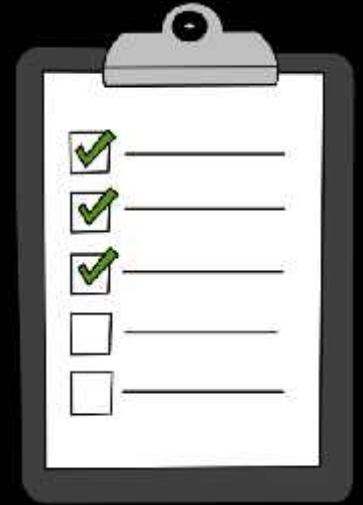




Proyecto integrador 1.1

POR:
GÉNESIS GONZÁLEZ
ROGER CRIOLLO
MICHAEL MALDONADO

OBJETIVO DEL PROYECTO



Integrar el conocimiento de las materias fundamentales en las ciencias de la computación como lo son Programacion Funcional y Reactiva y Fundamentos de Base de datos

CONCEPTUALIZACIÓ

N

CONCEPTUALIZACIÓ N

Base de datos



La historia de las bases de datos es muy amplia y ha ido modificándose y optimizándose acorde a las exigencias del desarrollo de programas y aplicaciones actuales, los inicios de las bases de datos los tenemos con el modelo jerárquico que con el tiempo ha sido remplazado principalmente por el modelo relacional u otros tipos.

Base de datos relacionales

Las bases de datos relacionales son un modelo en el cual se compone por tablas compuestas por registros, entidades y claves primarias. Su nombre se da debido a la forma en la que se puede trabajar con este tipo de dato con la libertad de relacionar entre diferentes tablas. La relación entre entidades se mediante una clave principal única.

La estructura relacional comprende los siguientes términos:

Relación: Comprende la idea general de tabla

Atributo: Corresponde una columna

Tupla: Comprende una fila

Grado: Número de atributos(n)

Cardinalidad: Número de tuplas (m)

Clave primaria: Identificador único (no existen tuplas con el mismo identificador)

Dominio: Colección de valores que da el valor al atributo.



PROGRAMACIÓN FUNCIONAL REACTIVA

Muchas aplicaciones, sobre todo en determinados dominios, requieren un cambio de enfoque a la hora de plantear soluciones, pasando de generar sistemas que son transformacionales (es decir, que presuponen un estado global que define el mundo, y por operaciones sucesivas van transformándolo para llegar a una solución) a generar sistemas reactivos por lo que se ha implementado la programación funcional y reactiva.

Archivo Csv



El nombre Csv corresponde a las siglas Comma Separated Values que en español significa valores separados por coma que nos da a entender en que consiste este tipo de archivo

C	D	E	F	G	H	I	J	K	L	M	N	O
type	homepage	id	keywords	original_lang	original_title	overview	popularity	production_c	production_o	release_date	revenue	runtime
Action Adventure	http://www.s	19995	culture clash	en	Avatar	In the 22nd ce	150437577	[[{"name": "In	[[{"so_3166_1	10/12/2009	2787965087	
Adventure Far	http://disney.	285	ocean drugat	en	Pirates of the Captain Barbo		139082615	[[{"name": "W	[[{"so_3166_1	19/5/2007	961000000	
Action Adventure	http://www.s	206647	spy based on	en	Spectre	A cryptic mes	1.07E+16	[[{"name": "Cc	[[{"so_3166_1	26/10/2015	880674609	
Action Crime	http://www.t	49026	dc comics crim	en	The Dark Knight Following the		11231295	[[{"name": "Le	[[{"so_3166_1	16/7/2012	1084939099	
Action Adventure	http://movies	49529	based on novel	en	John Carter	John Carter is	43926995	[[{"name": "W	[[{"so_3166_1	7/3/2012	284139300	
Fantasy Action	http://www.s	559	dual identity	en	Spider-Man 3: The weaving		1.1E+16	[[{"name": "Cc	[[{"so_3166_1	1/5/2007	800871626	
Animation Far	http://disney.	38757	hostage magi	en	Tangled	When the king	68881969	[[{"name": "W	[[{"so_3166_1	29/11/2010	591794936	
Action Adventure	http://marvel	99861	marvel comic	en	Avengers: Age When Tony St		1.34E+16	[[{"name": "M	[[{"so_3166_1	22/4/2015	1405403694	
Adventure Far	http://harryp	767	witch magic	en	Harry Potter : As Harry begi		98885637	[[{"name": "W	[[{"so_3166_1	7/7/2009	933859197	
Action Adventure	http://www.b	209112	dc comics vigi	en	Batman V Sup Fearing the ac		155790452	[[{"name": "Ox	[[{"so_3166_1	23/3/2016	873260194	
Adventure Far	http://www.s	1452	saving the wo	en	Superman Ret Superman ret		57925623	[[{"name": "Ox	[[{"so_3166_1	28/6/2006	391081192	
Adventure Act	http://www.s	10764	rolling underc	en	Quantum of S Quantum of S		1.08E+16	[[{"name": "Ec	[[{"so_3166_1	30/10/2008	586090727	
Adventure Far	http://disney.	58	witch fortune	en	Pirates of the Captain Jack		1.46E+16	[[{"name": "W	[[{"so_3166_1	20/6/2006	106505812	
Action Adventure	http://disney.	57201	texas horse st	en	The Lone Ran The Texas Rai		49046956	[[{"name": "W	[[{"so_3166_1	3/7/2013	89289910	
Action Adventure	http://www.s	49521	saving the wo	en	Man of Steel	A young boy is	95358009	[[{"name": "Le	[[{"so_3166_1	12/6/2013	662845518	
Adventure Family Fantasy		2454	based on novel	en	The Chronicle One year afte		51978602	[[{"name": "W	[[{"so_3166_1	15/5/2008	419651413	
Science Fictio	http://marvel	24428	new york shie	en	The Avengers	When an unes	144448633	[[{"name": "Fo	[[{"so_3166_1	25/4/2012	1519557930	
Adventure Act	http://disney.	1865	sea captain m	en	Pirates of the Captain Jack		135413856	[[{"name": "W	[[{"so_3166_1	14/5/2011	1045713802	
Action Comic	http://www.s	41154	time travel tle	en	Men in Black : Agents J (Will		52035179	[[{"name": "Ar	[[{"so_3166_1	23/5/2012	624026776	
Action Adventure	http://www.t	122917	corruption elv	en	The Hobbit: Ti Immediately		1.21E+16	[[{"name": "W	[[{"so_3166_1	10/12/2014	956019788	
Action Adventure	http://www.t	1930	loss of father	en	The Amazing : Peter Parker i		88866276	[[{"name": "Cc	[[{"so_3166_1	27/6/2012	752215857	
Action Adventure	http://www.s	20662	robin hood ar	en	Robin Hood	When soldier	37668301	[[{"name": "W	[[{"so_3166_1	12/5/2010	310669540	
Adventure Far	http://www.t	57158	elves dwarves	en	The Hobbit: Ti The Dwarves		94370564	[[{"name": "W	[[{"so_3166_1	11/12/2013	958400000	
Adventure Far	http://www.g	2168	england comp	en	The Golden Cr After overhea		42990906	[[{"name": "Nv	[[{"so_3166_1	4/12/2007	372214864	
Adventure Drama Action		254	film business	en	Kong Kong	In 1933 New Y	6.12E+16	[[{"name": "W	[[{"so_3166_1	14/12/2005	550000000	
Drama Romar	http://www.t	597	shipwreck ice	en	Titanic	84 years later	100025899	[[{"name": "Pa	[[{"so_3166_1	18/11/1997	1845334188	
Adventure Act	http://marvel	271110	civil war war	en	Captain Amer Following the		1.98E+16	[[{"name": "St	[[{"so_3166_1	27/4/2016	1153304495	
Villier Action Adventure Sci		44833	fight u.s. navy	en	Battleship	When makin	64928382	[[{"name": "Uu	[[{"so_3166_1	11/4/2012	303025485	

NORMALIZACIÓN

La normalización de base de datos consiste en el proceso de organizar los datos en una base de datos mediante una serie de reglas. La normalización existe para evitar la redundancia de datos, disminuir problemas de actualización y proteger la integridad de datos.

NORMALIZACIÓN

La normalización de base de datos consiste en el proceso de organizar los datos en una base de datos mediante una serie de reglas. La normalización existe para evitar la redundancia de datos, disminuir problemas de actualización y proteger la integridad de datos.

PRIMERA FORMA NORMAL

La primera forma normal significa que los datos están en un formato de entidad, lo que significa que se han cumplido las siguientes condiciones:

- Eliminar grupos repetidos en tablas individuales
- Crear una tabla independiente para cada conjunto de datos relacionados
- Identificar cada conjunto de relacionados con la clave principal

SEGUNDA FORMA NORMAL

Se deben 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

TERCERA FORMA NORMAL

La tercera forma normal comprueba las dependencias transitivas, eliminando campos que no dependen 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 tablas por medio de una clave externa.

DESARROLLO

```
background: url(../img/...);
background-size: 100vw 100vh;

}

.box{
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  width: 400px;
  padding: 40px;
  background: rgba(0, 0, 0, 0.5);
  box-sizing: border-box;
  box-shadow: 0 15px 25px rgba(0, 0, 0, 0.5);
  border-radius: 10px;
}

.box h2{
  margin: 0 0 30px;
  padding: 0;
  color: #fff;
  text-align: center;
}

.box h3{
  margin: 0 0 10px;
  padding: 0;
  color: #fff;
  text-align: center;
}

.box .input{
```

Análisis de columnas del Archivo Csv

Se reconocen 24 columnas y 4806 filas. Además se procede a indicar cada uno de los atributos hallados en la tabla universal movie_dataset y también las observaciones de cada uno de los atributos para indicar en que estados de encuentra cada columna ya sea que sus valores están correctos o poseen algún tipo de error o característica en específico con el fin de ir modelando nuestra modelo relacional

A screenshot of a Microsoft Excel spreadsheet titled 'movie_dataset - Excel'. The spreadsheet displays a table with 24 columns and 4806 rows of movie data. The columns are: review, popularity, production_c, production_c, release_date, revenue, runtime, spoken_language, status, and tagline. The data includes movie titles, production companies, release dates, revenues, runtimes, languages, and statuses. The interface shows the Excel ribbon with various tabs like 'Inicio', 'Insertar', 'Referencias', 'Datos', 'Formulas', 'Revisión', and 'Programador'. The status bar at the bottom indicates the file is 'movie_dataset - Excel' and the current cell is 'A1'.

Análisis de columnas del Archivo Csv en una tabla

En primer lugar encontramos un detalle en la columna Genres lo cual se observa al analizar este dato se observa que en genres existen campo multivaluados ya que existen más de un género y estos no están separados por ninguna comilla o símbolo sino que tiene un espacio entre cada tipo de género.



Atributo (columna)	Observaciones
Index	Esta columna no presenta errores
Budget	Existen varias filas en las que tiene valor de cero
Genres	En este caso tenemos una tabla multivaluada por valores que tienen espacios entre
Homepage	Algunos campos no tienen valor
Id	Todos los campos están completos y correctos
keywords	Contiene campos vacíos
Original_languages	Contiene todos los campos
Original_title	Contiene todos los campos
overview	En ciertas palabras contiene caracteres especiales
popularity	Contiene todos los campos correctos
Production_companies	Es una columna compuesta
Production_countries	Es una columna compuesta
Release_date	Contiene todos los campos
Revenue	Tiene campos con ceros
Runtime	Contiene campos con ceros
Spoken_languages	Es una columna compuesta
Status	Contiene todos los campos
Tagline	Algunos campos están vacíos
Vote_average	Contiene todos los campos
Vote_count	Contiene todos los campos
Cast	Contiene datos multivaluados
Crew	Es un campo compuesto
Director	En algunos tiene valores unidos
Title	Contiene todos los campos

Análisis de columnas del Archivo Csv en una tabla

Se procede a indicar cada uno de los atributos hallados en la tabla universal movie_dataset y también las observaciones de cada uno de los atributos para indicar en que estados de encuentra cada columna ya sea que sus valores están correctos o poseen algún tipo de error o característica en específico con el fin de ir modelando nuestra modelo relacional



genres
Drama Romance War
Comedy
Romance Comedy Music
Drama Action Crime
Western
Foreign Documentary
Romance Drama
Comedy
Adventure Horror Science Fiction
Adventure Action Thriller Science Fiction
Comedy Romance

Análisis de columnas del Archivo Csv en una tabla

A continuación encontramos en las siguientes columnas campos compuestos ya que como vemos existen datos separados por llaves. También al observar se ve que en ciertas partes se tienen caracteres codificados.



poken_languages
[{"iso_639_1": "en", "name": "English"}]
[{"iso_639_1": "en", "name": "English"}]
[{"iso_639_1": "en", "name": "English"}]
[{"iso_639_1": "en", "name": "English"}]
[{"iso_639_1": "en", "name": "English"}]
[{"iso_639_1": "en", "name": "English"}]
crew
[{"name": "Andrew Hyatt", "gender": 0, "department": "Directing", "job": "Director", "credit_id": "58"}, {"name": "Joel Paul Reisig", "gender": 0, "department": "Writing", "job": "Screenplay", "credit_id": "58"}, {"name": "Katie Cleary", "gender": 0, "department": "Production", "job": "Executive Producer", "credit_id": "58"}, {"name": "Irving Thalberg", "gender": 2, "department": "Production", "job": "Producer", "credit_id": "58"}]
[{"name": "Zoran Lisinac", "gender": 0, "department": "Directing", "job": "Director", "credit_id": "52f"}, {"name": "Larry Smith", "gender": 0, "department": "Camera", "job": "Director of Photography", "credit_id": "52f"}, {"name": "Morgan Schmidt", "gender": 0, "department": "Crew", "job": "Cinematography", "credit_id": "52f"}, {"name": "Tom Putnam", "gender": 0, "department": "Directing", "job": "Director", "credit_id": "52f"}]
director
George Miller
Louis Leterrier
Steven Spielberg
Alejandro Gonz\u00e1lez I\u00f1\u00e9rritu
David Soren
Gore Verbinski

ASIGNACIÓN DE ENTIDADES Y SUS ATRIBUTOS

Movie y sus atributos analizados para conocer que papel cumplen

Movie con sus respectivos atributos					
Entidad	Atributo	Dominio	Opcional	Multivalor	Comentario
Movie	Director	Varchar (30)	Not null	Verdadero	Director de la película
Movie	Relase_Date	Varchar (15)		Falso	Fecha de lanzamiento
Movie	Runtime	Int (5)	Not null	Falso	Tiempo de ejecución
Movie	Title	Int (20)	Not null	Falso	Título
Movie	Original_title	Int (20)		Falso	Título original
Movie	Original_language	Varchar (4)		Falso	Lenguaje Original
Movie	Tagline	Varchar (25)	Not null	Falso	Lema
Movie	Status	Varchar (15)	Not null	Falso	Estado
Movie	Budget	Int (12)	Not null	Falso	Presupuesto
Movie	Revenue	Int (15)	Not null	Falso	Ingresos
Movie	Vote_average	Int (3)		Falso	Promedio de votos
Movie	Vote_Count	Int (5)		Falso	Número de personas que opinaron
Movie	Homepage	Varchar (25)		Falso	Página de inicio
Movie	Index	Int (10)	Not null	Falso	Indice

ASIGNACIÓN DE ENTIDADES Y SUS ATRIBUTOS

Crew con sus respectivos atributos

Entidad	Atributo	Dominio	Opcional	Multivaluado	Comentario
Crew	Name	<u>Varchar</u> (50)	Not null	False	Nombre
Crew	Deparment	<u>Varchar</u> (10)	Not null	False	Departamento
Crew	Job	Varchar (25)	Not null	False	Trabajo
Crew	Credit id	Varchar (25)	Not null	False	Id de crédito
Crew	Id	Varchar (7)	Not null	False	Identificador

Entidad Production_companies y sus atributos

Entidad	Atributo	Dominio	Opcional	Multivaluado	Comentario
Prod_Companies	Name	Varchar (30)		Falso	Nombre
Prod_Companies	Id	Varchar (5)	Not null	Falso	Identificador

ASIGNACIÓN DE ENTIDADES Y SUS ATRIBUTOS

Spoken_languages y sus atributos respectivos.

Entidad	Atributo	Dominio	Opcional	Multivaluado	Comentario
Spoken_languages	Name	Varchar (15)		Falso	Nombre
Spoken_languages	ISO	Varchar (15)	Not null	Falso	Identificador

Production_countries y sus atributos

Entidad	Atributo	Dominio	Opcional	Multivaluado	Comentario
Production_countries	Name	Varchar (25)		Falso	Nombre
Production_countries	Id	Varchar (5)	Not null	Falso	Identificador

Cast y sus atributos respetivos

Entidad	Atributo	Dominio	Opcional	Multivaluado	Comentario
Cast	Rol	Varchar (15)		Falso	Papél
Cast	Name	Varchar (15)	Not null	Falso	Nombre
Cast	L_names	Varchar (15)		Falso	Segundo nombre

DIAGRAMA ENTIDAD RELACIÓN

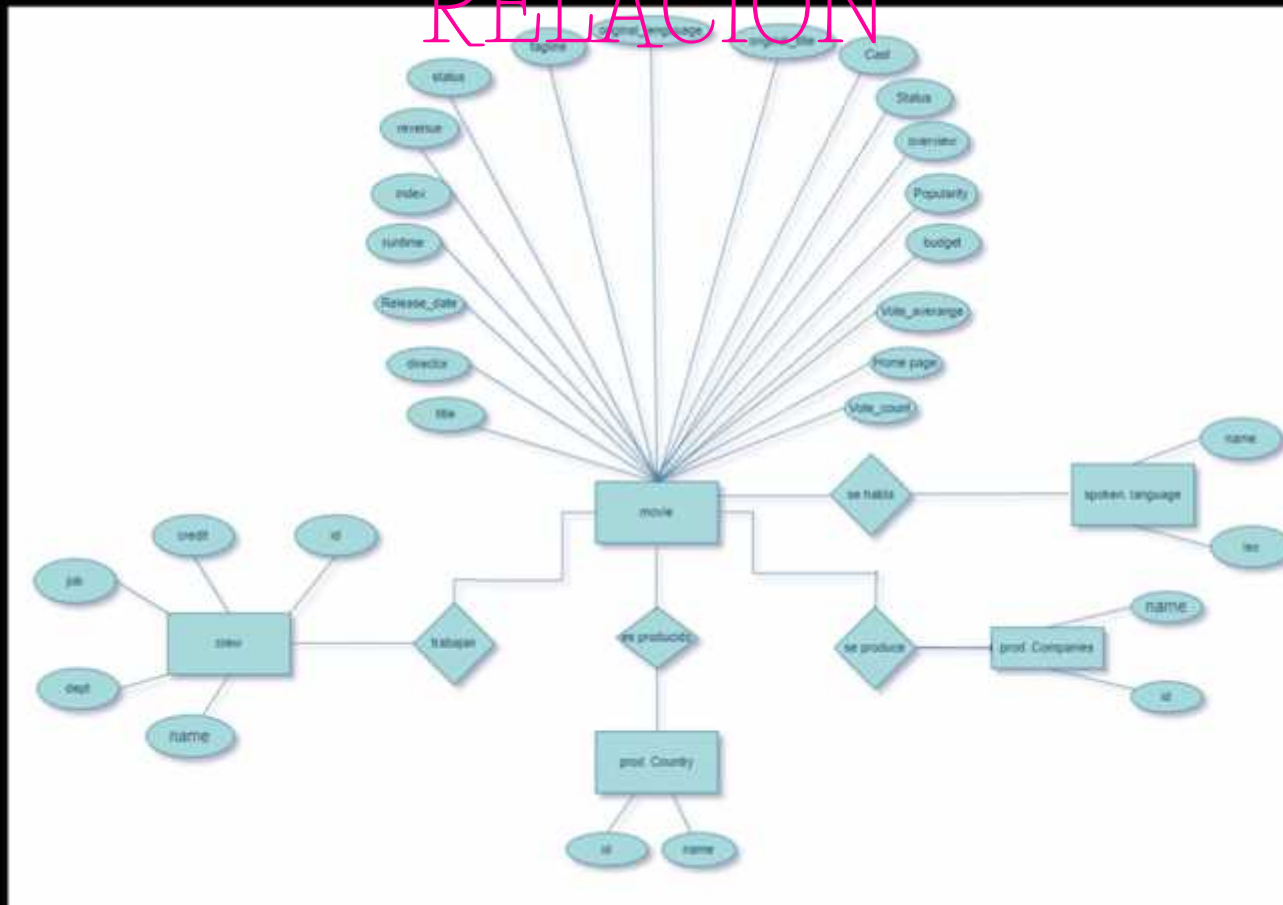


DIAGRAMA UML

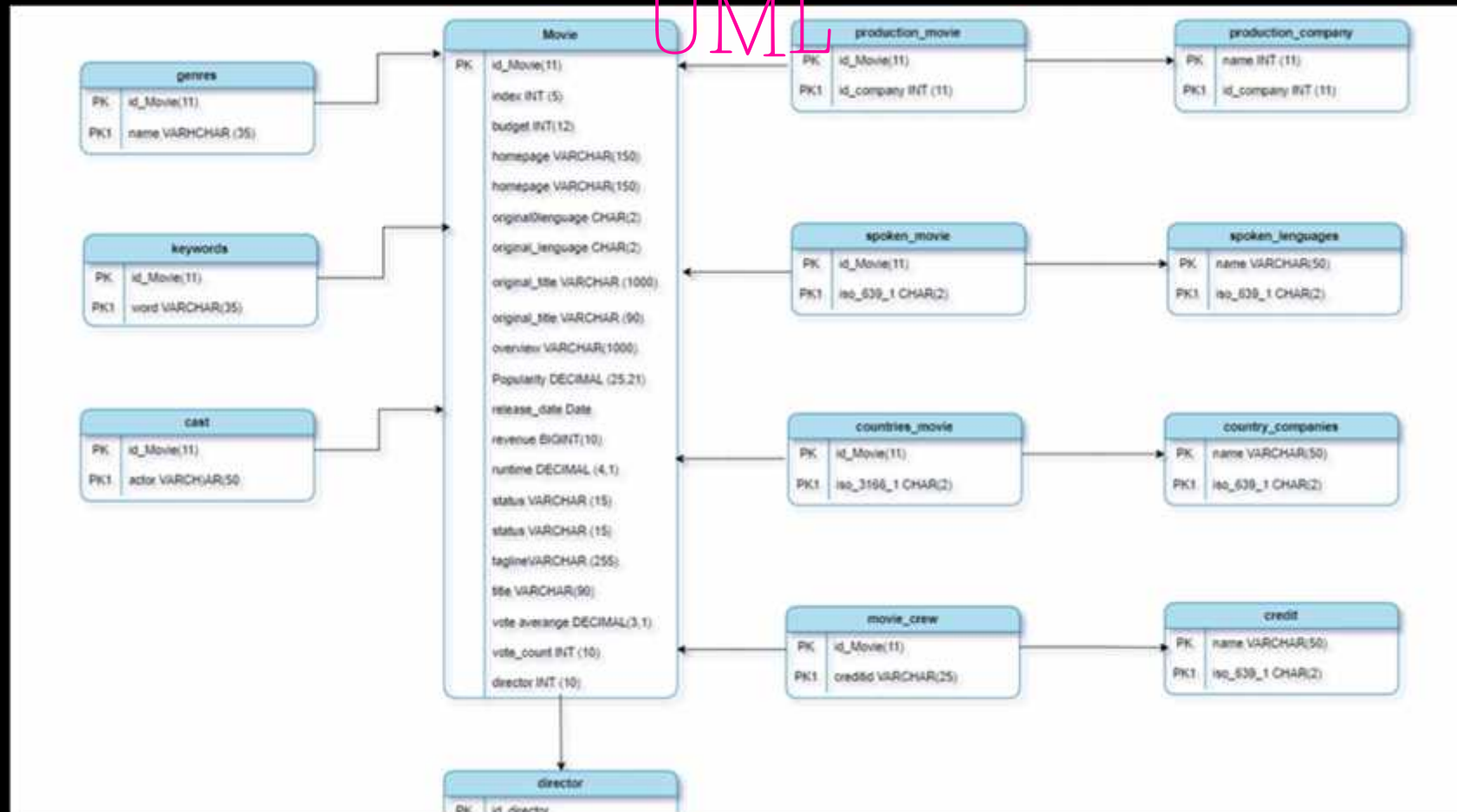
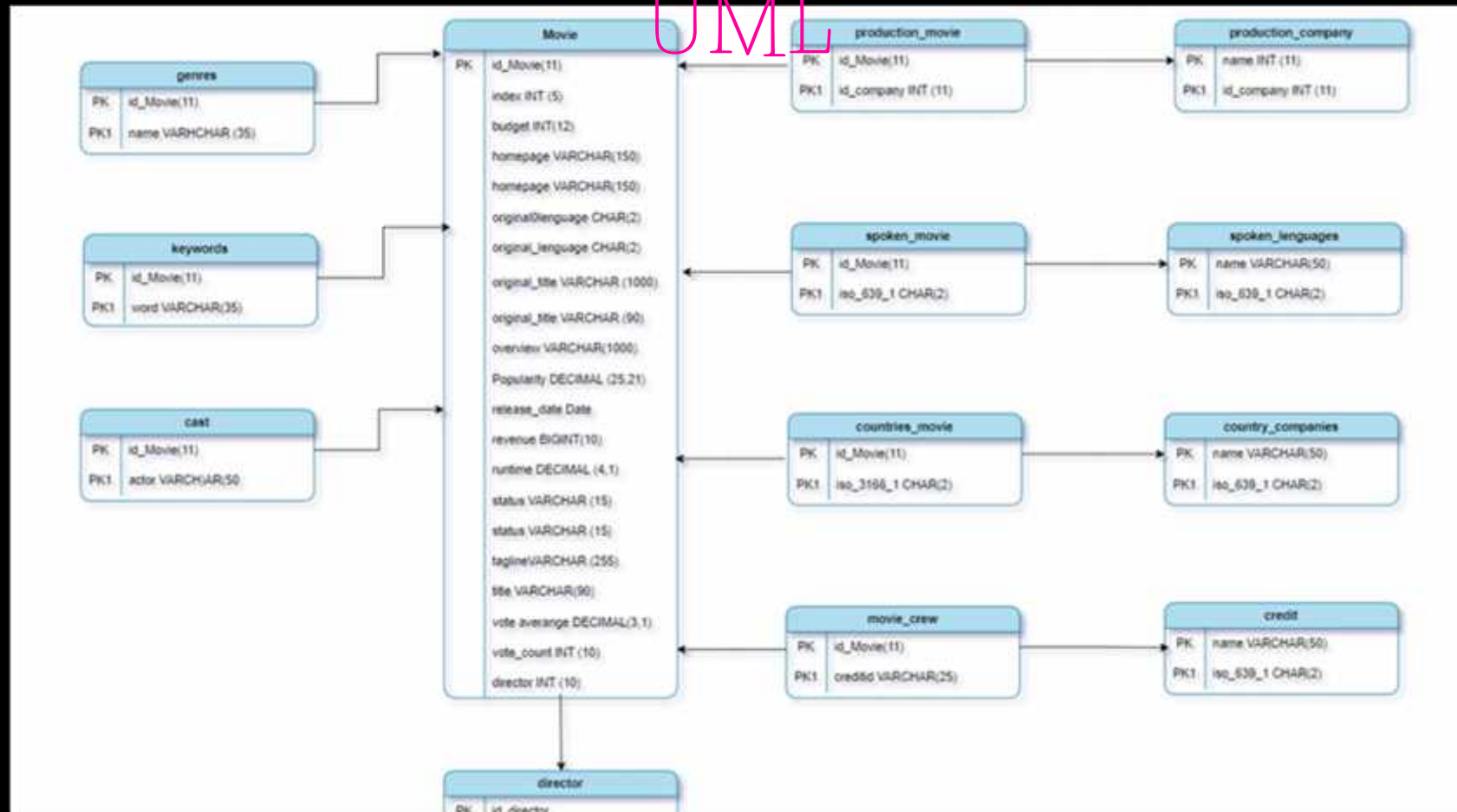
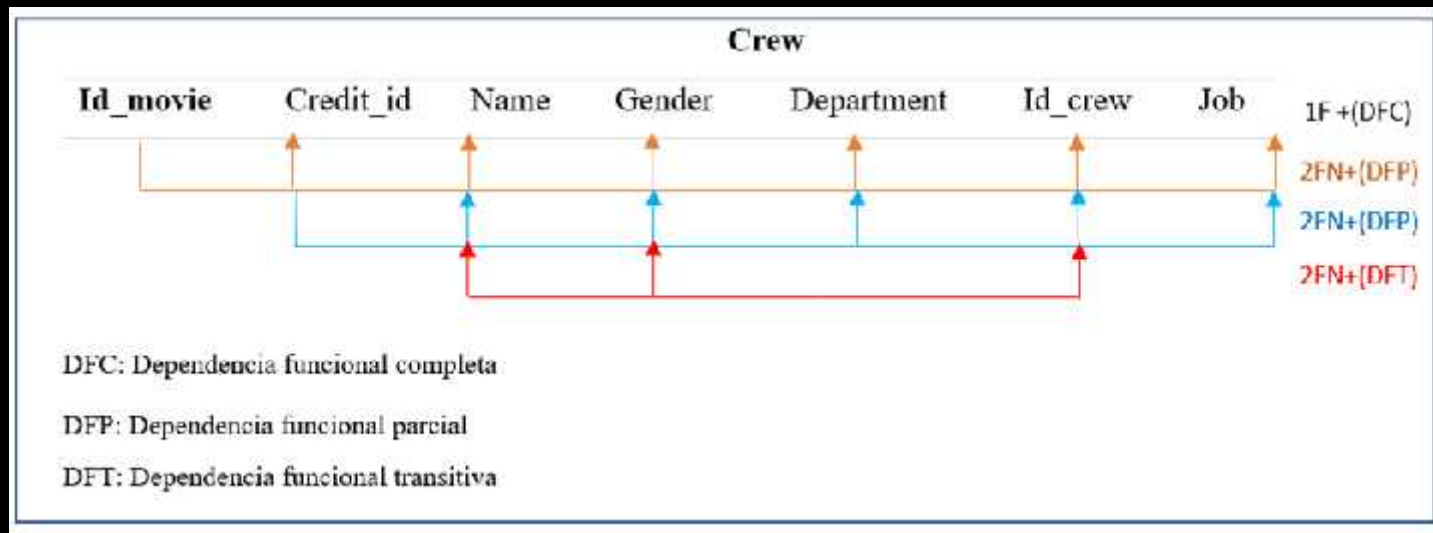
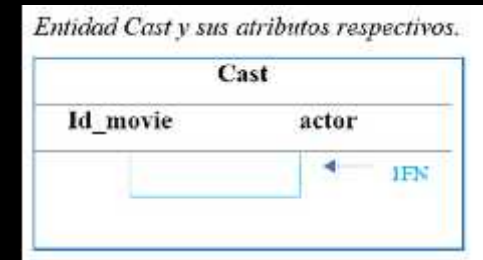
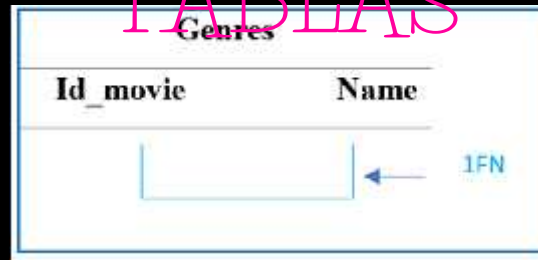
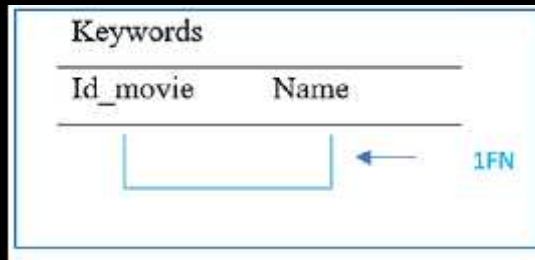


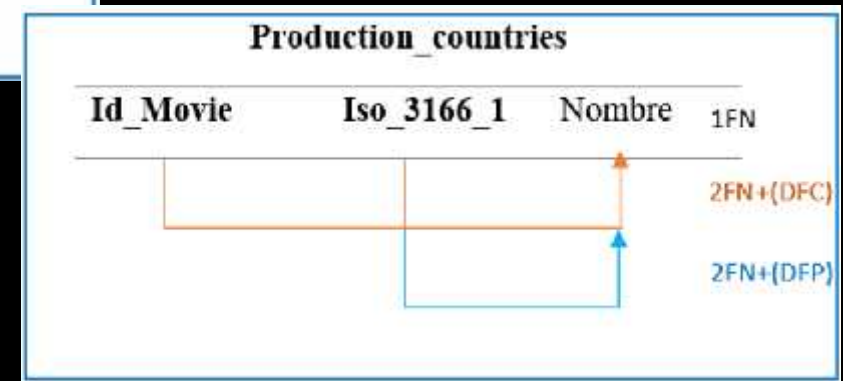
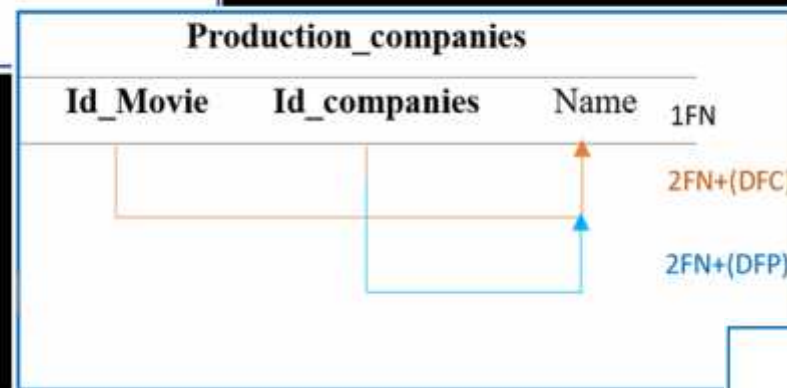
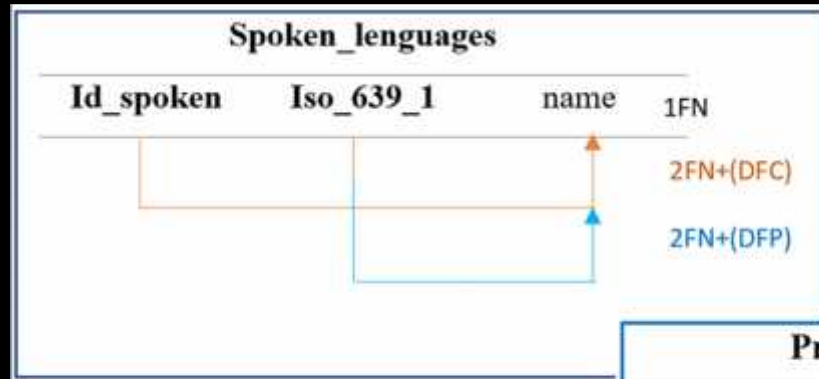
DIAGRAMA UML



NORMALIZACIÓN DE TABLAS

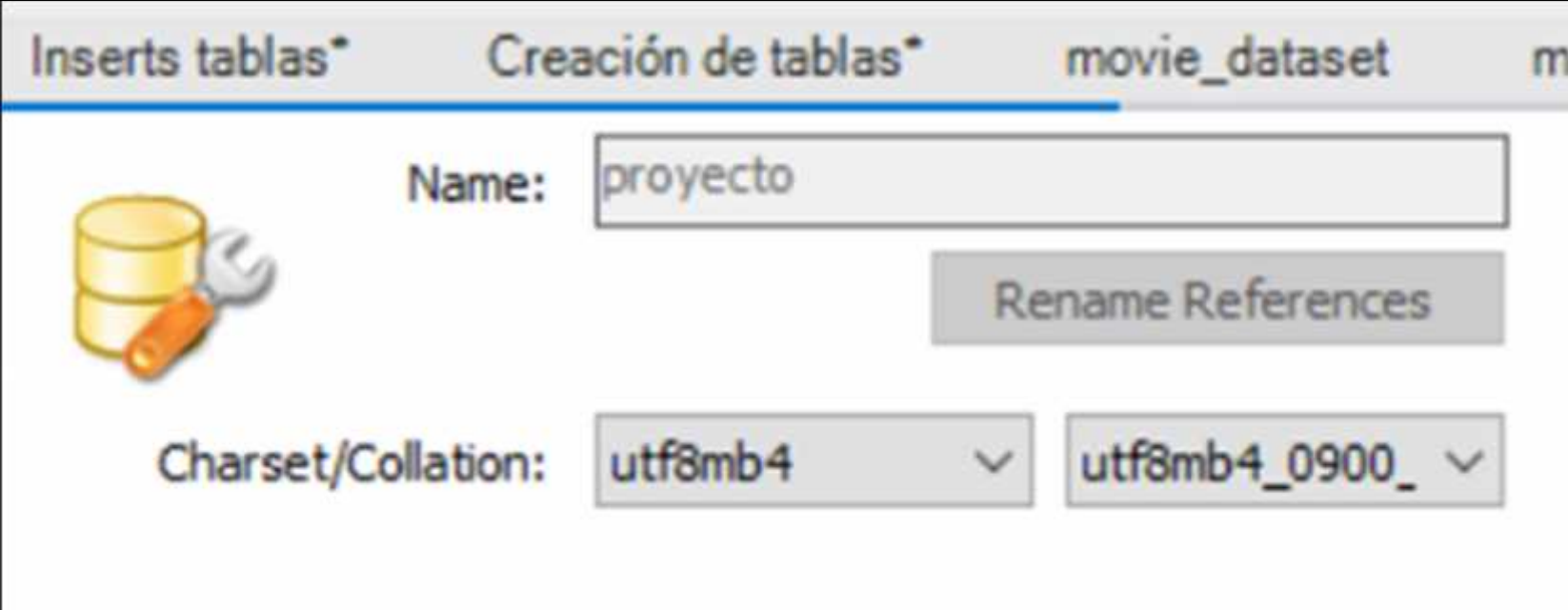


NORMALIZACIÓN DE TABLAS



IMPLEMENTACION DE LA BASE DE DATOS EN MYSQL


Creación de un nuevo esquema de base de datos



The image shows a screenshot of the MySQL Workbench 'Create Table' dialog box. The dialog has a tabbed interface with three tabs: 'Inserts tablas*', 'Creación de tablas*', and 'movie_dataset'. The 'Creación de tablas*' tab is currently selected. On the left side of the dialog, there is an icon representing a database cylinder and a wrench. The 'Name:' field contains the text 'proyecto'. To the right of this field is a button labeled 'Rename References'. Below the 'Name:' field, the 'Charset/Collation:' section shows two dropdown menus. The first dropdown is set to 'utf8mb4' and the second dropdown is set to 'utf8mb4_0900_'. The dialog is set against a dark background.

Inserts tablas* Creación de tablas* movie_dataset m

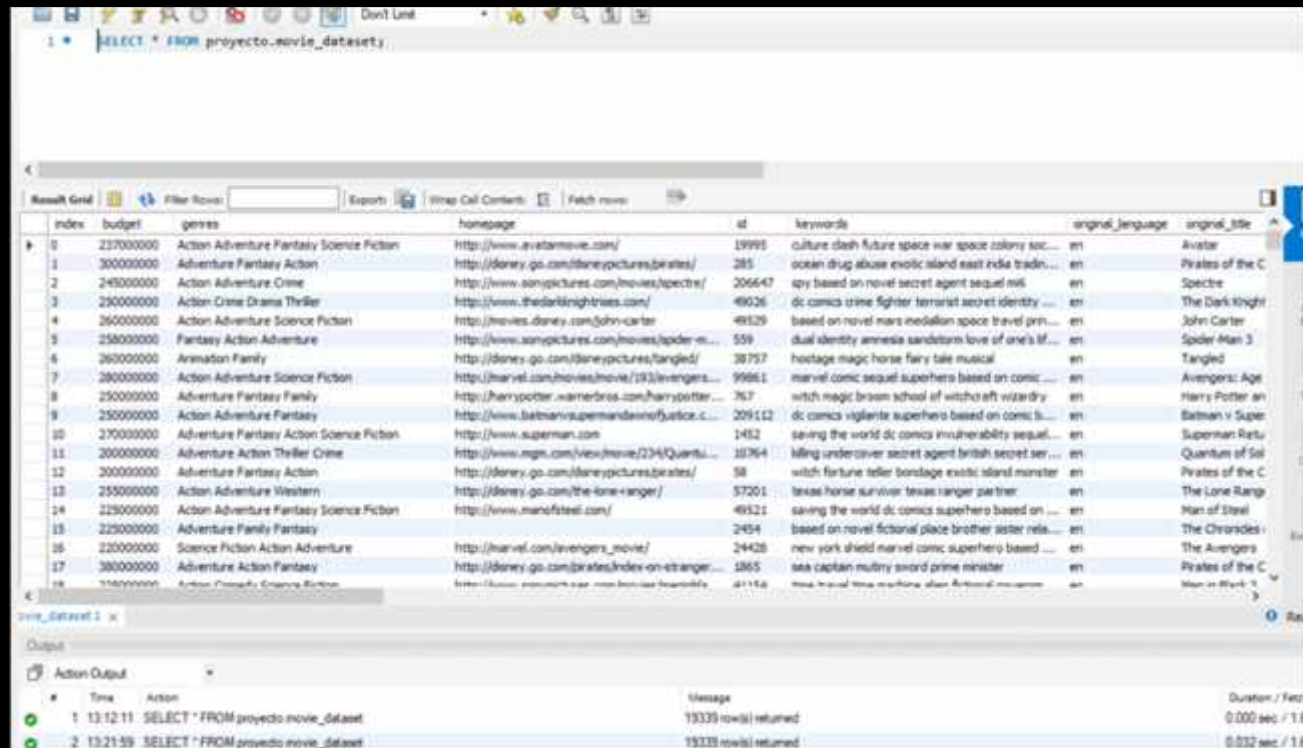
Name:



Charset/Collation:

IMPLEMENTACION DE LA BASE DE DATOS EN MYSQL

Importación del Csv a una tabla universal



The screenshot displays a MySQL database interface. At the top, a SQL query is entered: `SELECT * FROM proyecto.movie_dataset;`. Below the query editor, a table of movie data is shown. The table has columns: `index`, `budget`, `genres`, `homepage`, `id`, `keywords`, `original_language`, and `original_title`. The data includes movies like Avatar, Pirates of the Caribbean, Spectre, The Dark Knight, John Carter, Spider-Man 3, Tangled, Avengers: Age of Ultron, Harry Potter and the Deathly Hallows - Part 2, Batman v Superman: Dawn of Justice, Superman Returns, Quantum of Solace, Pirates of the Caribbean: On Stranger Tides, The Lone Ranger, Man of Steel, The Chronicles of Narnia: The Lion, the Witch and the Wardrobe, The Avengers, and Pirates of the Caribbean: The Curse of the Black Pearl.

Below the table, the 'Output' section shows the execution of the SQL query. It displays two messages:

#	Time	Action	Message	Duration / Percentage
1	13:12:11	SELECT * FROM proyecto.movie_dataset;	19339 rows returned	0.000 sec / 1.1%
2	13:21:59	SELECT * FROM proyecto.movie_dataset;	19339 rows returned	0.012 sec / 1.1%

```

DROP TABLE IF EXISTS movie_cleaned;
CREATE TABLE movie_cleaned AS
SELECT
    'index', budget, genres, homepage, id as id_movie, keywords, original_language, original_title, overview, popularity,
    production_companies, production_countries, release_date, revenue, runtime, spoken_languages, 'status',
    tagline, title, vote_average, vote_count, CONVERT(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
        REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
        REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
        REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
        REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
        REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
        REPLACE(REPLACE(cast, '1u00e1', 'á'), '1u00e5', 'ä'), '1u00e9', 'é'),
        '1u00eb', 'ë'), '1u00ed', 'í'), '1u00e9', 'à'), '1u00f1', 'ñ'), '1u00f8', 'ø'), '1u042', 'ð'),
        '1u043b', 'ñ'), '1u044f', 'a'), '1u0421', 'c'), '1u043d', 'h'), '1u0440', 'p'), '1u0433', 'r'),
        '1u044c', 'b'), '1u067e', 'u'), '1u06cc', 'j'), '1u0645', 'i'), '1u0627', 'l'), '1u0646', 'u'),
        '1u0640', 'c'), '1u062f', 'i'), '1u00e8', 'è'), '1u00f3', 'ó'), '1u0160', 'š'), '1u0197', 'č'),
        '1u0107', 'č'), '1u00f6', 'ö'), '1u00e4', 'ä'), '1u00e4', 'ö'), '1u00e4', 'c'), '1u0144', 'A'),
        '1u0219', 'A'), '1u00fc', 'ü'), '1u00c1', 'á'), '1u00ea', 'ä'), '1u00ea', 'ä'), '1u00e7', 'c'),
        '1u00dc', 'Ü'), '1u0159', 'P'), '1u00ds', 'ö'), '1u00fa', 'ü'), '1u010d', 'ž'), '1u010d', 'ž'),
        '1u00f0', 'ð'), '1u0219', 'p'), '1u00d3', 'ó'), '1u0110', 'ö'), '1u0161', 'č'), '1u0101', 'š'),
        '1u00c5', 'å'), '1u043b', 'a'), '1u014c', 'ö'), '1u016b', 'ö'), '1u014d', 'ö'), '1u015b', 'č'),
        '1u00ef', 'í'), '1u021b', 't'), '1u00c0b', 'g'), '1u00f4', 'ó'), '1u0301', 'í'), '1u00fb', 'ü'),
        '1u00fb', 'ü'), '1u0ed7', 'ö'), '1u01ecb', 'f'), '1u01e3', 'ä'), '1u01ebf', 'ä'), '1u015f', 's'),
        '1u015ea', 'ö'), '1u017e', 'ž'), '1u00c0', 'ä'), '1u0131', 'ä'), '1u013f', 'ä'), '1u01c1', 'ä'),
        '1u0039', 'z'), '1u00ee', 'é'), '1u00e0', 'a'), '1u00c9', 'é'), '1u00df', 'ß'), '1u015ea', 'ö') using utf8mb4) AS cast,

```

CREACIÓN DE TABLAS

```
-- Creación tabla movie;
DROP TABLE IF EXISTS movie;
CREATE TABLE movie (
  index_movie INTEGER ,
  budget INTEGER ,
  genres VARCHAR(255),
  homepage VARCHAR(255) ,
  id_movie INTEGER PRIMARY KEY ,
  keywords VARCHAR(255) ,
  original_language VARCHAR(255) ,
  original_title VARCHAR(255) ,
  overview VARCHAR(1000),
  popularity FLOAT ,
  release_date VARCHAR(255),
  revenue VARCHAR(255) not null,
  runtime VARCHAR(100),
  status_movie VARCHAR(255),
  tagline VARCHAR(255),
  title VARCHAR(255),
  vote_average FLOAT,
  vote_count INTEGER,
  cast VARCHAR(255),
  director VARCHAR(100)
);
```

Creación de la tabla movie juntos con sus atributos y clave primaria

CREACIÓN DE TABLAS

```
104 -- Creación tabla production_companies
105 • DROP TABLE IF EXISTS production_companies;
106 • CREATE TABLE production_companies(
107     name_comp VARCHAR(200),
108     id_production_companies INTEGER PRIMARY KEY
109 );
110 -- Creación tabla production_countries
111 • DROP TABLE IF EXISTS production_countries;
112 • CREATE TABLE production_countries (
113     iso_3166_1 VARCHAR(30) PRIMARY KEY NOT NULL,
114     name_production_countries VARCHAR(30) NOT NULL
115 );
116 -- Creación tabla spoken_languages
117 • DROP TABLE IF EXISTS spoken_languages;
118 • CREATE TABLE spoken_languages(
119     iso_639_1 VARCHAR(40) PRIMARY KEY NOT NULL,
120     "language" VARCHAR(30)
121 );
122 -- Creación tabla genres
123 • DROP TABLE IF EXISTS genres;
124 • CREATE TABLE genres(
125     id_genres VARCHAR(40) PRIMARY KEY NOT NULL,
126     nombre VARCHAR(50)
127 );
128 -- Creación tabla crew
129 • DROP TABLE IF EXISTS crew;
130 • CREATE TABLE crew (
131     idCrew INTEGER primary key NOT NULL,
132     name VARCHAR(400) NOT NULL,
133     gender INTEGER
134 );
```

creación de las tablas director, deparment, keywords, jobs, credit, para las cuales les asignamos sus atributos respectivamente además de la asignación de claves primarias. En la tabla credit se hace una relacion por lo cual se usa la clave foránea

CREACIÓN DE TABLAS

```
138 CREATE TABLE director(  
139     id_director VARCHAR(100) PRIMARY KEY NOT NULL,  
140     name_director VARCHAR(100)  
141 );  
142 DROP TABLE IF EXISTS departments;  
143 CREATE TABLE departments  
144     SELECT DISTINCT md5(department) AS id_department, department  
145     FROM tmp_crew;  
146  
147 DROP TABLE IF EXISTS keywords;  
148 CREATE TABLE keywords  
149     SELECT DISTINCT md5(keywords) AS id_keywords, keywords  
150     FROM tmp_key_words  
151     WHERE keywords != '';  
152  
153 DROP TABLE IF EXISTS jobs;  
154 CREATE TABLE jobs  
155     SELECT DISTINCT md5(job) AS idJob, job, md5(department) AS id_department  
156     FROM tmp_crew  
157     GROUP BY md5(job);  
158 DELETE FROM jobs WHERE idJob IS NULL ;  
159  
160 -- Creación tabla credit  
161 DROP TABLE IF EXISTS credit;  
162 CREATE TABLE credit(  
163     credit_id VARCHAR(100) primary key ,  
164     job VARCHAR(100),  
165     department VARCHAR(100),  
166     idCrew INTEGER,  
167     CONSTRAINT FK_credit_crew FOREIGN KEY (idCrew) REFERENCES crew(idCrew)  
168 );
```

Creación de las tablas director, deparment, keywords, jobs, credit, para las cuales les asignamos sus atributos respectivamente además de la asignación de claves primarias. En la tabla credit se hace una relacion por lo cual se usa la clave foránea

CREACIÓN DE TABLAS

```
177 -- Creación tabla movies_companies
178 DROP TABLE IF EXISTS movies_companies;
179 CREATE TABLE movies_companies(
180     id_movie Integer NOT NULL,
181     id_production_companies INTEGER NOT NULL,
182     primary key(id_movie,id_production_companies),
183
184     CONSTRAINT FK_id_movie_movie_companies
185         FOREIGN KEY(id_movie) REFERENCES movie(id_movie),
186
187     CONSTRAINT FK_id_movie_production_companies
188         FOREIGN KEY(id_production_companies) REFERENCES production_companies(id_production_companies),
189
190 );
191 -- Creación tabla movies_countries
192 DROP TABLE IF EXISTS movies_countries;
193 CREATE TABLE movies_countries(
194     id_movie INTEGER,
195     iso_3166_1 VARCHAR(30) NOT NULL,
196     primary key(id_movie,iso_3166_1),
197     CONSTRAINT FK_id_movies_movies_countries
198         FOREIGN KEY(id_movie) REFERENCES movie(id_movie),
199
200     CONSTRAINT FK_id_production_countries_movies_countries
201         FOREIGN KEY(iso_3166_1) REFERENCES production_countries(iso_3166_1),
202 );
203
204
```

Creación de las tablas `movies_companies`, `movies_countries` las cuales representan las relaciones que tienen cada tabla por lo cual se usa una clave foránea y a que entidad se hace referencia.

CREACIÓN DE TABLAS

```
207 * DROP TABLE IF EXISTS movies_languages;
208 * CREATE TABLE movies_languages(
209     id_movie Integer NOT NULL,
210     iso_639_1 VARCHAR(40) NOT NULL,
211     primary key(id_movie,iso_639_1),
212
213     CONSTRAINT FK_id_movies_languages
214         FOREIGN KEY(id_movie) REFERENCES movie(id_movie),
215     CONSTRAINT FK_id_movies_spoken_languages
216         FOREIGN KEY(iso_639_1) REFERENCES spoken_languages(iso_639_1)
217 );
218 -- Creación tabla genero_movie
219 * DROP TABLE IF EXISTS genres_movie;
220 * CREATE TABLE genres_movie(
221     id_movie Integer NOT NULL,
222     id_genres VARCHAR(100) NOT NULL,
223     primary key(id_movie,id_genres),
224     CONSTRAINT FK_id_movies_genres_movie
225         FOREIGN KEY(id_movie) REFERENCES movie(id_movie),
226     CONSTRAINT FK_id_movies_genres_genres
227         FOREIGN KEY(id_genres) REFERENCES genres(id_genres)
228 );
229 -- Creación tabla movies_credit
230 * DROP TABLE IF EXISTS movies_credit;
231 * CREATE TABLE movies_credit(
232     id_Movie INTEGER,
233     credit_id VARCHAR(100),
234     PRIMARY KEY (id_movie, credit_id),
235     CONSTRAINT FK_movie_credit_idMovie FOREIGN KEY (id_movie) REFERENCES movie(id_Movie),
```

Se crearon las tablas movies_languages, genres_movies, movies_credit las cuales representan las relaciones que tienen cada tabla por lo cual se usa una clave foránea y a que valor se hace referencia.

CREACIÓN DE TABLAS

```
207 * DROP TABLE IF EXISTS movies_languages;
208 * CREATE TABLE movies_languages(
209     id_movie Integer NOT NULL,
210     iso_639_1 VARCHAR(40) NOT NULL,
211     primary key(id_movie,iso_639_1),
212
213     CONSTRAINT FK_id_movies_languages
214         FOREIGN KEY(id_movie) REFERENCES movie(id_movie),
215     CONSTRAINT FK_id_movies_spoken_languages
216         FOREIGN KEY(iso_639_1) REFERENCES spoken_languages(iso_639_1)
217 );
218 -- Creación tabla genero_movie
219 * DROP TABLE IF EXISTS genres_movie;
220 * CREATE TABLE genres_movie(
221     id_movie Integer NOT NULL,
222     id_genres VARCHAR(100) NOT NULL,
223     primary key(id_movie,id_genres),
224     CONSTRAINT FK_id_movies_genres_movie
225         FOREIGN KEY(id_movie) REFERENCES movie(id_movie),
226     CONSTRAINT FK_id_movies_genres_genres
227         FOREIGN KEY(id_genres) REFERENCES genres(id_genres)
228 );
229 -- Creación tabla movies_credit
230 * DROP TABLE IF EXISTS movies_credit;
231 * CREATE TABLE movies_credit(
232     id_Movie INTEGER,
233     credit_id VARCHAR(100),
234     PRIMARY KEY (id_movie, credit_id),
235     CONSTRAINT FK_movie_credit_idMovie FOREIGN KEY (id_movie) REFERENCES movie(id_Movie),
```

Se crearon las tablas movies_languages, genres_movies, movies_credit las cuales representan las relaciones que tienen cada tabla por lo cual se usa una clave foránea y a que valor se hace referencia.

CREACIÓN DE TABLAS

```
1 • DROP PROCEDURE IF EXISTS Json2Relational_production_countries;
2 DELIMITER //
3 • CREATE PROCEDURE Json2Relational_production_countries()
4 BEGIN
5     DECLARE a INT Default 0 ;
6     DROP TABLE IF EXISTS tmp_production_countries ;
7     CREATE TABLE tmp_production_countries
8         (id_movie INT, name VARCHAR (200), iso_3166_1 VARCHAR (200));
9     simple_loop: LOOP
10     INSERT INTO tmp_production_countries (id_movie, name, iso_3166_1)
11         SELECT id_movie,
12             JSON_EXTRACT(CONVERT(production_countries using utf8mb4), CONCAT("$[",a,"].name")) AS name,
13             JSON_EXTRACT(CONVERT(production_countries using utf8mb4), CONCAT("$[",a,"].iso_3166_1")) AS iso_3166_1
14         FROM movie_cleaned mc
15         WHERE id_movie IN (SELECT id_movie FROM movie_cleaned WHERE a <= JSON_LENGTH (production_countries));
16     SET a=a+1;
17     IF a=6 THEN
18         LEAVE simple_loop;
19     END IF;
20     END LOOP simple_loop;
21     DELETE FROM tmp_production_countries WHERE name IS NULL ;
22 END //
23 DELIMITER ;
24 • Call Json2Relational_production_countries();
```

A continuación de proceder a crear los Json para crear las tablas temporales tmp_cast, tmp_crew, tmp_genres , tmp_production_companies, tmp_production_countries ,tmp_spoken_languages , este proceso se lo creo con fin de extraer los datos de las tablas con campos multivaluados o compuestos para luego almacenar cada columna en una tabla temporal para luego poder crear las tablas definitivas.

CREACIÓN DE TABLAS

```
1  * DROP PROCEDURE IF EXISTS Json2Relational_production_countries;
2  DELIMITER //
3  * CREATE PROCEDURE Json2Relational_production_countries()
4  BEGIN
5      DECLARE a INT Default 0 ;
6      DROP TABLE IF EXISTS tmp_production_countries ;
7      CREATE TABLE tmp_production_countries
8          (id_movie INT, name VARCHAR (200), iso_3166_1 VARCHAR (200));
9      simple_loop: LOOP
10         INSERT INTO tmp_production_countries (id_movie, name, iso_3166_1)
11             SELECT id_movie,
12                 JSON_EXTRACT(CONVERT(production_countries using utf8mb4), CONCAT("$[",a,"].name")) AS name,
13                 JSON_EXTRACT(CONVERT(production_countries using utf8mb4), CONCAT("$[",a,"].iso_3166_1")) AS iso_3166_1
14             FROM movie_cleaned mc
15             WHERE id_movie IN (SELECT id_movie FROM movie_cleaned WHERE a <= JSON_LENGTH (production_countries));
16         SET a=a+1;
17         IF a=6 THEN
18             LEAVE simple_loop;
19         END IF;
20     END LOOP simple_loop;
21     DELETE FROM tmp_production_countries WHERE name IS NULL ;
22     END //
23 DELIMITER ;
24 * Call Json2Relational_production_countries();
```

CREACIÓN DE TABLAS

```
DROP PROCEDURE IF EXISTS Json2Relational_production_companies ;
DELIMITER //
CREATE PROCEDURE Json2Relational_production_companies()
BEGIN
    DECLARE a INT Default 0 ;
    DROP TABLE IF EXISTS tmp_production_companies ;
    CREATE TABLE tmp_production_companies (id_movie INTEGER, id_production_company INT, name_company VARCHAR (100) );
    simple_loop: LOOP
        INSERT INTO tmp_production_companies (id_movie, name_company,id_production_company)
        SELECT id_movie,
            JSON_EXTRACT(mc.production_companies , CONCAT('$[',a,']-name')),
            JSON_EXTRACT(mc.production_companies, CONCAT('$[',a,']-id'))
        FROM movie_cleaned mc
        WHERE id_movie IN (SELECT id_movie FROM movie_cleaned WHERE a <= JSON_LENGTH (production_companies));
        SET a=a+1;
        IF a=10 THEN
            LEAVE simple_loop;
        END IF;
    END LOOP simple_loop;
    DELETE FROM tmp_production_companies WHERE id_production_company IS NULL ;
END //
DELIMITER ;
Call Json2Relational_production_companies();
```


CREACIÓN DE TABLAS

```
74
75 # DROP PROCEDURE IF EXISTS json2Relational_crew ;
76 DELIMITER //
77 # CREATE PROCEDURE json2Relational_crew()
78 BEGIN
79     DECLARE a INT Default 0 ;
80     DROP TABLE IF EXISTS tmp_crew;
81     CREATE TABLE tmp_crew
82     (id_movie INT, id_crew INT, job VARCHAR (200), name VARCHAR (400), gender INT, credit_id VARCHAR (50), department VARCHAR (50) );
83     simple_loop: LOOP
84         INSERT INTO tmp_crew (id_movie, id_crew, job, name, gender, credit_id, department)
85         SELECT id_movie,
86             JSON_EXTRACT(CONVERT(crew using utf8mb4), CONCAT("$",a,".id")) AS id_crew,
87             JSON_EXTRACT(CONVERT(crew using utf8mb4), CONCAT("$",a,".job")) AS job,
88             JSON_EXTRACT(CONVERT(crew using utf8mb4), CONCAT("$",a,".name")) AS name,
89             JSON_EXTRACT(CONVERT(crew using utf8mb4), CONCAT("$",a,".gender")) AS gender,
90             JSON_EXTRACT(CONVERT(crew using utf8mb4), CONCAT("$",a,".credit_id")) AS credit_id,
91             JSON_EXTRACT(CONVERT(crew using utf8mb4), CONCAT("$",a,".department")) AS department
92         FROM movie_cleaned mc
93         WHERE id_movie IN (SELECT id_Movie FROM movie_cleaned WHERE a <= JSON_LENGTH (crew) );
94     SET a=a+1;
95     IF a=436 THEN
96         LEAVE simple_loop;
97     END IF;
98     END LOOP simple_loop;
99     DELETE FROM tmp_crew WHERE id_crew IS NULL ;
100 END //
101 DELIMITER ;
102 # Call json2Relational_crew();
```

CREACIÓN DE TABLAS

```
109 * DROP PROCEDURE IF EXISTS Json2Relational_Key_words ;
110 DELIMITER //
111 * CREATE PROCEDURE Json2Relational_Key_words()
112 BEGIN
113     DECLARE a INT Default 0 ;
114
115     -- crear tabla temporal para almacenar datos de Key_words que están en JSON
116     DROP TABLE IF EXISTS tmp_Key_Words ;
117     CREATE TABLE tmp_Key_Words (id_movie INT, keyWords VARCHAR (100) );
118
119     -- ciclo repetitivo para ir cargando datos desde el arreglo JSON hacia la tabla temporal
120     simple_loop: LOOP
121         -- cargando datos del objeto JSON en la tabla temporal
122         INSERT INTO tmp_Key_Words (id_movie, keyWords)
123         SELECT id_Movie,
124             JSON_EXTRACT(CONCAT('[',REPLACE (REPLACE (keywords, '"', ' '), ' ', '"', ''), '", CONCAT("$[",a,""]")) AS keyWords
125         FROM movie_cleaned mc ;
126         SET a=a+1;
127         IF a=20 THEN
128             LEAVE simple_loop;
129         END IF;
130     END LOOP simple_loop;
131     -- limpieza de registros nulos
132     DELETE FROM tmp_Key_Words WHERE keyWords IS NULL ;
133 END //
134 DELIMITER ;
135 * Call Json2Relational_Key_words();
```


CREACIÓN DE TABLAS

```
138 * DROP PROCEDURE IF EXISTS Json2Relational_genres ;
139 DELIMITER //
140 * CREATE PROCEDURE Json2Relational_genres()
141 BEGIN
142     DECLARE a INT Default 0 ;
143     DROP TABLE IF EXISTS tmp_genres ;
144     CREATE TABLE tmp_genres (id_movie INT not null, idGe VARCHAR(100),genre VARCHAR (100) );
145     simple_loop: LOOP
146         INSERT INTO tmp_genres (id_movie,idGe,genre)
147         SELECT * FROM (
148             SELECT id_movie as id_movie,IDS(REPLACE(JSON_EXTRACT(CONCAT('["', REPLACE(REPLACE (genres, ' ', '"'),
149                 'Science',"Fiction", 'Science Fiction'), '"']'), CONCAT("$[",a,"]")), "", "")),
150             REPLACE(JSON_EXTRACT(CONCAT('["', REPLACE(REPLACE (genres, ' ', '"'),
151                 'Science',"Fiction", 'Science Fiction'), '"']'), CONCAT("$[",a,"]")), "", "")) AS genre
152             FROM movie_cleaned ) t
153         WHERE genre != "";
154         SET a=a+1;
155         IF a=6 THEN
156             LEAVE simple_loop;
157         END IF;
158     END LOOP simple_loop;
159     DELETE FROM tmp_genres WHERE genre IS NULL;
160 END //
161 DELIMITER ;
162 * Call Json2Relational_genres();
```

CREACIÓN DE TABLAS

```
164 -- BORRAR SI existe una version anterior (re-crear el procedimiento)
165 • DROP PROCEDURE IF EXISTS Json2Relational_Cast ;
166 DELIMITER //
167 • CREATE PROCEDURE Json2Relational_Cast()
168 BEGIN
169     DECLARE a INT Default 0 ;
170     -- crear tabla temporal para almacenar datos de Cast que estén en JSON
171     DROP TABLE IF EXISTS tmp_Cast ;
172     CREATE TABLE tmp_Cast (id_movie INT, "name" VARCHAR (100) );
173     -- ciclo repetitivo para ir cargando datos desde el arreglo JSON hacia la tabla temporal
174     simple_loop: LOOP
175     -- cargando datos del objeto JSON en la tabla temporal
176         INSERT INTO tmp_Cast (id_movie, "name")
177         SELECT id_movie,
178             JSON_EXTRACT(CONVERT(cast using utf8mb4), CONCAT("$[",a,"].id")) AS "name"
179         FROM movie_cleaned mc;
180         SET a=a+1;
181     IF a=6 THEN
182         LEAVE simple_loop;
183     END IF;
184     END LOOP simple_loop;
185     DELETE FROM tmp_Cast WHERE "name" IS NULL ;
186 END //
187 DELIMITER ;
188 • Call Json2Relational_Cast();
189
```

Tablas generadas en el esquema



A screenshot of a database schema diagram. It features a vertical list of 25 tables, each preceded by a small icon consisting of a blue triangle and a grey square. The tables are listed in the following order from top to bottom: crew, departments, director, genres, genres_movie, jobs, keywords, movie, movie_cleaned, movie_dataset, movie_keywords, movies_companies, movies_countries, movies_credit, movies_languages, production_companies, production_countries, spoken_languages, tmp_cast, tmp_crew, tmp_genres, tmp_key_words, tmp_production_companies, tmp_production_countries, and tmp_spoken_languages.

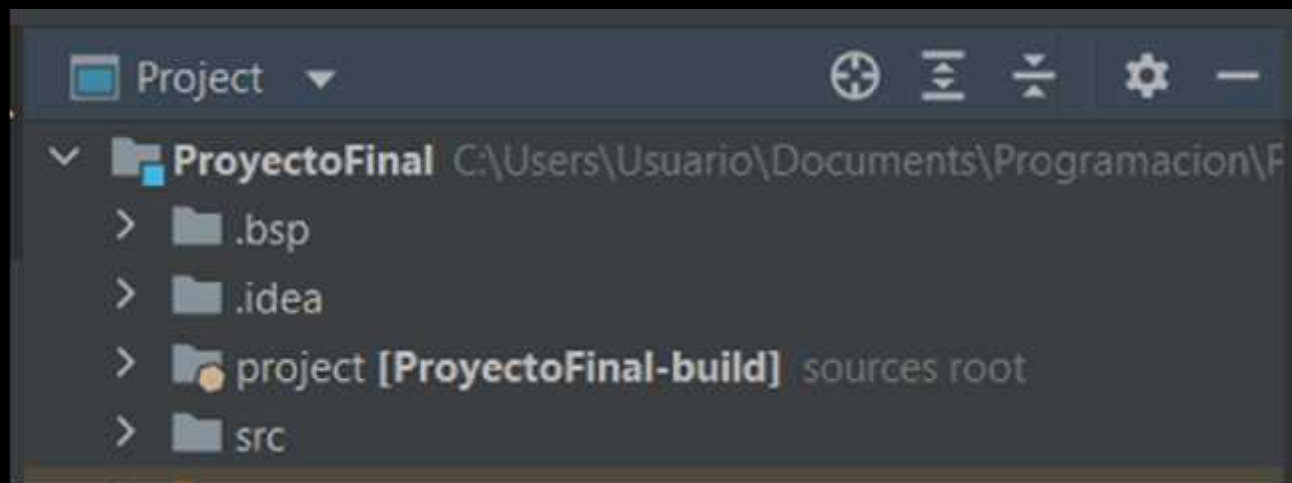
- crew
- departments
- director
- genres
- genres_movie
- jobs
- keywords
- movie
- movie_cleaned
- movie_dataset
- movie_keywords
- movies_companies
- movies_countries
- movies_credit
- movies_languages
- production_companies
- production_countries
- spoken_languages
- tmp_cast
- tmp_crew
- tmp_genres
- tmp_key_words
- tmp_production_companies
- tmp_production_countries
- tmp_spoken_languages

Borrado de tablas temporales

```
DROP TABLE IF EXISTS tmp_cast ;  
DROP TABLE IF EXISTS tmp_crew ;  
DROP TABLE IF EXISTS tmp_genres ;  
DROP TABLE IF EXISTS tmp_production_companies ;  
DROP TABLE IF EXISTS tmp_production_countries ;  
DROP TABLE IF EXISTS tmp_spoken_languages ;
```

GENERACIÓN DE CONSULTAS MEDIANTE PROGRAMACIÓN

CREACIÓN DE UN NUEVO PROYECTO



GENERACIÓN DE CONSULTAS MEDIANTE PROGRAMACIÓN

Importación de las librerías necesarias para realizar consultas con el archivo Csv

```
1  name := "PrivateFinal"
2
3  version := "0.1"
4
5  //scalaVersion := "2.13.8"
6
7  val kantanCsvVersion = "0.5.9"
8  //val kantanCsvVersion = "0.6.3"
9  libraryDependencies ++= Seq(
10
11    // Core library, included automatically if any other module is imported.
12    "com.hprinaudo" %% "kantan.csv" % kantanCsvVersion,
13
14    // Automatic type class instances derivation.
15    "com.hprinaudo" %% "kantan.csv-generic" % kantanCsvVersion,
16
17    // Provides instances for enumeratum types.
18    "com.hprinaudo" %% "kantan.csv-enumeratum" % kantanCsvVersion,
19
20    "com.hprinaudo" %% "kantan.csv-jackson" % kantanCsvVersion,
21
22    "com.typesafe.play" %% "play-json" % "2.9.2",
23    //SLICK
24
25    "com.typesafe.slick" %% "slick" % "3.3.2",
26    "ch.qos.logback" % "logback-classic" % "1.2.3",
27    "com.typesafe.slick" %% "slick-hikaricp" % "3.3.2",
28    "mysql" % "mysql-connector-java" % "8.0.23"
```





CONSULTA

Consultar las películas que salieron en el año 2000



CONSULTA

Consultar las películas que salieron en el año 2000

```
34
35     var filtered = iterator.filter((_,exists) => s.release_date.contains(anio.toString())).map(_.map(
36         s => list(s.index, s.original_title, s.release_date)))
37
38     //Nota esta parte es como una condicionall, exists(s => s.release_date.contains(anio.toString())) por que en todo de que
39     //no se cumple no va a filtrar ningun dato por eso filtra solo los datos que exista una movia y las movias que en se
40     //campo release_date contenga la cadena 2000 que es el año de salida
41
42     //Luego se presenta una sentencia de como va a salir en la consola la presentacion:
43     println("==== Resultado de consulta: ==== ")
44     printf("Las películas que fueron lanzadas en el año de %s fueron:\n", anio)
45     println("====\n")
46
47     //El filteras son los datos filtrados obtenidos en la linea 35, es decir los datos que si cumplieran con la condicional explicada en la linea 35;
48     //Luego se hace un foreach por que del filtered se recupera una lista y en caso de que existan varias películas que cumplan la
49     //condicional de que en el release_date contenga el "2000" se necesitara recorrer la lista que se le hace con un foreach
50     //s => el s le permite recuperar las listas de la linea 44
51     //s.right.get(0) permite recuperar el datos que se encuentran en la lista en la posicion 0
52     //Nota como tenemos las listas con los campos de la linea 44 (index,original_title,release_date) se cuenta desde 0,1,2
53     //por eso para recuperar el index con s.right.get(0); original_title con s.right.get(1) y release_date con s.right.get(2)
54     //las (0),(1),(2) son las posiciones de los datos que se encuentran en la lista de la linea 44
55     filtered.foreach(s => printf("%s\t%s\t%s\n",s.right.get(0),s.right.get(1),s.right.get(2)))
56     println("====\n")
57
58     //CREACION DEL CSV
59
60     //En campo se guarda de los datos filtrados que es una lista le damos un nombre y guardamos en una lista el index,el original_title y el release_date
61     val campos = filtered.map(s => list(s.right.get(0), s.right.get(1), s.right.get(2))) //Campos recupera una lista de lista de String
62     //Creamos uno de la libreria de la linea 4(java.io.File) para el new file
63     //write.csv permite escribir el csv y recibe una lista de lista de string, la cual es la variable campos de la linea 60.
64     //File.withHeader(.....) es el encabezado de los campos del csv es decir el csv se va a generar con el encabezado (.....)
65     //y luego se van a insertar los campos, el cual es la variable que tiene una lista de lista de string
66
67     val File("C:\\Users\\XX\\AppData\\Local\\Temp\\project\\final\\project\\final\\data\\data.csv") writeCsv(list(campos),File.withHeader("index,original_title,release_date"))
```



CONSULTA

Consultar las películas que salieron en el año 2000

```
----< Consultar las películas que salieron en el año 2000 >--  
<<<<< Resultado de ejecución: >>>>>  
Las películas que fueron lanzadas en el año: 2000 fueron:  
Index      Título                      Fecha de lanzamiento  
2148      High Fidelity                2000-03-17  
3613      Digimon: The Movie           2000-03-17  
4831      Antonio                      2000-02-03  
350       Charlie's Angels            2000-11-02  
896       He, Myself & Irene          2000-06-22  
2844      The Little Vampire           2000-10-27  
2292      Thomas and the Magic Railroad 2000-07-2  
3265      Shadow of the Vampire        2000-05-15  
3257      American Psycho              2000-04-13  
873       Shaft                        2000-06-15  
1043      Miss Congeniality            2000-12-14  
1090      All the Pretty Horses        2000-12-11  
2001      The Crew                     2000-08-21  
2255      The Yards                    2000-04-27  
2564      Love & Basketball            2000-04-21  
680       U-571                        2000-04-20  
1964      The Whole Nine Yards         2000-02-18  
2592      Highlander: Endgame           2000-09-01  
961       The Beach                    2000-02-11  
4111      Dinner Rush                  2000-09-01  
2735      Dude, Where's My Car?        2000-12-15  
4367      The Broken Hearts Club: A Romantic Comedy  
471       Little Nicky                  2000-11-10  
480       Battlefield Earth              2000-05-10
```



CONSULTA

Consultar las películas que tienen el genero terror

```
package ConsultasCSV

// Importamos varias: Principalmente la libreria hankm esta enfocada a scala para el manejo de case en programación
// Le libreria java.io.File nos permite generar o crear un archivo desde código, esta libreria es útil
// Al objeto que se va a generar el csv habido a que crea un archivo .csv en la línea:
object Preguntas extends App { // se genera el objeto pregunta 1 a extend App permite que el código escrito en este objeto pueda ser ejecutado dentro de el
// Consultar que películas pertenecen al género de terror
println("====> Consultar que películas pertenecen al género de terror <====")
// readData nos permite obtener la ubicación donde se encuentra el csv con el que vamos a trabajar, en este caso es /movie_dataset.csv
val readData: java.net.URI = getClass.getResource("data-movie_dataset.csv")
// case class Movie nos permite identificar cada una de las columnas que tiene el csv como es index, budget, .....
case class Movie(
  index: String, budget: String, genres: String,
  homepage: String, id: String, keywords: String, original_language: String, original_title: String,
  overview: String, popularity: Double, production_companies: String, production_countries: String,
  release_date: String, revenue: String, runtime: Option[Double], spoken_languages: String,
  status: String, tagline: String, title: String, vote_average: Double, vote_count: String,
  cast: String, crew: String, director: String)

// val iterator permite iterar en el csv obteniendo una lista de movies, es decir que el iterator nos permite
// leer el csv movie_dataset
val iterator = readData.readCsv(list, Movie)(rfc.withHeader(Header = true))
// value permite obtener todas las filas que tengan una película
val value = iterator.collect({ case Right(movie) => movie }) // filter, hace una colección de únicamente las filas que tengan películas
// En caso de que un objeto una película en alguna fila no value la va a recibir
val MovieValue = iterator.collect({ case Left(movie) => movie }) // Nota el Right es cuando tiene movies y el Left es cuando no tiene movies
// Como el género de la película que se desea sacar es de terror o horror se declara una variable genero
val genero = "Horror"
// filter sirve para filtrar los datos
// iterator.filter(_.exists(_.genres == genero)) // filter, hace una colección de únicamente cuando existen = Hay movies (...exist)
// == la fecha se llama lambda: el (x => x.genres) permite recorrer todas las filas de la columna(genres) del csv
```





CONSULTA

Consultar las películas que tienen el genero terror

```
----- consultar que películas pertenecen al género de terror
<<<<< Resultado de ejecución: >>>>>
Las películas que tienen el género de Horror son:
Género  Index  Título Original
Horror  4478   Bled
Horror  4102   Witchboard
Horror  2308   Land of the Dead
Horror  1993   Final Destination
Horror  1501   Stigmata
Horror  4585   ...E tu vivrai nel terrore! L'aldilà
Horror  1811   The Tooth Fairy
Horror  3312   Hostel: Part II
Horror  1407   House of Wax
Horror  1168   The Conjuring 2
Horror  1378   A Nightmare on Elm Street
Horror  1396   Poltergeist
Horror  1291   Legion
Horror  1823   Freddy vs. Jason
Horror  2238   Willard
Horror  2337   The Fog
Horror  2379   Restoration
Horror  2438   Evil Dead
Horror  2462   The Texas Chainsaw Massacre: The Beginning
Horror  2526   Mama
Horror  2551   Halloween II
Horror  2576   Urban Legends: Final Cut
Horror  2902   Triangle
Horror  2965   Darkness
Horror  3015   Sinister J
```




Consultar películas que tienen el género acción

[illegible]



Consultar películas que tienen el género acción

```
// New Mergle Code Editor Build Run Tools VCS Window Help - Proyecto1\src\main\scala\Comentarios.scala - Comentario1.scala [Project:Proyecto1]
// src/main/scala/Comentarios.scala
// Preguntalocalia

Preguntalocalia = Preguntalocalia - Preguntalocalia

// x => List(x.index, x.original_title, x.release_date) Permite sacar una lista de los datos que se necesitan en este paso
// se saca el index, el original_title y el release_date
val filtered = iterator.filter(_.exists(s => s.genres==genero)).map(_._map(
  s => List(s.genres, s.index, s.original_title)))
// Nota esta parte es como una condiciona! (_.exists(s => s.genres==genero)) por que en caso de que
// no se cumpla ni se va a filtrar ningun dato por eso filtra solo las peticos que existe una movie y las series que en su
// campo genres tengan solo el genero de Action
// luego se presenta unos consejos de como se va salir en la consola la presentacion
println("==== Resultado en iteracion: =====")
printf("Las peliculas que tienen el genero de %s son:\n", genero)
println("Resumen\tIndex\tTitulo Original")
// El filtro con los datos filtrados obtenidos en la linea 41, es decir los datos que al compararlos con la condicional explicada en la linea 41
// luego se hace un foreach por que del filtered se recupera una lista y en caso de que existan varias peliculas que cumplen la
// condicional de que en el release_date contenga el "2002" se necesitara recorrer la lista que se lo hace con un foreach
// x => el x luego permite recuperar las listas de la linea 44
// x.right.get(0) permite recuperar el dato que se encuentran en la lista en la posicion 0
// Nota como tenemos las listas con los campos de la linea 44 (index,originaltitle,release_date) se cuenta desde 0,1,2
// por eso para recuperamos el index con x.right.get(0), originaltitle con x.right.get(1) y genres con x.right.get(2)
// los (0),(1),(2) son las posiciones de los datos que se encuentran en la lista de la linea 44
Filtered.foreach(s => printf("%s\t%s\t%s\n",s.right.get(0),s.right.get(1),s.right.get(2)))
println("=====")
// =====Codigo para la creacion del csv=====
// En campo se guarda de los datos filtrados que es una lista hacemos un nuevo y guardamos en una lista el index,el original title y el release_date
val campos = filtered.map(s => List(s.right.get(0), s.right.get(1), s.right.get(2))) //campos recupera una lista de lista de String
// Hacemos uso de la libreria de la linea 6 (java.io.File) para el new file
// write csv permite escribir el csv y recibir una lista de lista de String, la cual es la variable campos de la linea 64
// cfc.withHeader(...) es el encabezado de las columnas del csv se dice el csv se va a generar con el encabezado [...]
// y luego se van a insertar los campos, el cual es la variable que tiene una lista de lista de String
new File(campos).write(cfc.withHeader("Index","Titulo Original","Release Date").withFirstRow(campos).withFirstRow("Index"))
```



CONSULTA

Consultar películas que ³ tienen el género acción

```
C:\Users\Usuario\jdk\openjdk-17.0.1\bin\java.exe ...  
-----< Consultar las películas que pertenecen al género de acción >-----  
<<<<< Resultado de ejecución: >>>>>  
Las películas que tienen el género de Action son:  
Genero  Index  Titulo Original  
Action  750      State of Play  
Action  2309     逃出生天  
Action  560      Driven  
Action  162      Stealth  
Action  1701     Once Upon a Time in Mexico  
Action  1398     Max Payne  
Action  44       Furious 7  
Action  708      Maze Runner: The Scorch Trials  
Action  1104     The Bounty Hunter  
Action  1249     Torque  
Action  1528     Criminal  
Action  1848     Agent Cody Banks  
Action  2142     Risen  
Action  2602     The Man with the Iron Fists  
Action  2777     Excessive Force  
Action  3142     American Heist  
Action  3335     Pound of Flesh  
Action  3485     Only the Strong  
Action  3848     D.E.B.S.  
Action  4215     Bloodsport  
Action  4761     13 Hours
```







CONSULTA

Consultar nombres de los 4 directores que dirigieron películas de aventura



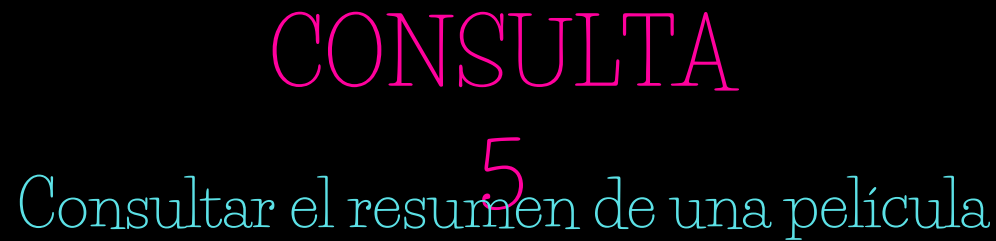
CONSULTA

5

Consultar el resumen de una película

```
preguntascsv: Preguntascsv - Preguntascsv - Preguntascsv - Preguntascsv
package ConsultasCSV;

// Importaciones: Principalmente la libreria kunen esta enfocada a scala para el manejo de csv en programación
import java.io.File; // La libreria java.io.File nos permite generar o crear un archivo desde código, esta libreria es util
object Preguntascsv extends App { // se genera el objeto pregunta 2 y extend App permite que el código escrito en este objeto pueda ser ejecutado dentro de el
    println("----> Consulta del resumen de una película ingresada por consola <----")
    // rawData nos permite obtener la ubicación donde se encuentra el csv con el que vamos a trabajar, en este caso es /movie_dataset.csv
    val rawData: java.net.URL = getClass.getResource("data/movie_dataset.csv")
    case class Movies( // case class Movie nos permite identificar cada una de las columnas que tiene el csv
        index: String, budget: String, genres: String,
        homepage: String, id: String, keywords: String, original_language: String, original_title: String,
        overview: String, popularity: Double, production_companies: String, production_countries: String,
        release_date: String, revenue: String, runtime: Option[Double], spoken_languages: String,
        status: String, tagline: String, title: String, vote_average: Double, vote_count: String,
        cast: String, crew: String, director: String)
    // val iterator permite iterar en el csv obteniendo una lista de movies, es decir que el iterator nos permite
    // leer el csv movie_dataset
    val iterator = rawData.readCsv[List, Movies](rfc.withHeader("yes" = true))
    // value permite obtener todas las filas que tengan una película
    var value = iterator.collect({ case Right(movies) => movies }) // Itera hace una colección de únicamente las filas que tienen películas
    // En caso de que no exista una película en alguna fila nos value la va a recibir
    val Notvalue = iterator.collect({ case Left(movies) => movies }) // Note el Right es cuando tiene movies y el Left es cuando no tiene movies
    // Como el genero de la película que se desea sacar es de terror a horror se declara una variable genero
    val nombrePeli = scala.io.StdIn.readLine()
    // filtered sirve para filtrar los datos
    // iterator.filter(_._exists(_.genres)) filtra los datos únicamente cuando existen y hay movies(_._exists)
    // == la fecha se tiene lambda; el (x==> x.genres) permite recorrer todas las filas de la columna(genres) del csv
    // x.original_title == nombrePeli luego value si los datos dentro de esa columna original_title es igual al nombre de la película ingresada mediante consola
    // map(_._map) luego hace map el primer map permite recuperar los movies de la lista movie y el segundo map permite recuperar los atributos de cada movie
    // x => List(x.index, x.original_title, x.release_date) Permite sacar una lista de los datos que se necesitan, en este caso
    // se saca el index, el original_title y el release_date
```





CONSULTA

5 Consultar el resumen de una película

```
<<<< Resultado de ejecucion: >>>>
```

Index	Titulo	Resumen
-------	--------	---------

CONCLUSIONES



La normalización es un proceso bastante necesario para optimizar los datos, evitar redundancia de datos y errores, combatir problemas de actualización, entre otros ya que gracias a estas reglas se puede crear un modelo relacional concreto, fácil de entender.

Finalmente se logro concluir que el manejo de grandes compendios se base de datos conlleva un trabajo ordenado y analítico ya que esto permite detectar anomalías, errores campos multivaluados, campos compuestos entre otra característica de archivos de este tipo con el fin de dar solución a todo tipo de problemática de una manera profesional.

t h e
e n d