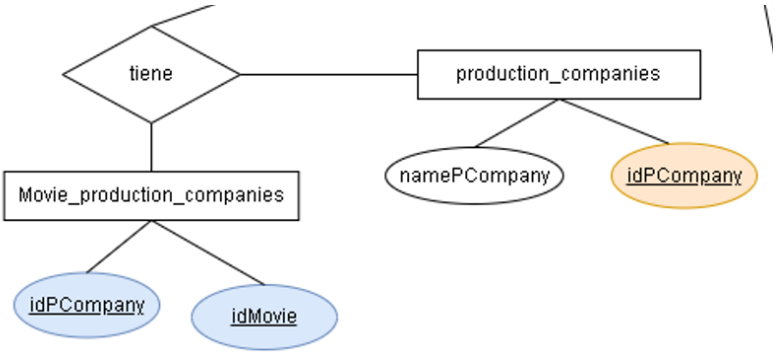
Cada Movie tiene un index y un id diferente. Como llave primaria utilizamos id el cual nombramos idMovie para no confundirlo con otros id que existan en otras columnas.



- Esta columna 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.

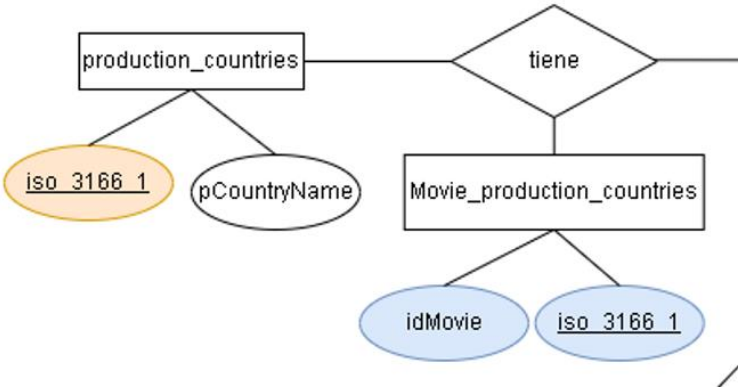
La llave primaria es idGenre

- 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.
  - Las columnas de “Movie_production_companies” y “production_countries” son String de JSON.

- La solución es crear una tabla separada que se llame production_companies.
- Tiene dos atributos, namePCompany e idPCompany 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 pCountryName
  - Para solucionar, cada columna debe contener un solo valor.
  - La solución es crear una tabla separada que se llame Movie_production_countries.
  - Tiene dos atributos, iso_3166_1 y idMovie.
  - Una Movie puede tener muchos production_countries, y un production_countries puede aparecer en muchas Movie.

- 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.

- Cast contiene un String que contiene los nombres de los actores de una película que puede contener 0 a n Cast member.
    - Para solucionar, cada columna debe contener un solo valor.
    - En este caso tenemos una lista de cast. Existen múltiples valores.
    - La solución es crear una tabla separada que se llame Cast.
    - La primary key es CastName.

- Una Movie puede tener muchos Cast member, y un Cast member puede aparecer en muchas Movie.
- La relación (muchos a muchos) se soluciona con una tabla llamada Movie_Cast utilizando la llave primaria de cada uno.

- Crew contiene un String de JSON que contiene un name, gender, department, job, credit_id, id que puede contener 0 a n Crew members.
  - Para solucionar, cada columna debe contener un solo valor.
  - En este caso tenemos una lista de Crew members. Existen múltiples valores.
  - La solución es crear una tabla separada que se llame Crew.
  - Tiene los atributos name, gender, department, job, credit_id, id los cuales nombraremos crewName, gender, department, job, credit_id, idCrew el cual es la primary key.
  - Una Movie puede tener muchos Crew members, y un Crew members puede aparecer en muchas Movie.
  - La relación (muchos a muchos) se soluciona con una tabla llamada Movie_Crew utilizando la llave primaria de cada uno.

- Cada Movie tiene uno de los siguientes:
  - Production_countries
  - Production_companies

- Spoken_languages
- Crew

Pasos para Realizar el Proyecto

Creación de “Schema”.

MEDIANTE LA CREACIÓN AUTOMÁTICA

Para crear automáticamente el esquema en la base de datos de MySql, primero debes acceder a la parte superior izquierda de la pantalla donde se encuentran los íconos. Luego, debes seleccionar el ícono destinado para la creación de un esquema, como se puede ver en el anexo proporcionado.:

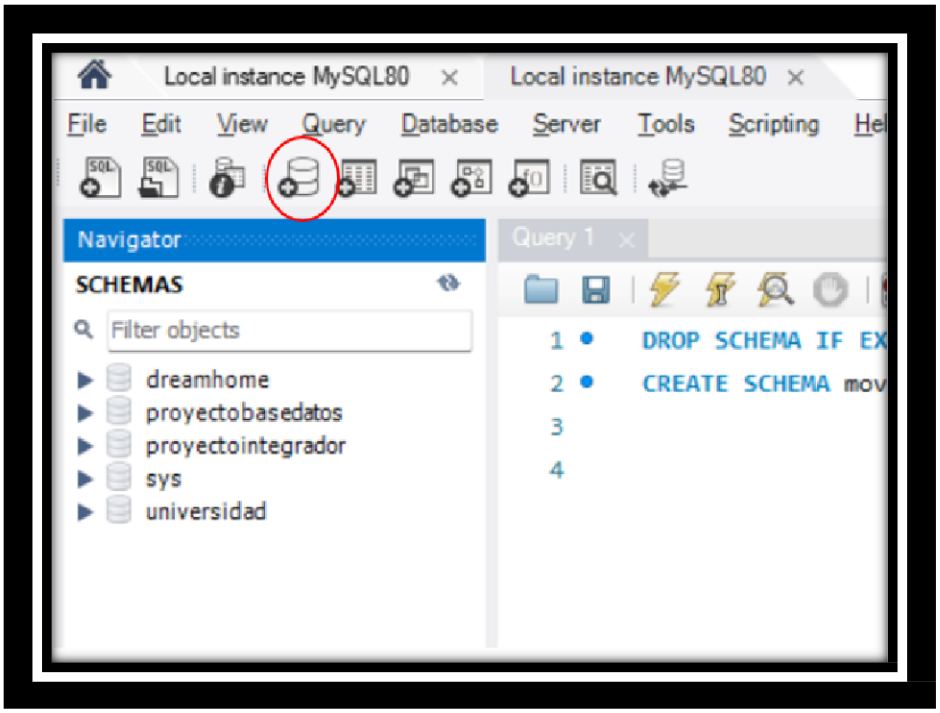


Figura 6: Representación de la barra de opciones perteneciente a Workbench

Después de seleccionar el ícono para crear un esquema, se le dará un nombre, en este caso "movie_dataset". En este esquema, se pueden realizar varias modificaciones, como la selección de un juego de caracteres, en este caso se ha utilizado "utf8":

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

MEDIANTE SENTENCIAS SQL

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

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

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

Conexión del “Schema” a la Base De Datos.

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

```
USE `Movie_Dataset` ;
```

Importación del CSV.

DataGrip es un entorno de desarrollo integrado (IDE) desarrollado por JetBrains para la gestión de bases de datos. Este contiene una herramienta que facilita la importación de diferentes tipos de archivos a las tablas de MySql. Los siguientes son los pasos que se describen en la imagen proporcionada:

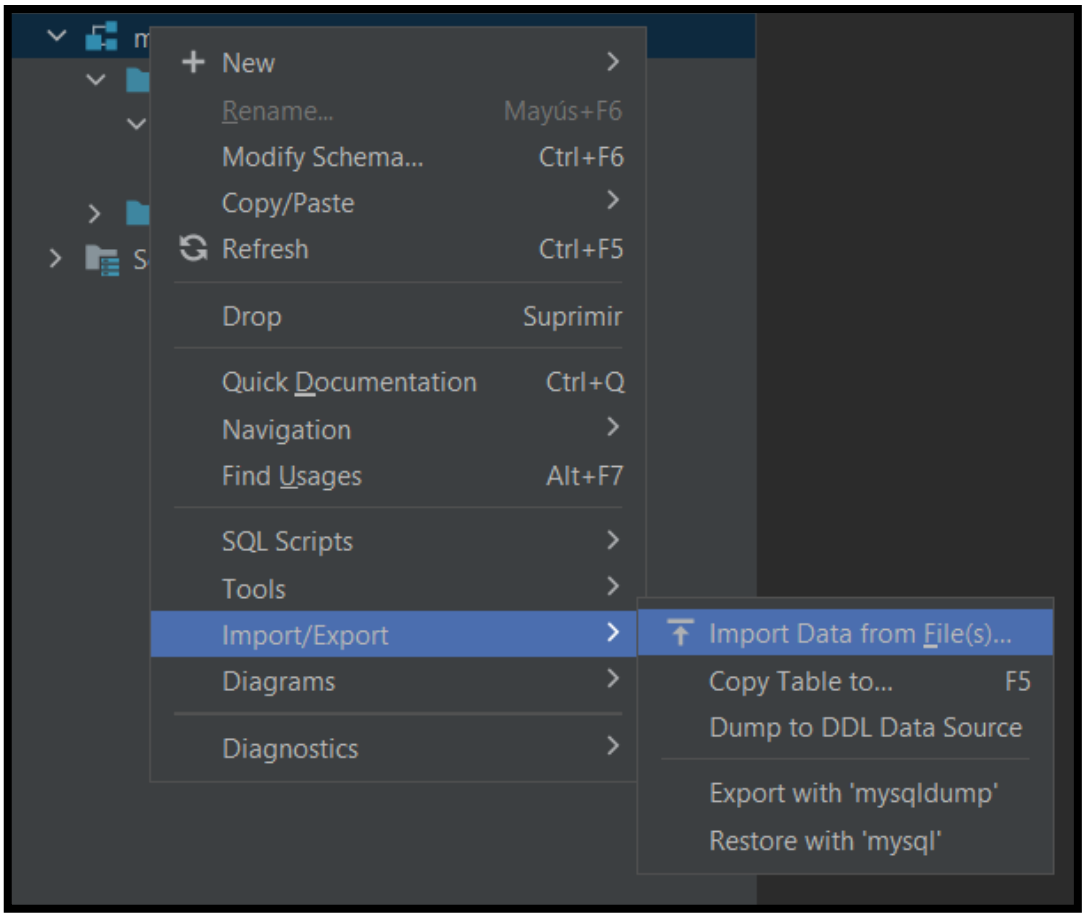


Figura 8: Representación de la funcionalidad de importación de archivos en DataGrip

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

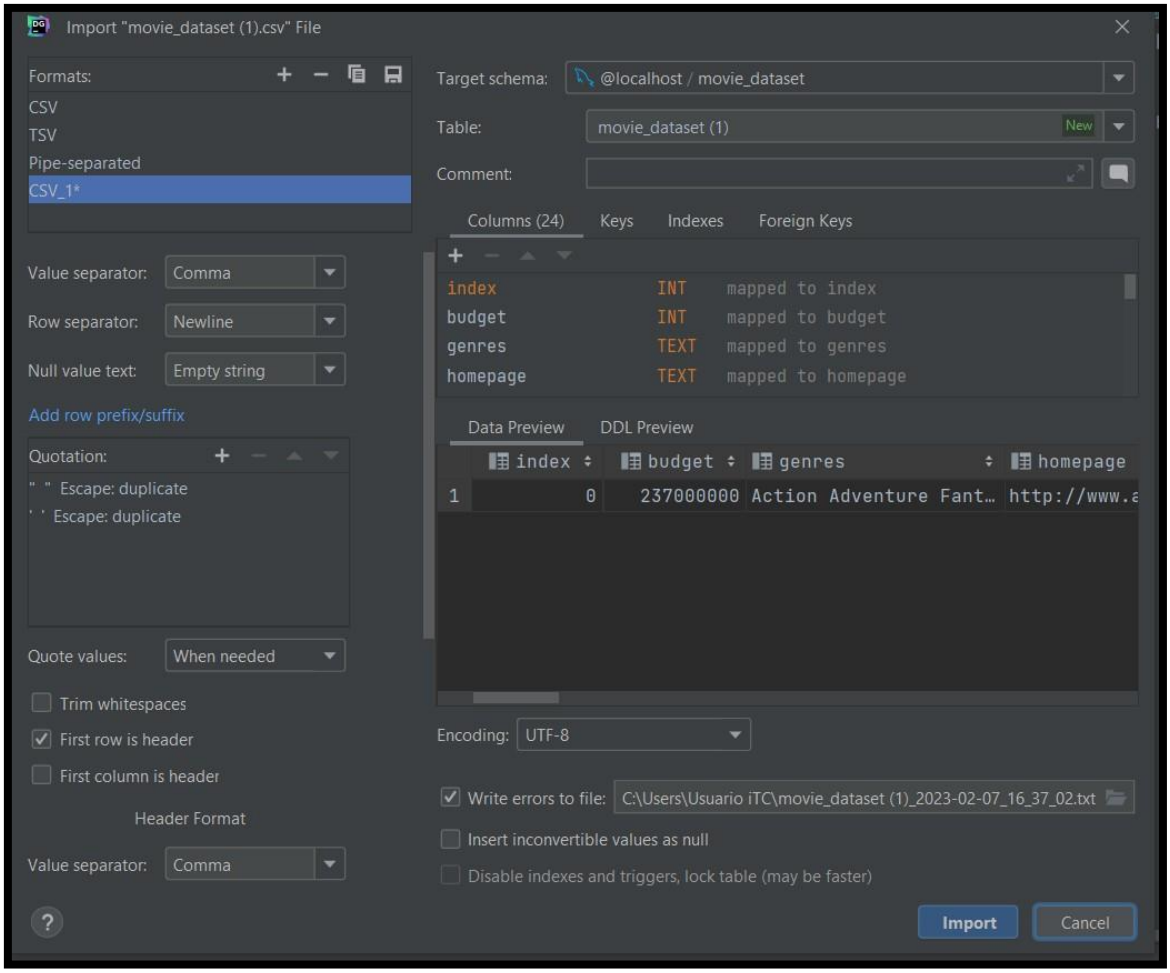


Figura 8: Representación de la interfaz de importación de archivos en DataGrip

Creación de funciones que Permitan Extraer y Limpiar los Datos.

En el procedimiento almacenado de SQL, se comienza verificando si existe un procedimiento con el nombre "TablaDirector". Si existe, se elimina. Luego, se declaran dos variables: la primera es "done" con un valor predeterminado de "FALSE", y la segunda es "nameDirector", que es de tipo VARCHAR y tiene una longitud máxima de 100 caracteres.

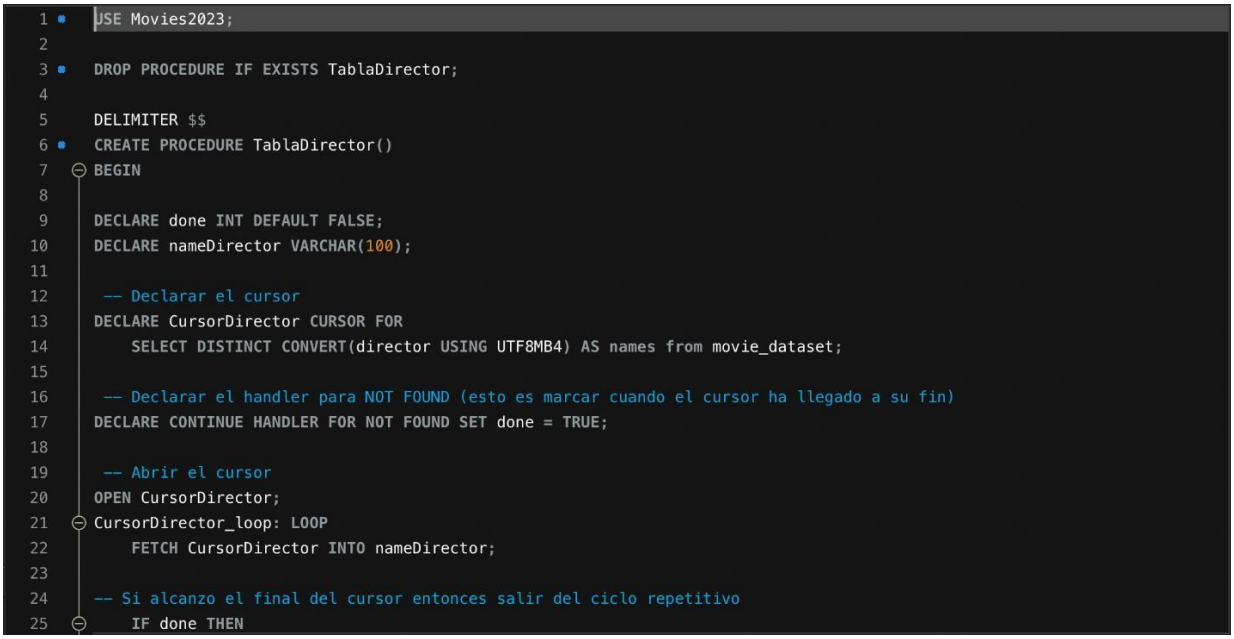
A continuación, se declara un cursor llamado "CursorDirector", que selecciona los nombres únicos de los directores en la tabla "movie_dataset". Se establece un manejador de continuación para detectar cuándo se ha llegado al final del cursor. Luego, se abre el cursor y se inicia un ciclo que recorre cada uno de los nombres de los directores en el cursor.

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

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

Finalmente, se cierra el cursor y se finaliza el procedimiento almacenado. En resumen, este procedimiento permite crear y llenar una tabla con los nombres únicos de los directores en una tabla de origen.Este código nos servirá para poblar estas tres tablas.

Población Tabla “Director”



```sql
USE Movies2023;

DROP PROCEDURE IF EXISTS TablaDirector;

DELIMITER $$
CREATE PROCEDURE TablaDirector()
BEGIN

DECLARE done INT DEFAULT FALSE;
DECLARE nameDirector VARCHAR(100);

 -- Declarar el cursor
DECLARE CursorDirector CURSOR FOR
    SELECT DISTINCT CONVERT(director USING UTF8MB4) AS names 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;
CursorDirector_loop: LOOP
    FETCH CursorDirector INTO nameDirector;

-- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
```

```
        LEAVE CursorDirector_loop;
    END IF;
    IF nameDirector IS NULL THEN
        SET nameDirector = '';
    END IF;
    SET @_oStatement = CONCAT('INSERT INTO directorCURSOR (name) VALUES (\'',
    REPLACE(REPLACE(nameDirector, '\'', '\\\''), '\\u', '\\\\u')
    ,'\');');
    PREPARE sent1 FROM @_oStatement;
    EXECUTE sent1;
    DEALLOCATE PREPARE sent1;

END LOOP;
CLOSE CursorDirector;
END $$
DELIMITER ;

CALL TablaDirector();

DROP TABLE IF EXISTS directorCURSOR;

CREATE TABLE directorCURSOR (
    name varchar(255) PRIMARY KEY
);
```

Población Tabla “Status”

```
USE Movies2023;

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) AS names 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;
CursorStatus_loop: LOOP
    FETCH CursorStatus INTO nameStatus;

-- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
```

```
        LEAVE CursorStatus_loop;
    END IF;
    IF nameStatus IS NULL THEN
        SET nameStatus = '';
    END IF;
    SET @_oStatement = CONCAT('INSERT INTO statusCURSOR (name) VALUES (\'',
    nameStatus,'\');');
    PREPARE sent1 FROM @_oStatement;
    EXECUTE sent1;
    DEALLOCATE PREPARE sent1;

END LOOP;
CLOSE CursorStatus;
END $$
DELIMITER ;

CALL TablaStatus();

DROP TABLE IF EXISTS statusCURSOR;

CREATE TABLE statusCURSOR (
    name varchar(255) PRIMARY KEY
);

SELECT * FROM statusCURSOR;
```

Población Tabla “Original_language”

```
USE Movies2023;

DROP PROCEDURE IF EXISTS TablaOriginalLanguage;

DELIMITER $$
CREATE PROCEDURE TablaOriginalLanguage()
BEGIN

DECLARE done INT DEFAULT FALSE;
DECLARE nameOL VARCHAR(100);

 -- Declarar el cursor
DECLARE CursorOL CURSOR FOR
    SELECT DISTINCT original_language AS names 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 CursorOL;
CursorOL_loop: LOOP
    FETCH CursorOL INTO nameOL;

-- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
```

```
        LEAVE CursorOL_loop;
    END IF;
    IF nameOL IS NULL THEN
        SET nameOL = '';
    END IF;
    SET @_oStatement = CONCAT('INSERT INTO original_languageCURSOR (name) VALUES (\'',
    nameOL,'\');');
    PREPARE sent1 FROM @_oStatement;
    EXECUTE sent1;
    DEALLOCATE PREPARE sent1;

END LOOP;
CLOSE CursorOL;
END $$
DELIMITER ;

CALL TablaOriginalLanguage();

DROP TABLE IF EXISTS original_languageCURSOR;

CREATE TABLE original_languageCURSOR (
    name varchar(255) PRIMARY KEY
);

SELECT * FROM original_languageCURSOR;
```

Debido a que la tabla "Movie" hace uso de claves externas, era necesario crear las tablas "Director", "Status" y "original_language" para utilizar sus claves primarias como claves externas en la tabla "Movie" (REFERENCIAS).

Población Tabla “Movie”

El siguiente código SQL es un procedimiento almacenado que crea una tabla llamada "películas" en la base de datos. La tabla se crea a partir de los datos de un conjunto de datos llamado "movie_dataset". El procedimiento almacenado comienza eliminando todos los procedimientos 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 los 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 "gestor" para manejar el caso en el que se llegue al final del cursor.

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

Si se llega al final del cursor, el ciclo termina. En caso contrario, se verifica si el nombre del director está vacío. Si es así, se le asigna un valor de cero. Se ejecuta una consulta para obtener la identificación del director a partir de su nombre y se guarda en la variable "Director_idDirector".

Finalmente, se inserta una nueva fila en la tabla "películas" con los valores obtenidos de la tabla "movie_dataset" y el ID del director. Cuando se completa el ciclo, el cursor se cierra y el procedimiento almacenado concluye.

```sql
DROP PROCEDURE IF EXISTS TablaMovie;

DELIMITER $$
CREATE PROCEDURE TablaMovie()
BEGIN

DECLARE done INT DEFAULT FALSE;
DECLARE Movindex INT;
DECLARE Movbudget BIGINT;
DECLARE Movhomepage VARCHAR(1000);
DECLARE MovidMovie INT;
DECLARE Movkeywords TEXT;
DECLARE Movoriginal_language VARCHAR(255);
DECLARE Movoriginal_title VARCHAR(255) ;
DECLARE Movoverview TEXT;
DECLARE Movpopularity DOUBLE;
DECLARE Movrelease_date VARCHAR(255);
DECLARE Movrevenue BIGINT;
DECLARE Movruntime DOUBLE;
DECLARE Movstatus VARCHAR(255);
DECLARE Movtagline VARCHAR(255);
DECLARE Movtitle VARCHAR(255);
DECLARE Movvote_average DOUBLE;
DECLARE Movvote_count INT;
DECLARE nameDirector VARCHAR(255);

DECLARE Director_nameDirector varchar(255);
DECLARE Director_nameStatus varchar(255);
DECLARE Director_nameOriginal_language varchar(255);

 -- Declarar el cursor
DECLARE CursorMovie CURSOR FOR
    SELECT `index`,budget,homepage,id,keywords,original_language,original_title,overview,popularity,release_date,revenue,
        tagline,title,vote_average,vote_count,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 CursorMovie;
CursorMovie_loop: LOOP
    FETCH CursorMovie INTO Movindex,Movbudget,Movhomepage,MovidMovie,Movkeywords,Movoriginal_language,Movoriginal_title,M
    Movpopularity,Movrelease_date,Movrevenue,Movruntime,Movstatus,Movtagline,Movtitle,Movvote_average,Movvote_count,nameD

    -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
    IF done THEN
        LEAVE CursorMovie_loop;
    END IF;
    IF nameDirector IS NULL THEN
    SET nameDirector = '';
```

```
    END IF;

    SELECT `name` INTO Director_nameDirector FROM directorCURSOR WHERE directorCURSOR.name = nameDirector;
    SELECT `name` INTO Director_nameStatus FROM statusCURSOR WHERE statusCURSOR.name = Movstatus;
    SELECT `name` INTO Director_nameOriginal_language FROM original_languageCURSOR WHERE original_languageCURSOR.name = M

    INSERT INTO MovieCURSOR (`index`,budget,homepage,id,keywords,original_language,original_title,overview,popularity,rel
        tagline,title,vote_average,vote_count,director)
    VALUES (Movindex,Movbudget,Movhomepage,MovidMovie,Movkeywords,Director_nameOriginal_language,Movoriginal_title,Movove
    Movpopularity,Movrelease_date,Movrevenue,Movruntime,Director_nameStatus,Movtagline,Movtitle,Movvote_average,Movvote_c

END LOOP;
CLOSE CursorMovie;
END $$
DELIMITER ;

CALL TablaMovie ();

DROP TABLE IF EXISTS MovieCURSOR;

CREATE TABLE MovieCURSOR (
    `index` int,
    budget bigint,
    homepage varchar(1000),
    id int PRIMARY KEY,
    keywords TEXT,
    original_language varchar(255),
    original_title varchar(255),
    overview TEXT,
    popularity double,
    release_date varchar(255),
    revenue bigint,
    runtime double,
    `status` varchar(255),
    tagline varchar(255),
    title varchar(255),
    vote_average double,
    vote_count int,
    director varchar(255),
    FOREIGN KEY (original_language) REFERENCES original_languageCURSOR(name),
    FOREIGN KEY (status) REFERENCES statusCURSOR(name),
    FOREIGN KEY (director) REFERENCES directorCURSOR(name)
);

DROP TABLE IF EXISTS MovieCURSOR;

SELECT COUNT(*) FROM MovieCursor;
SELECT * FROM MovieCursor;
```

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

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

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

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

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

Población Tabla “Production_companies”

```sql
USE Movies2023;

DROP PROCEDURE IF EXISTS TablaProduction_companies;

DELIMITER $$
CREATE PROCEDURE TablaProduction_companies ()

BEGIN

 DECLARE done INT DEFAULT FALSE ;
 DECLARE jsonData json ;
 DECLARE jsonId varchar(250) ;
 DECLARE jsonLabel varchar(250) ;
 DECLARE 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 los arrays
    SET i = 0;

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

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

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

```
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;

  END WHILE;

 END LOOP ;

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

END$$
DELIMITER ;

call TablaProduction_companies();

CREATE TABLE production_companiesCURSOR (
    id INT PRIMARY KEY,
    name varchar(100)
);

DROP TABLE production_companiesCURSOR;
```

Población Tabla “Production_countries”

```
USE Movies2023;

DROP PROCEDURE IF EXISTS TablaProduction_countries;

DELIMITER $$
CREATE PROCEDURE TablaProduction_countries ()

BEGIN

 DECLARE done INT DEFAULT FALSE ;
 DECLARE jsonData json ;
 DECLARE jsonId varchar(250) ;
 DECLARE jsonLabel varchar(250) ;
 DECLARE 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), nameCountry 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_countriesCURSOR
   SELECT DISTINCT iso_3166_1, nameCountry
   FROM production_countriesTem;
   drop table if exists production_countriesTem;
CLOSE myCursor ;

END$$
DELIMITER ;

call TablaProduction_countries();

SELECT * FROM production_countriesCURSOR;

CREATE TABLE production_countriesCURSOR (
    iso_3166_1 varchar(2) PRIMARY KEY,
    name varchar(100)
);
```

Población Tabla “Spoken_languages”

```sql
USE Movies2023;

DROP PROCEDURE IF EXISTS TablaSpokenLanguages;

DELIMITER $$
CREATE PROCEDURE TablaSpokenLanguages ()

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 spokenLanguagesTem;
    SET @sql_text = 'CREATE TABLE spokenLanguagesTem ( iso_639_1 varchar(2), nameLang VARCHAR(100));';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
 cursorLoop: LOOP
  FETCH myCursor INTO jsonData;

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

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

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

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

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

LIMPIEZA COLUMNA CREW

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".

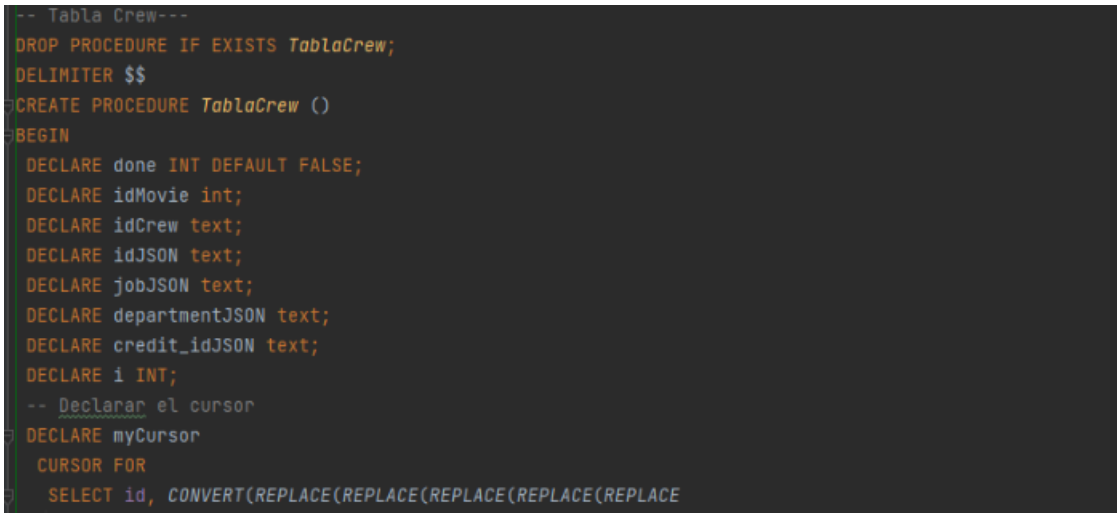
El cursor se abre y se inicia un loop infinito que se repetirá hasta que se alcance el final del cursor o se salga explícitamente del loop. Dentro del loop, se usa la función "FETCH" para

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.

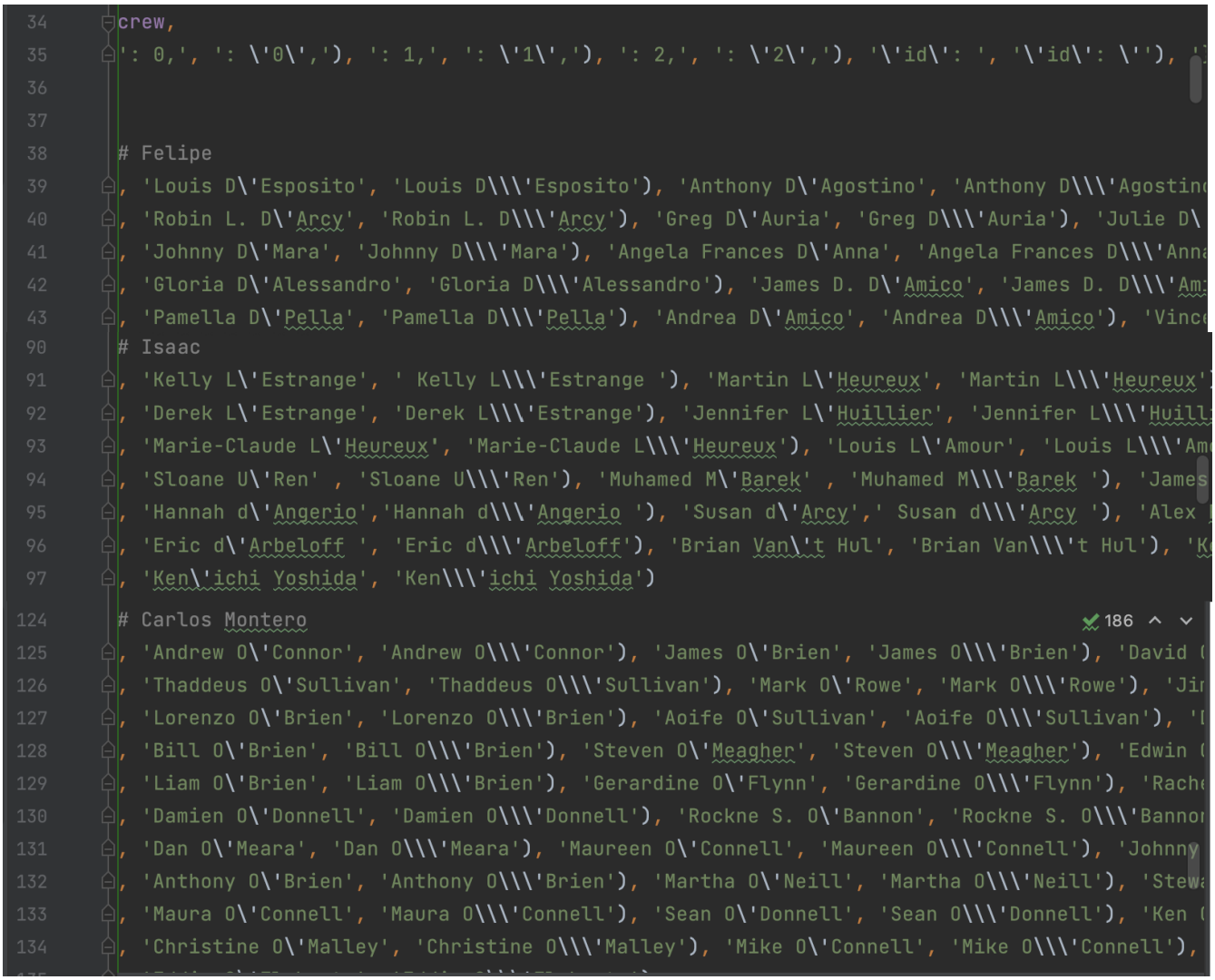
Finalmente, se construye una sentencia SQL dinámica usando la función "CONCAT" y se ejecuta usando las funciones "PREPARE", "EXECUTE" y "DEALLOCATE PREPARE".

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();".



```
-- 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
```



Conclusiones

Este proyecto lo llevé a cabo siguiendo unas normas de modelado de base de datos, como normalización, diseño conceptual, lógico, etc. Después de mucho esfuerzo por mi parte y de todos los demás miembros del grupo, logramos cumplir con todos los objetivos establecidos al inicio del desarrollo. Debemos destacar que enfrentamos varios desafíos a medida que avanzábamos en el proyecto, pero gracias a las directrices de nuestros profesores y a nuestra capacidad de resolver problemas, pudimos satisfacer los requisitos dados.

Una vez que completé todos los pasos necesarios para finalizar el proyecto, pude ver un gran progreso en mi capacidad de tomar decisiones sobre el modelado y la población de una base de datos, además de aprender nuevas estrategias de programación aplicables a situaciones de la vida real. El desarrollo de este proyecto sin duda me acerca más a ser un profesional seguro a la hora de trabajar en proyectos futuros.

Aprendí que los datos deben ser procesados cuidadosamente antes de su tratamiento para evitar errores y consistencia. También comprendí la importancia de las herramientas al momento de interactuar con la base de datos. Puedo concluir que todo lo que estudié este ciclo fue muy importante, ya que lo utilicé en este proyecto integrador. Debemos destacar que este proyecto integrador fue un desafío para mí como estudiante, ya que es la primera vez que trabajo con tanta cantidad de datos