



UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

Informe Proyecto Final
Fundamentos de Bases de Datos

Autor: Felipe Sebastián Jiménez Viñán

Octubre 2022 – Febrero 2023

Introducción	3
Modelos	4
Modelo Conceptual.....	4
Modelo Lógico	6
Dependencias Funcionales -Tabla Universal-.....	7
Modelo Físico	9
Normalización	10
First Normal Form (1NF).....	10
Second Normal Form (2NF)	11
Third Normal Form (3NF)	11
Pasos Para Realizar el Proyecto.....	17
Limpieza De Crew.....	¡Error! Marcador no definido.
Consultas	25
Conclusiones	31

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.

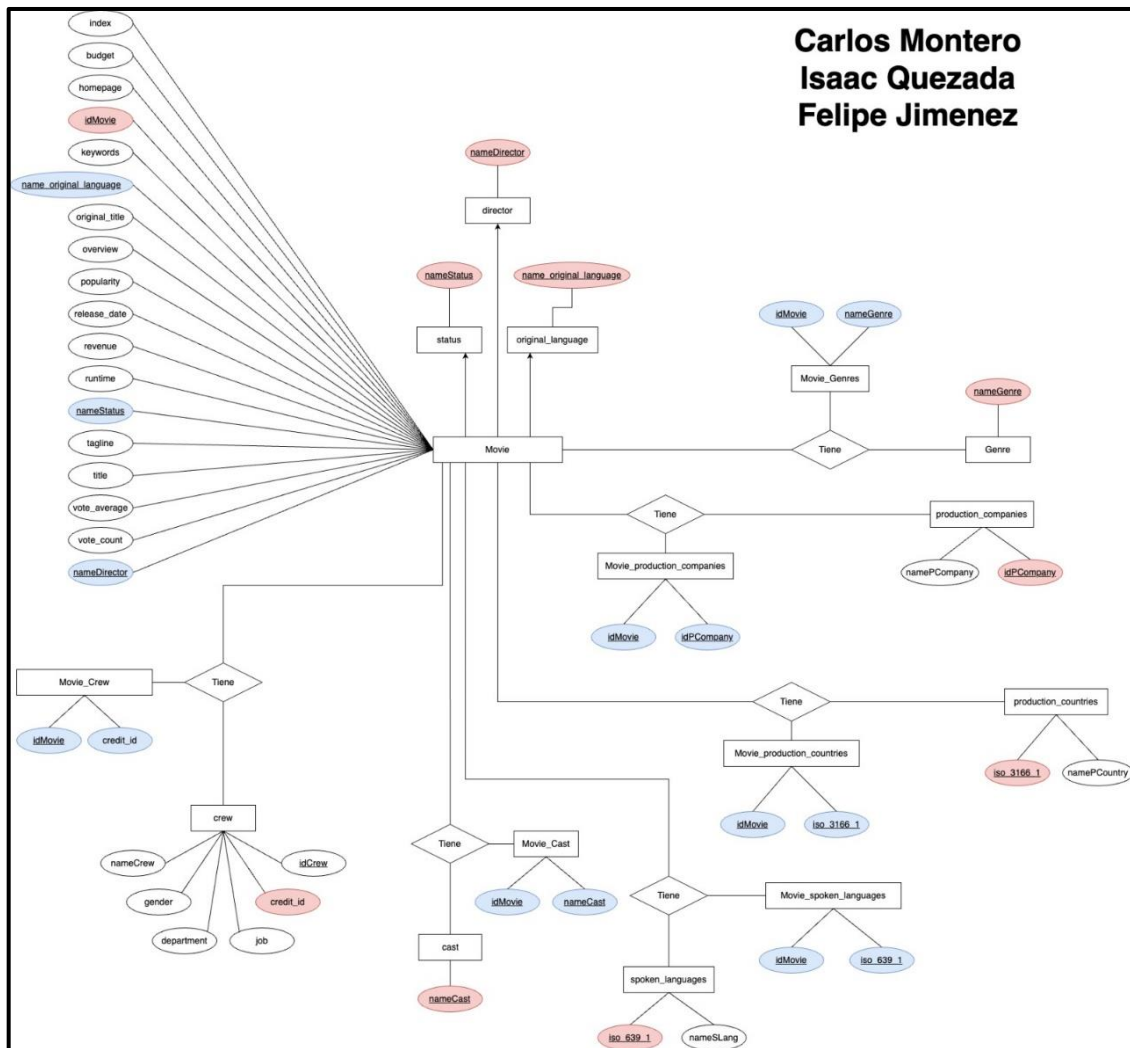
Este informe busca producir resultados orientados a la gestión y uso eficiente de la información relevante y de interés, a partir del conjunto de datos de trabajo. Además, nos permitió recordar varios temas estudiados en la materia tales como diagramas de Entidad/Relación, modelos conceptuales, lógicos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

Modelos

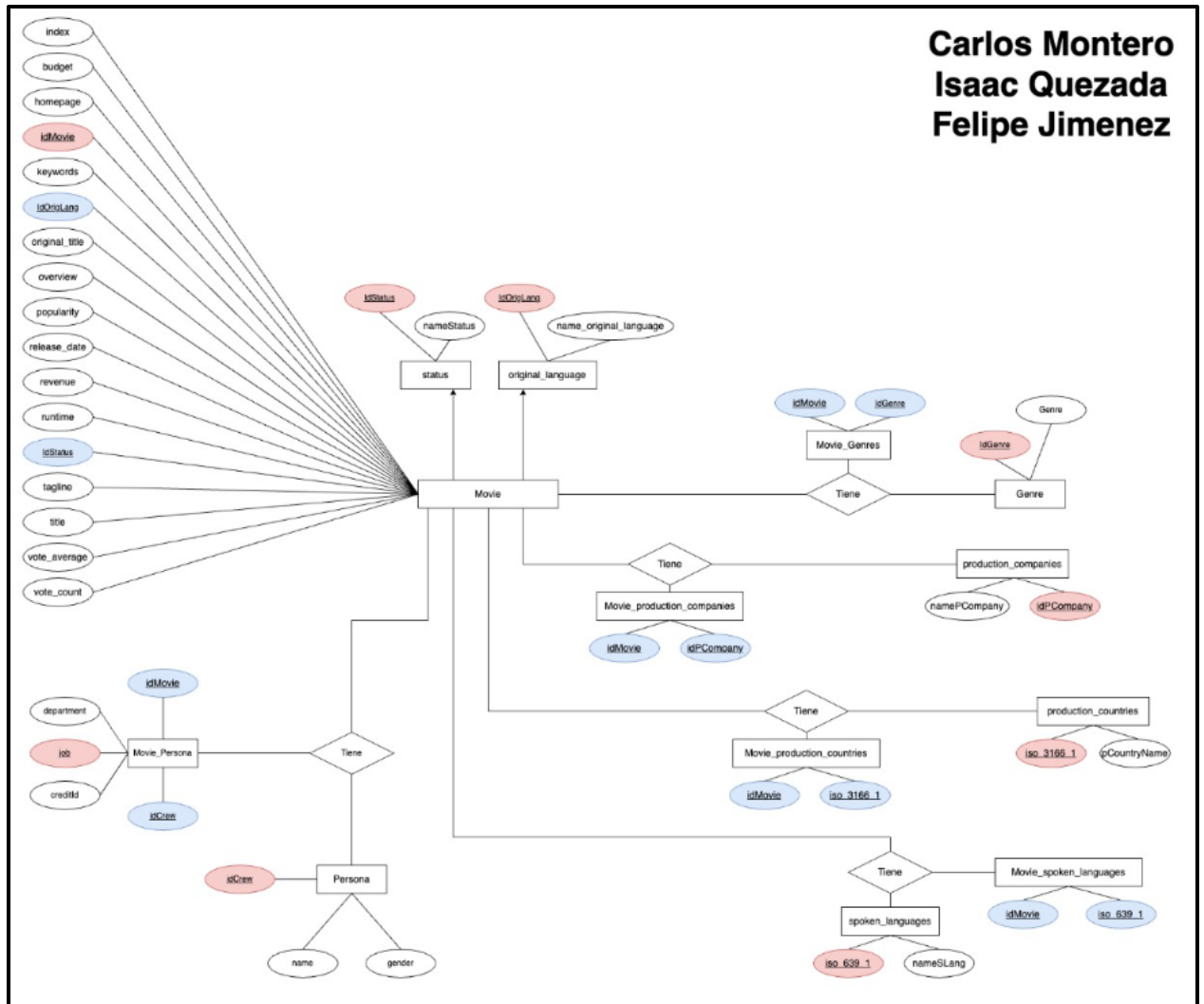
Modelo Conceptual

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

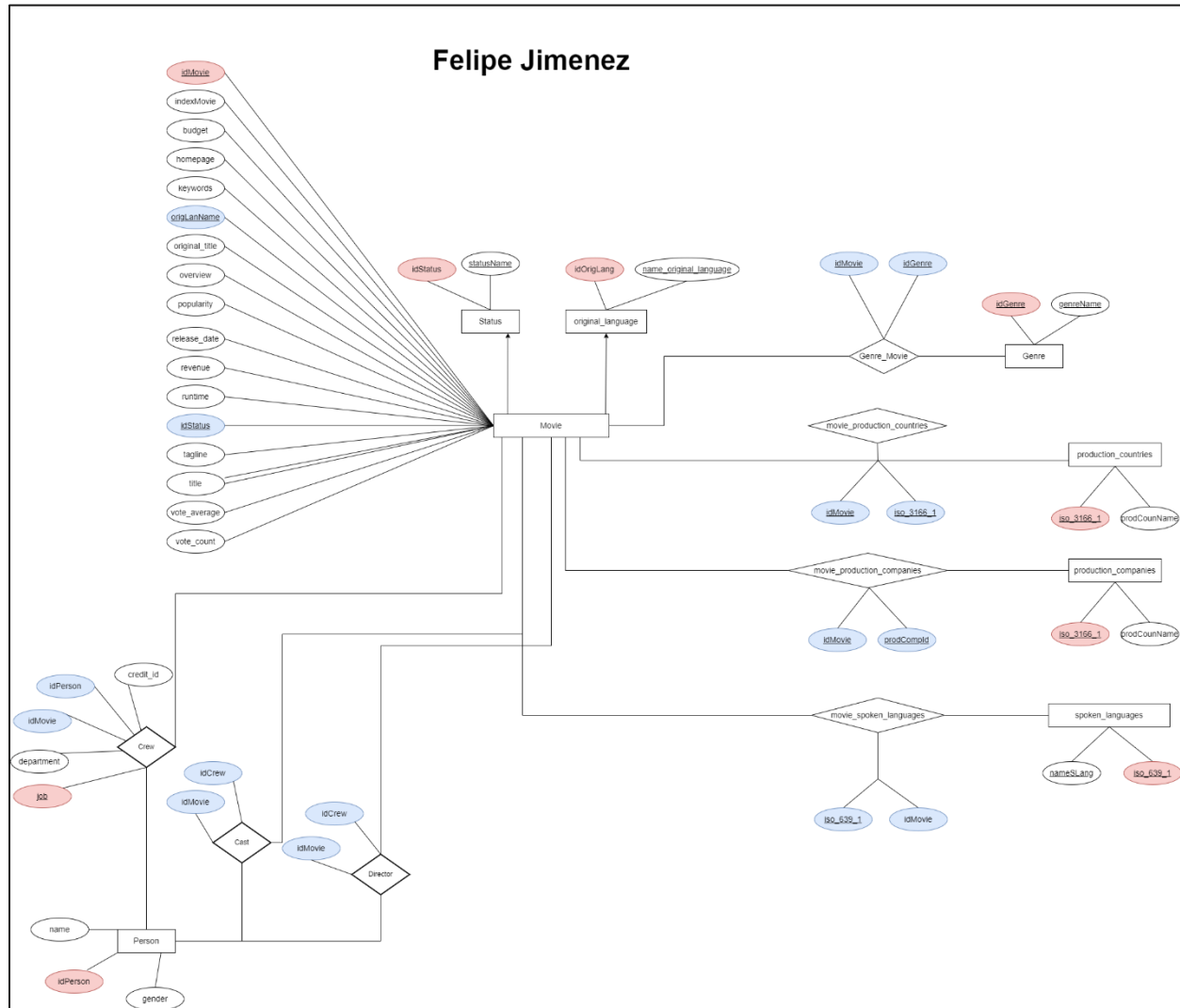
VERSION 1



VERSION 2



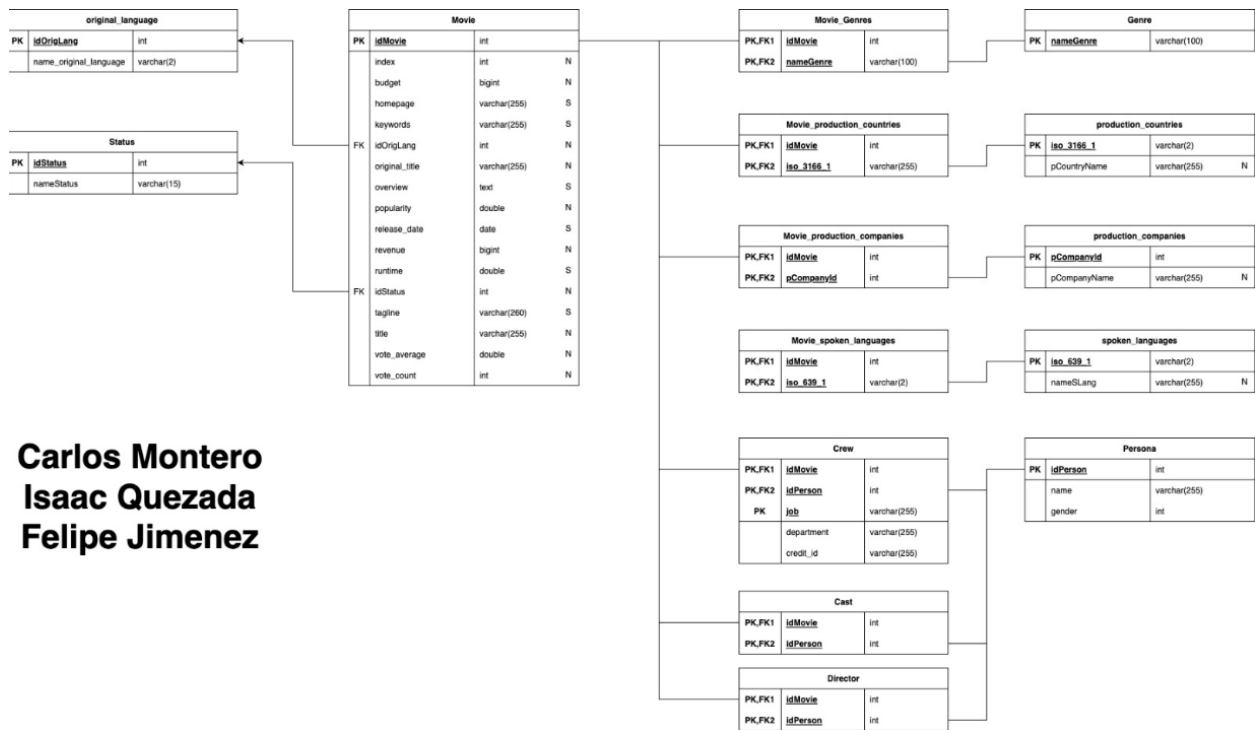
VERSION FINAL



Modelo Lógico

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



Carlos Montero
Isaac Quezada
Felipe Jimenez

Dependencias Funcionales -Tabla Universal-

Una vez establecido el modelo lógico y conceptual se creó las dependencias funcionales existentes en la tabla universal.

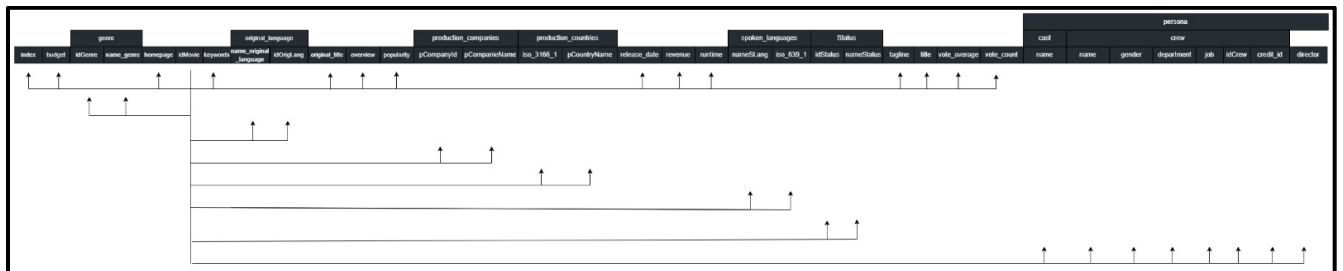


Figura 3: La imagen muestra una descripción gráfica de las dependencias existentes en las columnas del dataset

id → { index, budget, homepage, keywords, original_title, overview, popularity, release_date, revenue, runtime, tagline, title, vote_average, vote_count }

id → { idGenre, name_Genre }

id → { name_original_language, idOrigLang }

id → { pcompanyId, pcompanyName }

id → { iso_3166_1, pcountryName }

id → { nameSLang, iso_639_1 }

id → { idStatus, nameStatus }

id → { name, gender, department, job, idCrew, credit_id, director }

Figura 4: Dependencias Funcionales textualizadas para un mejor entendimiento

Modelo Físico

Una vez terminado el modelo lógico pasamos a la fase final que es la creación del modelo físico donde se especificó las tablas, columnas, claves principales y a su vez sus claves externas o foráneas con sus respectivas relaciones. Con este modelo podemos pasar a generar las respectivas sentencias DDL

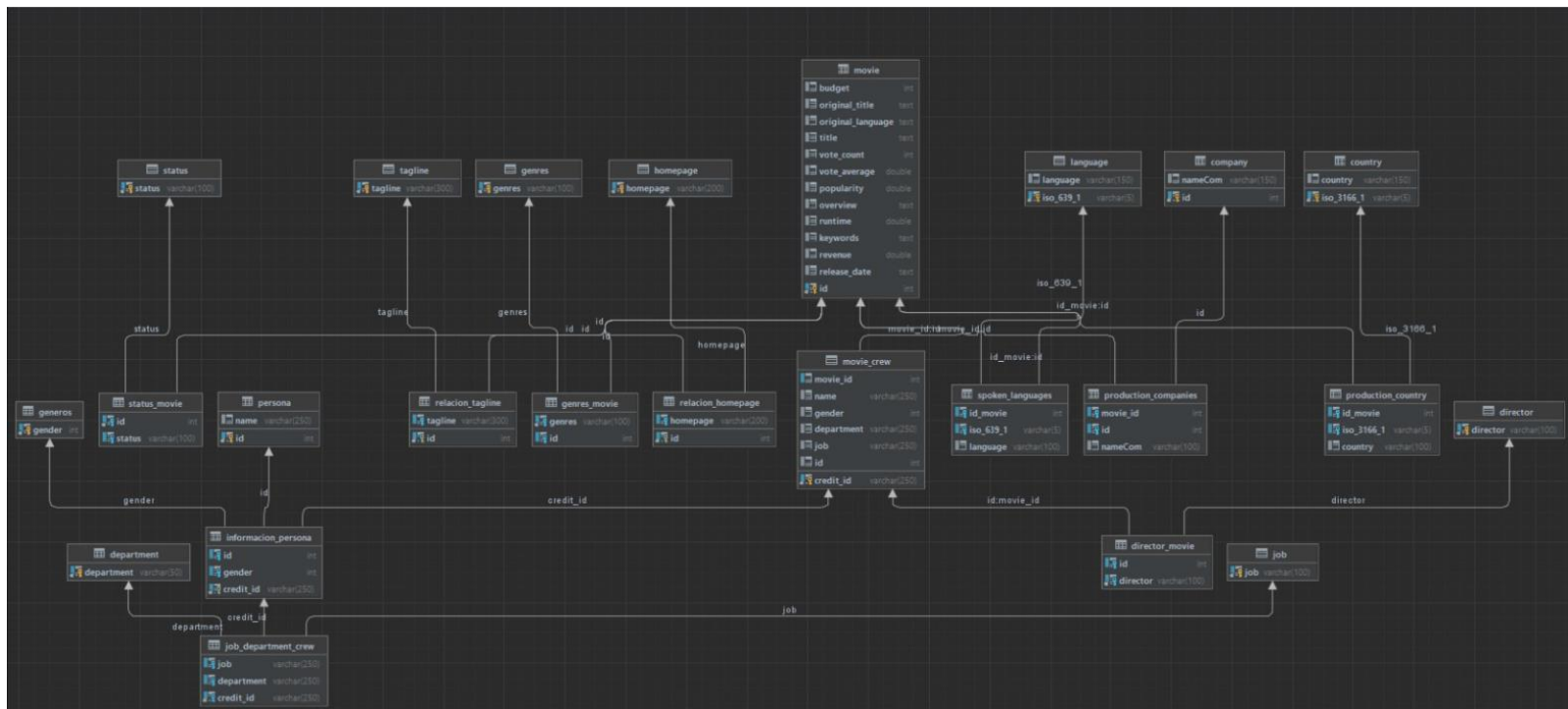


Figura 5: Representación gráfica del Modelo Físico

Normalización

Antes de entrar a este tema debemos saber su concepto y lo que conlleva aplicarlo en este proyecto, este proceso se realiza en la estructura de las bases de datos para poder así mejorar en temas de eficiencia, organización e integridad, con el objetivo de eliminar los tipos de redundancia y dependencia funcional. Otro concepto mayor mente conocido por los usuarios es la división de información en diferentes tablas y hacemos relaciones entre sí.

Seguimos los siguientes pasos para normalizar los datos del archivo CSV:

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 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 statusName.

Como solución, utilizamos el statusName como llave foránea en Movie. Director es una columna con el nombre de un único director.

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

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 origLanName.
- Como solución, utilizamos el origLanName 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.

- Genres contiene un String de géneros que puede contener 0 a n géneros.
 - Para solucionar, cada columna debe contener un solo valor.
 - En este caso tenemos una lista de géneros. Existen múltiples valores.
 - La solución es crear una tabla separada que se llame Genres.
 - La llave primaria es genreName
 - Una Movie puede tener muchos géneros, y un género puede estar en muchas Movie.
 - La relación (muchos a muchos) se soluciona con una tabla llamada Movie_Genres utilizando la llave primaria de cada uno.

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

- Production_companies contiene un String de JSON que contiene un name y un id que puede contener 0 a n production_companies.
 - Para solucionar, cada columna debe contener un solo valor.
 - En este caso tenemos una lista de production_companies. Existe 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.
 - **'Cast', 'Crew' y 'Director'** contienen los nombres de integrantes que participan de alguna forma en la película. Estos necesitan ser normalizados en una tabla con datos comunes, en este caso llamada **Persona**.
 - Se crea una tabla Persona que incluya los atributos comunes a todas las entidades, entre los cuales están: gender, idCrew y name.

- Ya que la relacion Persona-Movie es muchos a muchos, por consecuencia se necesita otra tabla que contenga el idMovie e idCrew.
- Es momento de crear tablas de '*Cast*', '*Crew*' y '*Director*', para la migracion de datos hacia la tabla persona.
- Antes de migrar los datos a la tabla Persona, es importante realizar una limpieza de datos para garantizar que los datos sean precisos y coherentes.

Pasos Para Realizar el Proyecto

1. 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:

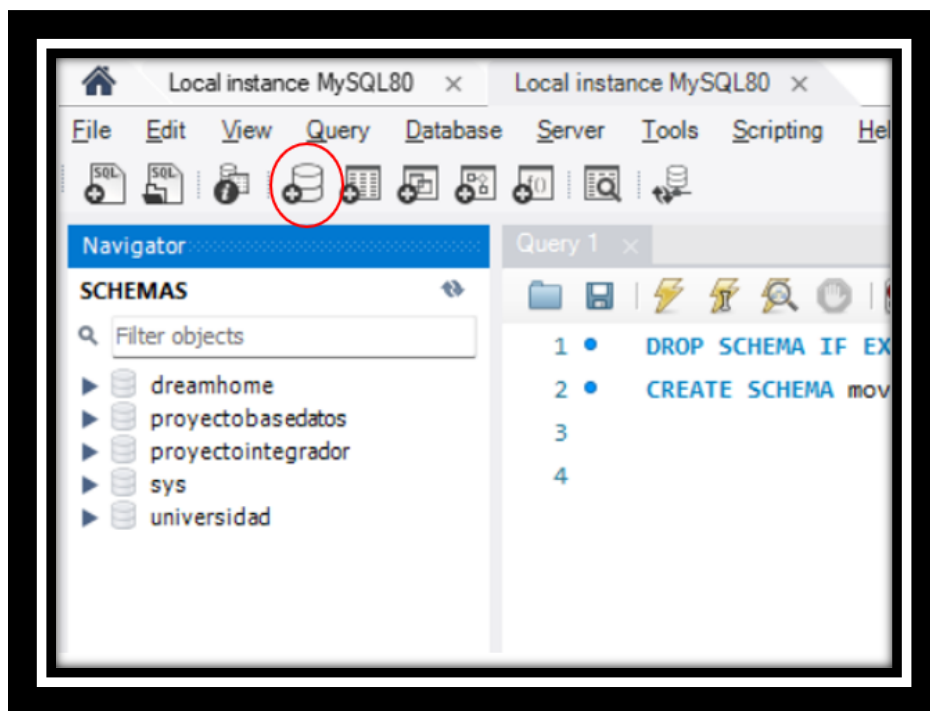


Figura 6: Representación de la barra de opciones perteneciente a Workbench

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”:

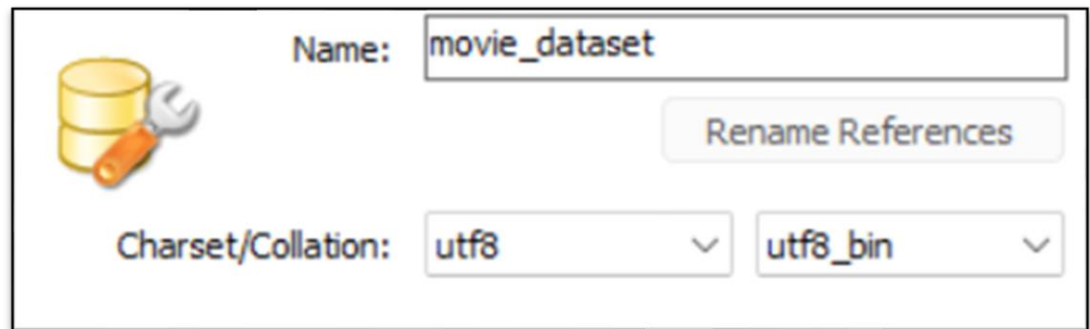


Figura 7: Representación de la funcionalidad de creación de Schema automáticamente

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 `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.

2. 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` ;
```

3. 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:

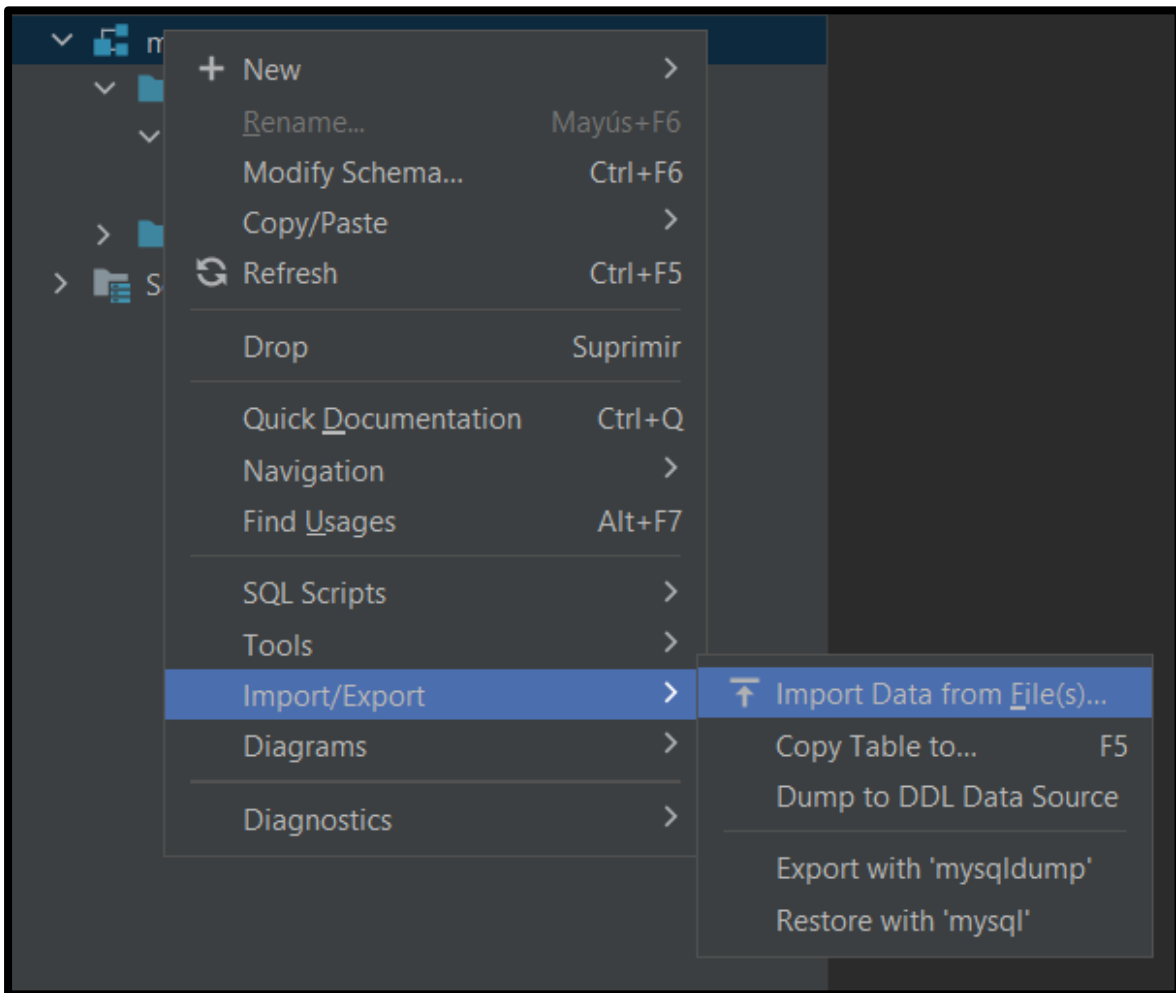


Figura 8: Representación de la funcionalidad de importación de archivos en DataGrip

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

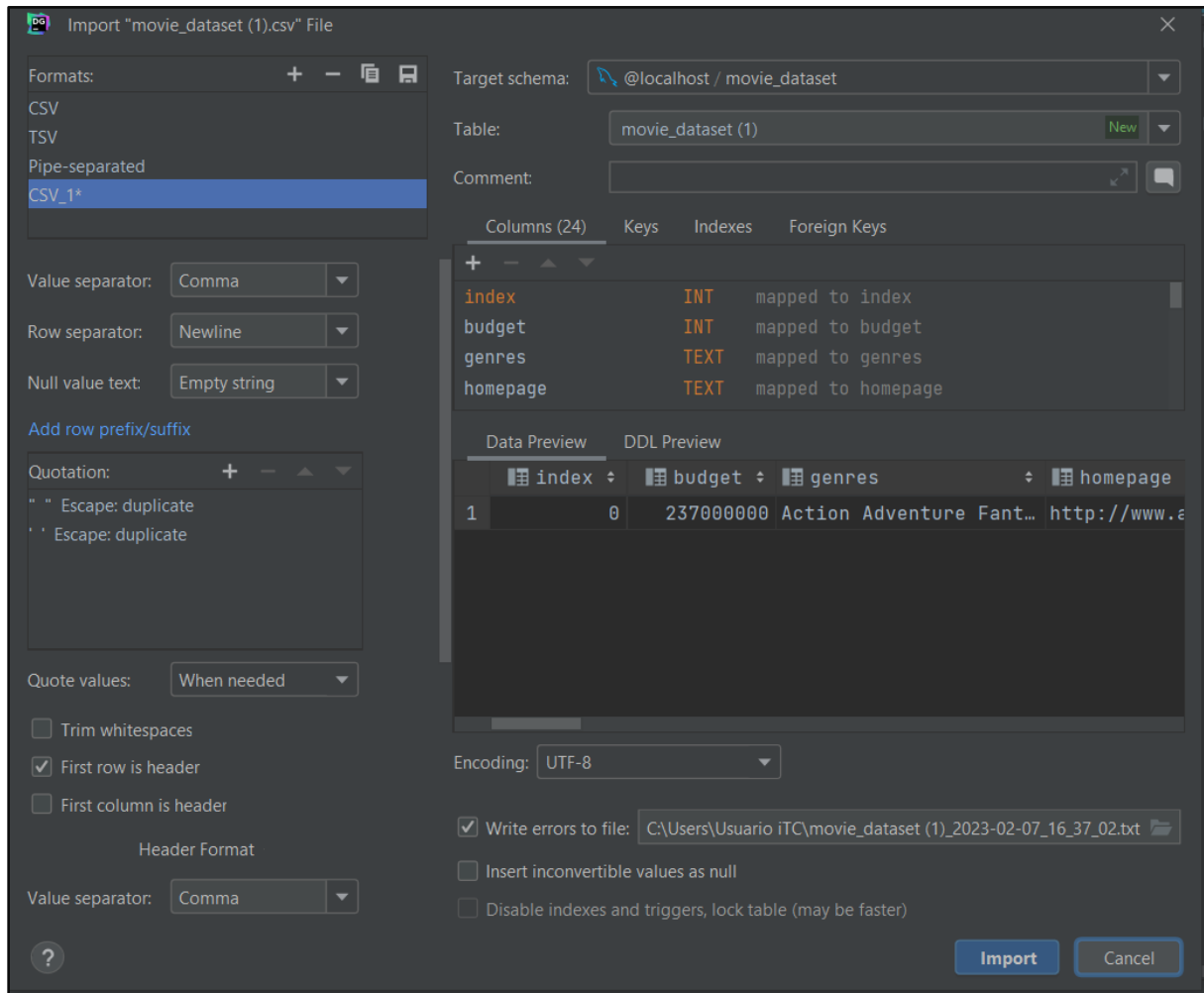


Figura 8: Representación de la interfaz de importación de archivos en DataGrip

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

Original_language. Para la integridad de las tablas, original language tuvo que ser creada primero junto a status, ya que estas no dependen de ninguna clave foránea y en cambio movies dependen de estas. Se hizo el siguiente procedimiento

```
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;
```

Status. Este sigue la lógica de original_language para la creación de la tabla. Se hizo un procedimiento para este.

```
-- 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;
```

Cast. Dentro de cast, no pudimos encontrar un patrón claro en el que podamos separar los nombres de manera coherente. Es por esto y por falta de tiempo que se decidió

Movie. Una vez creadas las tablas de relaciones uno a muchos, es momento de poblar la tabla principal, la cual es movie

```

OPEN CursorMovie;
) CursorMovie_loop: LOOP
    FETCH CursorMovie INTO Movindex,Movbudget,Movhomepage,MovidMovie,Movkeywords,Movororiginal_language,Movororiginal_title,Movoverview,
    Movpopularity,Movrelease_date,Movrevenue,Movruntime,Movstatus,Movtagline,Movtitle,Movvote_average,Movvote_count,nameDirector;

    -- 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 = Movororiginal_langu

) INSERT INTO MovieCURSOR (`index`,budget,homepage,id,keywords,original_language,original_title,overview,popularity,release_date,revenu
~ tagline,title,vote_average,vote_count,director)
) VALUES (Movindex,Movbudget,Movhomepage,MovidMovie,Movkeywords,Director_nameOriginal_language,Movororiginal_title,Movoverview,
~ Movpopularity,Movrelease_date,Movrevenue,Movruntime,Director_nameStatus,Movtagline,Movtitle,Movvote_average,Movvote_count,Director_na

~ END LOOP;
CLOSE CursorMovie;

```

Production_companies. Al ser este un JSON, es necesario usar la función JSON_EXTRACT para separar los objetos del JSON y pasarlos a la tabla

```

OPEN myCursor ;

drop table if exists MovieProdCompTemp;
SET @sql_text = 'CREATE TABLE MovieProdCompTemp ( 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 MovieProdCompTemp VALUES (', idMovie, ', ', REPLACE(idJSON,'\"', '\"'), '); ');
    PREPARE stmt FROM @sql_text;

```

Production_countries.

```
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 MovieProdCompTemp VALUES (', idMovie, ', ', REPLACE(idJSON, '\\', '\\\\'), '); ');
        PREPARE stmt FROM @sql_text;
        EXECUTE stmt;
        DEALLOCATE PREPARE stmt;
```

Crew. Crew tiene una particularidad, y es que tendría que ser un JSON, sin embargo tiene un error de formato que necesita ser tratado mediante el siguiente código:

```
SELECT
JSON_EXTRACT(CONVERT(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE
    (REPLACE(crew, '"', '\\'), '{', '\\', '{'),
    '\\: \', '"': '"').\\, \', '"', '"').\\: ', '"': ')', '\\', ', "')
USING UTF8mb4 ), '$[*]') FROM movie_dataset
```

```

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

```

Director.

```

-- Abrir el cursor
OPEN CursorDirector;
drop table if exists directorTemp;
SET @sql_text = 'CREATE TABLE directorTemp ( idMov int, idPer 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 Persona WHERE Persona.name = MovDirector;
    INSERT INTO directorTemp VALUES (Movid, idPersonas);

END LOOP;
CLOSE CursorDirector;

```


Consultas

The screenshot displays a database management interface with the following components:

- Left Panel (Database Explorer):** Shows the 'MoviesTaller' database with 15 tables. The 'Cast' table is selected.
- Top Panel (SQL Editor):** Contains the following SQL queries:




```
26 SELECT * FROM Cast WHERE nameCast LIKE '%Chris%';  
27 SELECT COUNT(*) FROM Cast;  
28 ✓ SELECT * FROM Cast WHERE nameCast LIKE '%jr%';  
29 SELECT * FROM Cast;
```
- Bottom Panel (Services):** Displays a list of services and their execution times:
 - Genre 132 ms
 - Genre 132 ms
 - Cast 250 ms
 - Cast 250 ms
 - console 65 ms
 - console 65 ms (selected)
 - Practica.sql
 - PRACTICE CREW.
 - Movie_Crew 306 ms
 - Movie_Crew 306 ms
 - Movie 182 ms
- Output Panel (MoviesTaller.Cast):** Shows the results of the selected query, displaying 17 rows. The first 9 rows are visible:









	nameCast
1	Cuba Gooding Jr.
2	Cuba Gooding Jr. William
3	Damon Wayans Jr.
4	Ed Begley Jr.
5	Eric D. Hill Jr.
6	Freddie Prinze Jr.
7	James Pickens Jr. Phylicia
8	John Gallagher Jr.
9	Jr Cheech Marin



36 

37

SELECT SUM(budget) FROM Movie;

 Output  SUM(budget):bigint 

  1 row      

 `SUM(budget)` 

1

139503326521

@localhost 5 of 9

MoviesTaller

tables 15

Cast

Crew

Director

Genre

Movie

Movie_Crew

Movie_Genres

Movie_production_ci

Movie_production_ci

Movie_spoken_langa

26

27

28 ✓

29

30

31

32

```

SELECT * FROM Cast WHERE nameCast LIKE '%Chris%';
SELECT COUNT(*) FROM Cast;
SELECT * FROM Cast WHERE nameCast LIKE '%jr%';
SELECT * FROM Cast;

```

Services

Output

MoviesTaller.Cast

17 rows

nameCast

1 Cuba Gooding Jr.

2 Cuba Gooding Jr. William

3 Damon Wayans Jr.

4 Ed Begley Jr.

5 Eric D. Hill Jr.

6 Freddie Prinze Jr.

7 James Pickens Jr. Phylicia

8 John Gallagher Jr.

9 Jr Cheech Marin

Genre 132 ms

Genre 132 ms

Cast 250 ms

Cast 250 ms

console 65 ms

console 65 ms

Practica.sql

PRACTICE CREW.:

Movie_Crew 306 ms

Movie_Crew 306 ms

Movie 182 ms

36 ✓ `SELECT SUM(budget) FROM Movie;`

37

Output SUM(budget):bigint ×

1 row

	1
1	139503326521

36 ✓ `SELECT title, budget FROM Movie WHERE budget > 100000000 ORDER BY 1;`

37

Output MoviesTaller.Movie ×

279 rows

	title	budget
1	2012	200000000
2	300: Rise of an Empire	110000000
3	47 Ronin	175000000
4	A Bug's Life	120000000
5	A Christmas Carol	200000000
6	After Earth	130000000
7	Alexander	155000000
8	Ali	107000000
9	Alice in Wonderland	200000000

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.