

# UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA



## TITULACIÓN DE INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN

### TEMA:

Ciclo de vida de bases de datos relacionales normalizada

**Tercer semestre.**

**Alumno:** Denis Alexander Cuenca Buele

**Tutor:** Ing. Nelson Piedra

SEGUNDO BIMESTRE  
2021-2022

## 1. Introducción:

En el presente proyecto se aspira explicar el proceso de normalización de una base de datos “Movie dataset” la cual presenta errores de diseño los cuales imposibilitan el uso de la misma, motivo por el cual se aplicaran varias estrategias para la modificación de esta base de datos, que nos permitan volverla un modelo que responda de forma eficiente y estable reglas de normalización y su uso en futuros procesos de implementación. En este documento se encuentra detallado el proceso para construir un modelo estable, detallando los campos existentes en las tablas, su errores, recomendaciones de diseño y aplicación de estos desde un enfoque práctico.

## 2. Desarrollo:

A continuación se nos propone un modelo de base de datos “movie dataset” la misma que contiene información sobre varias películas como su nombre, presupuesto, cast, entre otras que se detallaran más adelante; la base de datos cuenta con una tabla única “Movie” con un total de 24 atributos entre los cuales resaltan: una clave primaria, claves candidatas, atributos multivaluados, atributos con formato JSON; todo esto en un mismo archivo csv.

A fin de crear un modelo sostenible y consistente se pretende la aplicación de reglas de normalización sobre este primer estado de la base de datos, para lo cual se debe comprender a fondo el modelo con el que se trabajará, así pues se comienza por describir la única entidad que contiene el modelo, la tabla “Movie”.

La entidad “Movie” contiene los siguientes atributos:

MOVIE:

- index
- budget
- homepage
- id
- original language
- original title
- overview
- popularity
- realese date
- revenue
- runtime
- status
- tagline
- title
- vote average
- vote count
- genres
- keywords
- production companies
- production countries
- spoken languages
- cast
- crew

## 2.1. Antes de aplicar Normalización:

En la entidad "Movie" existen varios campos que no son compatibles con el concepto de modelo sostenible, según la normalización, por ejemplo existen atributos multivaluados además de una dependencia funcional parcial en toda la tabla, lo cual se entra en conflicto con la segunda y tercera forma normal.

Los campos que presentan estos problemas en cuestión son los siguientes:

## 2.2. Campos Multivaluados (con sus respectivos atributos):

- **genres:** id\_movie, genre
- production companies:** id\_movie, id, company
- production countries:** id\_movie, iso\_3166\_1, country
- spoken languages:** Iso\_639\_1, id\_movie, name
- cast:** name, id, id\_movie
- **crew:** id\_movie, id\_crew, job, name, gender, credit\_id, department
- Director:** name, id, id\_movie

Los campos aquí expuestos representan un problema de primera forma normal, al ser campos que almacenan valores multivaluados, por lo cual se los separa en tablas diferentes con esperando así generar un mejor diseño, el resultado final se presenta a continuación donde se describe cada tabla con sus atributos:

## 2.3. Dependencias funcionales de las nuevas tablas:

A continuación se describe las dependencias funcionales de cada una de las nuevas tablas:

- **genres:**  
id\_movie → genre
- keywords:**  
id\_movie, id → keyword
- movie\_production companies:**  
id\_movie, id → id\_movie, id
- production companies:**  
id → company name
- movie\_production countries:**  
id\_movie, iso\_3166\_1 → id\_movie, id

**-production countries :**

iso\_3166\_1 → country

**-movie\_spoken languages :**

Iso\_639\_1, id\_movie → Iso\_639\_1, id\_movie

**-spoken languages :**

Iso\_639\_1 → name

**- crew:**

id\_movie, credit\_id --> job, department, id\_crew

**- person:**

credit\_id → name, gender,

**-movie\_director:**

id\_movie, credit\_id --> id\_movie, credit\_id

Siendo todas las tablas, a excepción de crew dependencias funcionales completas las cuales responden al modelo que nos propone la segunda forma normal.

**ENTIDAD MOVIE:**

Columna	Dominio	Opcional	Multivaluado	Proposito	Tipo	comentario
id	int(10)	FALSE	FALSE	Identificar las pelicula de forma única	Clave primaria	
Vote Average	double	FALSE	FALSE	Promedio de los votos recolectados		
Vote Count	int(8)	FALSE	FALSE	Total de votos recolectados		
home Page	varchar(20)	FALSE	FALSE	Página oficial de la película		El valor introducido es una url
revenue	double	TRUE	FALSE	Ganancia recolectada por la película		
budget	double	TRUE	FALSE	Presupuesto destinado para la producción de la película		
status	varchar(10)	FALSE	FALSE	Estatus de producción de la película		
tagline	varchar(20)	TRUE	FALSE	Lema de la película		
original language	varchar(20)	FALSE	FALSE	Lenguaje original en el que la película fue grabada		
original title	varchar(20)	FALSE	FALSE	Titulo seleccionado para la película		

runtime	double	TRUE	FALSE	Duración de la proyección de la película		
release date	date	FALSE	FALSE	Fecha de salida del juego		Valor introducido en el formato: D/M/A
overview	Varchar(500)	TRUE	FALSE	Sinopsis de la trama principal de la película		
popuarity	Int(15)	TRUE	FALSE	Popularidad en puntos que obtuvo la película		
title	Varchar(30)	TRUE	FALSE	Título de la película		
Caste	Varchar(300)	TRUE	FALSE	Nombre de la trabajadores		

**Entidad Production\_companies:**

Columna	tipo de dato	Opcional	Multivaluado	Propósito	tipo	comentario
id	Int(10)	FALSE	FALSE	Identificar de forma única la columna	Clave primaria	
name	Varchar(20)	FALSE	FALSE	Nombre de la compañía productora del película		

**Entidad movie\_production\_companies:**

Columna	tipo de dato	Opcional	Multivaluado	Propósito	tipo	comentario
id	Int(10)	FALSE	FALSE	Identificar de forma única la columna	Clave primaria	
Id(Movie)	int(10)	FALSE	FALSE	Identificar las pelicula de forma única	Clave primaria y foránea	

**Entidad Production\_countries:**

Columna	tipo de dato	Opcional	Multivaluado	Propósito	tipo	comentario
name	Varchar(15)	FALSE	FALSE	Nombre del país donde se produjo la película.		
iso_3166_1	Varchar(2)	FALSE	FALSE	Código de abreviación de un país	Clave primaria	

**Entidad Production\_countries:**

Columna	tipo de dato	Opcional	Multivaluado	Propósito	tipo	comentario
iso_3166_1	Varchar(2)	FALSE	FALSE	Código de abreviación de un país	Clave primaria	
Id(Movie)	int(10)	FALSE	FALSE	Identificar las película de forma única	Clave primaria y foranea	

**Entidad Spoken\_languages:**

Columna	tipo de dato	Opcional	Multivaluado	Propósito	tipo	comentario
name	Varchar(20)	FALSE	FALSE	Nombre del lenguaje usado en la película.		
Iso_639_1	Varchar(15)	FALSE	FALSE	Código de abreviación de un lenguaje	Clave primaria	

**Entidad movie\_spoken\_languages:**

Columna	tipo de dato	Opcional	Multivaluado	Propósito	tipo	comentario
Iso_639_1	Varchar(15)	FALSE	FALSE	Código de abreviación de un lenguaje	Clave primaria	
Id(Movie)	int(10)	FALSE	FALSE	Identificar las película de forma única	Clave primaria y foranea	

**Entidad crew:**

Columna	tipo de dato	Opcional	Multivaluado	Propósito	tipo	comentario
name	Varchar(40)	FALSE	FALSE	Nombre del trabajador		
Gender	Int(1)	FALSE	FALSE	Genero del trabajador		Atributo identificado con 2 para hombre y 1 para mujer
Credit_id	Varchar(20)	FALSE	FALSE	Identificador del crédito del trabajador		
Id	Int(5)	FALSE	FALSE	Identifica de forma única a cada trabajador	Clave primaria	
Id(Movie)	int(10)	FALSE	FALSE	Identificar las película de forma única	Clave primaria	

**movie\_director:**

Columna	tipo de dato	Opcional	Multivaluado	Propósito	tipo	comentario
Credit_id	Varchar(20)	FALSE	FALSE	Identificador del crédito del trabajador		
Id(Movie)	int(10)	FALSE	FALSE	Identificar las película de forma única	Clave primaria	

**Genres:**

Columna	tipo de dato	Opcional	Multivaluado	Propósito	tipo	comentario
Genre	Varchar(20)	FALSE	FALSE	Nombre del género		
Id(Movie)	int(10)	FALSE	FALSE	Identificar las película de forma única	Clave primaria	

**person:**

Columna	tipo de dato	Opcional	Multivaluado	Propósito	tipo	comentario
name	Varchar(20)	FALSE	FALSE	Nombre del trabajador		
Credit_id	Varchar(20)	FALSE	FALSE	Identificador del crédito del trabajador	Clave primaria	
gender	int(10)	FALSE	FALSE	Genero del trabajador		

**2.4 Diagrama Entidad Relación:**

A continuación se presenta el resultado de separar cada una de las tablas y se presenta la relación existente entre ellas, nótese que por el estilo de flecha la carnalidad es de muchos a muchos (m:n) entre la mayoría de las entidades



En el diagrama, todas las entidades, a excepción de department y job, tienen relación con la entidad Movie, pues es esa la entidad de donde surgen, la relación de movie con keywords es de muchos a



muchos, con ello se podrá guardar por una película varias palabras clave, y una palabra clave será capaz de describir a varias películas; similar es el caso para las entidades genres, director, cast y spoken languages. Production companies y movie comparten la misma cardinalidad muchos a muchos en donde una compañía será capaz de producir varias películas y una película producida por muchas compañías; lo mismo sucede con la entidad production countries, La columna crew conserva un formato JSON en donde se detallan a forma de clave-valor los campos que ha de contenerse en una tabla de forma individual separada, en esta tabla las persona pertenecientes al crew son capaces de pertenecer a uno o más departamentos, y estas mismas personas podrán ejercer una o mas funciones de trabajo en la película, por lo que estas son entidades que comparten una cardinalidad de muchos a muchos con la entidad crew.

## 2.5. Tercera forma normal:

La tercera forma normal nos indica que no debe existir dependencias funcionales transitivas, además se añade que sin una tabla se encuentra en primera y segunda forma normal, entonces la tercera norma ya está aplicada por lo que se puede decir que nuestro esquema se encuentra normalizado.

## 2.6. Implementación en sql:

### Estrategia de creación tablas y carga de datos:

Para empezar se realiza la carga directa de los datos del archivo CSV en nuestra base de datos, previo a realizar la carga en tablas temporales se realiza la limpieza de los atributos con formato JSON:

```
15 -- Se borra si existe una version previa del procedimiento:
16 DROP TABLE IF EXISTS movie_dataset_cleaned;
17 -- Creacion de la tabla temporal:
18
19 CREATE TABLE movie_dataset_cleaned AS
20 SELECT
21
22 -- Se almacena los valores a implementar en la tabla.
23 'index', budget, genres, homepage, id, keywords, original_language, original_title, overview, popularity,
24 production_countries, production_countries, release_date, revenue, runtime, spoken_languages, 'status',
25 tagline, title, vote_average, vote_count, CONVERT(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
26 (REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
27 REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
28 REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
29 REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
30 REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
31 REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
32 REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
33 REPLACE(REPLACE(cast, '\u00e1', 'á'), '\u00e5', 'å'), '\u00e9', 'é'),
34 '\u00eb', 'ë'), '\u00ed', 'í'), '\u00e0', 'à'), '\u00f1', 'ñ'), '\u00f8', 'ø'), '\u0042', 'B'),
35 '\u00438', 'N'), '\u0044f', 'я'), '\u00421', 'C'), '\u0043d', 'H'), '\u00440', 'p'), '\u00433', 'r'),
36 '\u0044c', 'b'), '\u0067e', 'ج'), '\u006cc', 'ج'), '\u00645', 'م'), '\u00627', 'l'), '\u00646', 'ج'),
37 '\u00646', 'ع'), '\u0062f', 'ج'), '\u000e8', 'è'), '\u00f3', 'ó'), '\u0160', 'Š'), '\u0107', 'ć'),
38 '\u0107', 'ć'), '\u00f6', 'ö'), '\u00e4', 'ä'), '\u00e4', 'ä'), '\u00e4', 'ä'), '\u0144', 'ñ'),
39 '\u02019', 'í'), '\u00fc', 'ü'), '\u00c1', 'Á'), '\u00ea', 'é'), '\u00ea', 'é'), '\u00e7', 'ç'),
40 '\u00dc', 'Ü'), '\u0159', 'ř'), '\u00d8', 'Ø'), '\u00fa', 'ú'), '\u010d', 'č'), '\u010d', 'č'),
41 '\u00f0', 'ð'), '\u0219', 'š'), '\u00d3', 'ó'), '\u0110', 'Đ'), '\u0161', 'š'), '\u0101', 'ā'),
42 '\u00c5', 'Å'), '\u0043b', 'n'), '\u014c', 'ō'), '\u016b', 'ū'), '\u014d', 'ö'), '\u015b', 'ś'),
43 '\u00ef', 'ï'), '\u021b', 't'), '\u00c9b', 'g'), '\u00f4', 'ô'), '\u0301', 'í'), '\u00fb', 'ü'),
44 '\u00fb', 'ü'), '\u01ed7', 'ð'), '\u01ecb', 'i'), '\u01ea3', 'ä'), '\u01ebf', 'é'), '\u015f', 'ş'),
45 '\u015ea', 'D'), '\u017e', 'ž'), '\u00c0', 'À'), '\u0131', 'ı'), '\u011f', 'ğ'), '\u01ec1', 'é'),
46 '\u00639', 'ع'), '\u00ee', 'iع'), '\u00e6', 'æ'), '\u00c9', 'É'), '\u00df', 'ß'), '\u015ea', 'D') using utf8mb4) AS cast,
```

```

0    (REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE
1      REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE
2        (REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
3          REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(
4            REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(crew,'\\\'', '\\'),
5              uuuu, uu)),
6                u',   uu,   uu,   uu),
7                  u':   uu,   uu:: uu),
8                    u': u,   uu:: u),
9                      u',   u,   uu),
10                     u{'u, u{uuu}, "\\t", uu),
11                   '\\u00e9','é'),'\\u00e1','á'),'\\u00f1','ñ'),'\\u00c9','É'),
12                   '\\u0159','ř'),'\\u00f4','ô'),'\\u00f3','ó'),'\\u00ed','í'),
13                   '\\u00d8','ø'),'\\u00f8','ø'),'\\u00c5','Å'),'\\u00f6','ö'),
14                   '\\u0142','ł'),'\\u017e','ž'),'\\u0161','š'),'\\u00e8','è'),
15                   '\\u0144','ň'),'\\u00e7','ç'),'\\u00ef','ï'),'\\u0160','Š'),
16                   '\\u00fc','ü'),'\\u00d3','ð'),'\\u00fd','ý'),'\\u00cf','İ'),
17                   '\\u00e3','ä'),'\\u00ee','î'),'\\u00e2','â'),'\\u00e4','ä'),
18                   '\\u00e5','å'),'\\u00eb','ë'),'\\u00eb','ë'),'\\u00fa','ú'),
19                   '\\u00df','ß'),'\\u00f0','ð'),'\\u00c1','Á'),'\\u0107','č'),
20                   '\\u015','

|   |
|---|
| P |
| A |
| S |
| E |

'),'\\u0165','t'),'\\u00ea','ê'),'\\u014c','ō'),
21                   '\\u00c0','À'),'\\u2019','''),'\\u00fb','û'),'\\u00e6','æ'),
22                   '\\u00fe','þ'),'\\u0141','Ł'),'\\u0411','Б'),'\\u043e','о'),
23                   '\\u0440','р'),'\\u0438','и'),'\\u0441','с'),'\\u0421','С'),
24                   '\\u0443','у'),'\\u0442','Т'),'\\u0430','а'),'\\u0446','ц'),
25                   '\\u043a','к'),'\\u0439','й'),'\\u010d','Ď'),'\\u5f20','张'),
26                   '\\u07ac','立'),'\\u00d6','ö'),'\\u0441','с'),'\\u010c','Č'),
27                   '\\u00cd','Í'),'\\u00f5','õ'),'\\u00e0','à'),'\\u00f2','ò'),
28                   '\\u00d4','ô'),'\\u011b','ě'),'\\u00de','P'),'\\u00ec','ì'),
29                   '\\u00da','Ú'),'\\u010e','ď'),'\\u0433','r'),
30                   uuuu,   uu),   uuuu,   uuuu)using utf8mb4) AS Crew,
```

Luego se da inicio a la creación de procedimientos para extraer los datos cargados en la tabla `movie_dataset`:

```

96      -- Se eliminan las tablas temporales de la tabla
97      DROP PROCEDURE IF EXISTS Jsons ;
98      DELIMITER //
99      CREATE PROCEDURE Jsons()
100      BEGIN
101          -- Se repite para ir cargando datos desde el array JSON hacia la tabla temporal
102          DECLARE a INT Default 0 ;
103          -- Se eliminan y crean tablas temporales para almacenar los datos:
104          DROP TABLE IF EXISTS tmp_production_companies ;
105          CREATE TABLE tmp_production_companies (id_movie INT, id_production_company INT, name_production_company VARCHAR (100) );
106          DROP TABLE IF EXISTS tmp_production_countries ;
107          CREATE TABLE tmp_production_countries (id_movie INT, iso_3166_1 VARCHAR (7), country VARCHAR (100) );
108          DROP TABLE IF EXISTS tmp_spoken_languages;
109          CREATE TABLE tmp_spoken_languages (id_movie INT, iso_639_1 VARCHAR (5), `language` VARCHAR (100) );
110          DROP TABLE IF EXISTS tmp_crew ;
111          CREATE TABLE tmp_crew(id_movie INT, id_crew INT, job VARCHAR (200), name VARCHAR (400), gender INT, credit_id VARCHAR (50), department VARCHAR (50) );
112          DROP TABLE IF EXISTS tmp_genres;
113          CREATE TABLE tmp_genres (id_movie INT,genre VARCHAR(100));
114          DROP TABLE IF EXISTS tmp_cast;
115          CREATE TABLE tmp_cast(id_movie INT, cast VARCHAR(100));
116          DROP TABLE IF EXISTS tmp_movie_keyword;
117          CREATE TABLE tmp_movie_keyword(id_movie INT, keyword VARCHAR(100));
118

```

```

119 -- inicio del loop
120 simple_loop: LOOP
121     -- Se insertan los datos en formato json en las tablas temporales:
122     INSERT INTO tmp_production_companies (id_movie, id_production_company, name_production_company)
123     SELECT id as id_Movie,
124
125             JSON_EXTRACT(CONVERT(production_companies using utf8mb4), CONCAT("$[,a,].id")) AS id_production_company,
126             JSON_EXTRACT(CONVERT(production_companies using utf8mb4), CONCAT("$[,a,].name")) AS id_production_company
127 FROM movie_dataset m ;
128 INSERT INTO tmp_production_countries (id_movie, iso_3166_1, country)
129 SELECT id,
130
131         JSON_EXTRACT(production_countries, CONCAT('$[,a,].iso_3166_1')) AS iso_3166_1,
132         JSON_EXTRACT(production_countries , CONCAT('$[,a,].name')) AS country
133 FROM movie_dataset m ;
134 INSERT INTO tmp_spoken_languages (id_movie, iso_639_1, `language`)
135 SELECT id,
136
137         JSON_EXTRACT(spoken_languages , CONCAT('$[,a,].iso_639_1')) AS iso_639_1,
138         JSON_EXTRACT(spoken_languages , CONCAT('$[,a,].name')) AS language
139 FROM movie_dataset m ;
140 INSERT INTO tmp_crew (id_movie, id_crew, job, name, gender, credit_id, department)
141 SELECT id as id_Movie,
142
143         JSON_EXTRACT(CONVERT(crew using utf8mb4), CONCAT("$[,a,].id")) AS id_crew,
144         JSON_EXTRACT(CONVERT(crew using utf8mb4), CONCAT("$[,a,].job")) AS job,
145         JSON_EXTRACT(CONVERT(crew using utf8mb4), CONCAT("$[,a,].name")) AS name,
146         JSON_EXTRACT(CONVERT(crew using utf8mb4), CONCAT("$[,a,].gender")) AS gender,
147         JSON_EXTRACT(CONVERT(crew using utf8mb4), CONCAT("$[,a,].credit_id")) AS credit_id,
148         JSON_EXTRACT(CONVERT(crew using utf8mb4), CONCAT("$[,a,].department")) AS department
149 FROM movie_dataset cleaned_m ;

```

```

INSERT INTO tmp_cast(id_movie, cast)
SELECT id,
-- Se implementa una estructura condicional que nos permita conocer la longitud y espacios en blanco existentes en cada tabla.
REPLACE(JSON_EXTRACT(
  CONCAT('["',
    IF(SpacesNumber >= 13, CONCAT(SUBSTRING_INDEX(SUBSTRING_INDEX(cast, ' ', -14), ' ', 2), '"', '"'), '') ,
    IF(SpacesNumber >= 11, CONCAT(SUBSTRING_INDEX(SUBSTRING_INDEX(cast, ' ', -12), ' ', 2), '"', '"'), '') ,
    IF(SpacesNumber >= 9, CONCAT(SUBSTRING_INDEX(SUBSTRING_INDEX(cast, ' ', -10), ' ', 2), '"', '"'), '') ,
    IF(SpacesNumber >= 7, CONCAT(SUBSTRING_INDEX(SUBSTRING_INDEX(cast, ' ', -8), ' ', 2), '"', '"'), '') ,
    IF(SpacesNumber >= 5, CONCAT(SUBSTRING_INDEX(SUBSTRING_INDEX(cast, ' ', -6), ' ', 2), '"', '"'), '') ,
    IF(SpacesNumber >= 3, CONCAT(SUBSTRING_INDEX(SUBSTRING_INDEX(cast, ' ', -4), ' ', 2), '"', '"'), '') ,
    IF(SpacesNumber >= 1, CONCAT(SUBSTRING_INDEX(cast, ' ', -2), '"', '"'), '') ,
  '"]'), CONCAT("$[",a,""])),
"=", "") AS CastJson
FROM (
  SELECT id, cast, LENGTH(cast) - LENGTH(REPLACE(cast, ' ', '')) AS SpacesNumber FROM (
  SELECT id,
  REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(cast, '"', ''), ' Jr.', '=Jr.'), ' Jr ', '=Jr '), ' ', ' '), '='),
  ' The ', ' The='), ' the ', ' the='), 'the=Cable Guy', 'the=Cable=Guy'), 'Tupac Amaru Shakur', 'Tupac Amaru=Shakur') AS cast
FROM movie_dataset WHERE Cast <> '' t1) t2;

```

```

175 INSERT INTO tmp_movie_keyword
176 SELECT id,
177 -- Se implementa una estructura condicional que nos permita conocer la longitud y espacios en blanco existentes en cada tabla.
178 REPLACE(JSON_EXTRACT(REPLACE(
179   CONCAT('["',
180     IF(SpacesNumber >= 13, CONCAT(SUBSTRING_INDEX(SUBSTRING_INDEX(keywords, ' ', -14), ' ', 2), '"', '"'), '') ,
181     IF(SpacesNumber >= 11, CONCAT(SUBSTRING_INDEX(SUBSTRING_INDEX(keywords, ' ', -12), ' ', 2), '"', '"'), '') ,
182     IF(SpacesNumber >= 9, CONCAT(SUBSTRING_INDEX(SUBSTRING_INDEX(keywords, ' ', -10), ' ', 2), '"', '"'), '') ,
183     IF(SpacesNumber >= 7, CONCAT(SUBSTRING_INDEX(SUBSTRING_INDEX(keywords, ' ', -8), ' ', 2), '"', '"'), '') ,
184     IF(SpacesNumber >= 5, CONCAT(SUBSTRING_INDEX(SUBSTRING_INDEX(keywords, ' ', -6), ' ', 2), '"', '"'), '') ,
185     IF(SpacesNumber >= 3, CONCAT(SUBSTRING_INDEX(SUBSTRING_INDEX(keywords, ' ', -4), ' ', 2), '"', '"'), '') ,
186     IF(SpacesNumber >= 1, CONCAT(SUBSTRING_INDEX(keywords, ' ', -2), '"', '"'), '') ,
187     '"]', '['', '[]')', CONCAT("$[",a,""])),
188   "=", "") AS keywordJson
189 FROM(
190   select id, keywords, LENGTH(keywords) - LENGTH(REPLACE(keywords, ' ', '')) AS SpacesNumber FROM (
191   SELECT id,
192   REPLACE(REPLACE(keywords, '"', ''), 'marvel comic', 'marvel=comic') AS keywords
193 FROM movie_dataset WHERE keywords <> '' t1) t2;
194 SET a=a+1;
195 IF a=10 THEN
196   LEAVE simple_loop;
197 END IF;
198 END LOOP simple_loop;

```

Una vez se crean y cargan los datos dentro de las tablas temporales se da inicio a la creación de las tablas finales, donde se almacenaran los datos de nuestro modelo ya normalizado.

Primero se borran las tablas implementadas, en caso de existir

```

16 DROP TABLE IF EXISTS spoken_languages;
17 DROP TABLE IF EXISTS production_countries;
18 DROP TABLE IF EXISTS production_companies;
19 DROP TABLE IF EXISTS worker;
20 DROP TABLE IF EXISTS credit;
21 DROP TABLE IF EXISTS movies_crew;
22 DROP TABLE IF EXISTS movies_production_companies;
23 DROP TABLE IF EXISTS movies_production_countries;
24 DROP TABLE IF EXISTS movies_spoken_languages;
25 DROP TABLE IF EXISTS movie;
26

```

```

CREATE TABLE movie AS
SELECT DISTINCT `index`, budget, genres, homepage, id, keywords, original_language, original_title, overview, popularity, release_date, revenue, runtime, `status`,
tagline, title, vote_average, vote_count, cast, director
FROM movie_dataset_cleaned;
ALTER TABLE movie ADD PRIMARY KEY (id);

```

```

CREATE TABLE spoken_languages(
  iso_639_1 VARCHAR(5) NOT NULL,
  `language` VARCHAR(100),
  PRIMARY KEY(iso_639_1)
);

```

```

CREATE TABLE movie_genre(
  id_movie INT not null,
  genre VARCHAR(100),
  PRIMARY KEY (id_movie, genre),
  CONSTRAINT FK_id_movie5 FOREIGN KEY (id_movie) REFERENCES movie(id)
);

```

```
CREATE TABLE production_countries(
    iso_3166_1 VARCHAR(5) NOT NULL,
    country VARCHAR (100),
    PRIMARY KEY(iso_3166_1)
);
```

```
CREATE TABLE production_companies(
    id_production_company INT NOT NULL,
    name_production_company VARCHAR (100),
    PRIMARY KEY(id_production_company)
);
```

```
CREATE TABLE person(
    credit_id CHAR(100) NOT NULL,
    `name` VARCHAR(500),
    gender INT,
    PRIMARY KEY (credit_id)
);
```

```
CREATE TABLE movie_production_companies (
    id_movie INT NOT NULL,
    id_production_company INT NOT NULL,
    PRIMARY KEY(id_movie, id_production_company),
    CONSTRAINT FK_id_movie FOREIGN KEY(id_movie) REFERENCES movie(id),
    CONSTRAINT FK_id_production_company FOREIGN KEY(id_production_company)
    | REFERENCES production_companies(id_production_company)
);
```

```
CREATE TABLE movie_production_countries (
    id_movie INT NOT NULL,
    iso_3166_1 VARCHAR (5) NOT NULL,
    PRIMARY KEY(id_movie, iso_3166_1),
    CONSTRAINT FK_id_movie2 FOREIGN KEY(id_movie)
    REFERENCES movie(id),
    CONSTRAINT FK_iso_3166_1 FOREIGN KEY(iso_3166_1)
    REFERENCES production_countries(iso_3166_1)
);
```

```
CREATE TABLE movie_spoken_languages (
    id_movie INT NOT NULL,
    iso_639_1 VARCHAR (5) NOT NULL,
    PRIMARY KEY(id_movie, iso_639_1),
    CONSTRAINT FK_id_movie3 FOREIGN KEY(id_movie)
    REFERENCES movie(id),
    CONSTRAINT FK_iso_639_1 FOREIGN KEY(iso_639_1)
    REFERENCES spoken_languages(iso_639_1)
);
```

```
CREATE TABLE movie_crew (
    id_movie INT NOT NULL,
    credit_id CHAR(100) NOT NULL,
    id_crew INT,
    job VARCHAR(100),
    departament VARCHAR(100),
    PRIMARY KEY(id_movie, credit_id),
    CONSTRAINT FK_id_movie4 FOREIGN KEY (id_movie)
    REFERENCES movie(id),
    CONSTRAINT FK_credit_id FOREIGN KEY (credit_id)
    REFERENCES person(credit_id)
);
```

```
CREATE TABLE movie_keyword(
    id_movie INT not null,
    keyword VARCHAR(100) NOT NULL ,
    PRIMARY KEY (id_movie, keyword),
    CONSTRAINT FK_id_movie6 FOREIGN KEY (id_movie)
    REFERENCES movie(id)
);
```

```
CREATE TABLE movie_cast(
  id_movie INT NOT NULL,
  cast VARCHAR(100),
  PRIMARY KEY (id_movie, cast),
  CONSTRAINT FK_id_movie7 FOREIGN KEY (id_movie)
    REFERENCES movie(id)
);
```

```
CREATE TABLE movie_director(
  id_movie INTEGER NOT NULL,
  credit_id CHAR(100),
  PRIMARY KEY (id_movie, credit_id),
  CONSTRAINT FK_id_movie8 FOREIGN KEY (id_movie)
    REFERENCES movie(id),
  CONSTRAINT FK_credit_id2 FOREIGN KEY (credit_id)
    REFERENCES person(credit_id)
);
```

Con las tablas finales creadas se procede a cargar los datos dentro de las mismas.

```
-- Se insertan los datos de la tabla production_countries:
INSERT INTO production_countries(iso_3166_1, country)
SELECT DISTINCT iso_3166_1, country
FROM tmp_production_countries;

-- Se insertan los datos de la tabla movie_production_countries:
INSERT INTO movie_production_countries(id_movie, iso_3166_1)
SELECT DISTINCT id_movie, iso_3166_1
FROM tmp_production_countries;

-- Se insertan los datos de la tabla production_companies:
INSERT INTO production_companies(id_production_company, name_production_company)
SELECT DISTINCT id_production_company, name_production_company
FROM tmp_production_companies;
```

```
-- Se insertan los datos de la tabla movie_production_companies:
INSERT INTO movie_production_companies(id_movie, id_production_company)
SELECT DISTINCT id_movie, id_production_company
FROM tmp_production_companies;

-- Se insertan los datos de la tabla spoken_languages:
INSERT INTO spoken_languages(iso_639_1, `language`)
SELECT DISTINCT iso_639_1, `language`
FROM tmp_spoken_languages;

-- Se insertan los datos de la tabla movie_spoken_languages:
INSERT INTO movie_spoken_languages(id_movie, iso_639_1)
SELECT DISTINCT id_movie, iso_639_1
FROM tmp_spoken_languages;

-- Se insertan los datos de la tabla person:
INSERT INTO person(credit_id, name, gender)
SELECT DISTINCT credit_id, name, gender
FROM tmp_crew;
```

```
-- Se insertan los datos de la tabla movie_crew:
INSERT INTO movie_crew(id_movie, credit_id, id_crew, job, department)
SELECT DISTINCT id_movie, credit_id, id_crew, job, department
FROM tmp_crew;

-- Se insertan los datos de la tabla movie_cast:
INSERT INTO movie_cast(id_movie, cast)
SELECT DISTINCT id_movie, cast
FROM tmp_cast;

-- Se insertan los datos de la tabla movie_genre:
INSERT INTO movie_genre(id_movie, genre)
SELECT DISTINCT id_movie, genre
FROM tmp_genres;

-- Se insertan los datos de la tabla movie_keyword:
INSERT INTO movie_keyword(id_movie, keyword)
SELECT DISTINCT id_movie, keyword
FROM tmp_movie_keyword;
```

```
-- Se insertan los datos de la tabla movie_director:
INSERT INTO movie_director(id_movie, credit_id)
SELECT DISTINCT m.id, c.credit_id
FROM movie_dataset m, person c
WHERE m.director = REPLACE(c.name, ' ', '');
```

De esta manera se obtiene nuestro modelo normalizado y con todos sus datos cargados, podemos entonces proceder a eliminar las tablas temporales.

```
-- Se eliminan las tablas temporales, una vez creadas
DROP TABLE IF EXISTS tmp_cast;
DROP TABLE IF EXISTS tmp_genres;
DROP TABLE IF EXISTS tmp_production_countries;
DROP TABLE IF EXISTS tmp_production_companies;
DROP TABLE IF EXISTS tmp_spoken_languages;
DROP TABLE IF EXISTS tmp_crew;
DROP TABLE IF EXISTS movie_dataset_cleaned;
DROP TABLE IF EXISTS tmp_spoken_languages;
DROP TABLE IF EXISTS tmp_movie_keyword;
```

### **3. Conclusiones:**

- El uso de las formas normales es fundamental en la creación de una base de datos, donde la información está siempre expuesta a errores de almacenamiento, y se corre el riesgo de perderla o hacer un mal uso de esta por malas prácticas.
- La base de datos movie dataset tenía errores de diseño los cuales imposibilitan el uso correcto de los datos almacenados, volviéndola para algunos fines una base de datos inútil, al menos como se presentó en una primera etapa, sin embargo tras la aplicación de formas normales y otras modificaciones que nos permitan su mejor uso movie dataset se vuelve funcional y estable.

## Universidad Técnica Particular de Loja

- Ingeniería en Ciencias de la Computación
- Materia: Fundamentos de Base de Datos - Octubre 2021 - Febrero 2022
- Proyecto Final - Ciclo de vida de bases de datos relacionales normalizada
- Estudiante: Denis Alexander Cuenca Buele | [a]utpl.edu.ec
- Link del proyecto en Github: [https://github.com/DenisCuenca/Proyecto\\_integrador.git](https://github.com/DenisCuenca/Proyecto_integrador.git)
- Profesor: Nelson Piedra | <http://investigacion.utpl.edu.ec/nopiedraE>
- Fecha: Loja, 8 de febrero del 2022

n  
l  
a  
c  
e  
s  
  
a  
  
u  
n  
  
s  
i  
t  
i  
o  
  
e  
x  
t  
e  
r  
n  
o  
.