UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA

La Universidad Católica de Loja

Entrega Preliminar del Proyecto Final

Fundamentos de Base de Datos

Autor:

- Edgar Santiago Espinoza Velásquez

Octubre 2022 – Febrero 2023

Contenido

1 Introducción

En el presente proyecto de la materia Fundamentos de Base de datos, se pretende aplicar los conocimientos obtenidos durante todo el ciclo para de esa forma trabajar con el archivo CSV llamado “movie_dataset”, el cual fue obtenido de un repositorio de GitHub, el mismo que tendrá que ser, leído, modelado, limpiado y por último explotado utilizando el lenguaje de consulta SQL específicamente en el sistema de gestión de bases de datos relacional MySQL.

2 Desarrollo del Componente

2.1 Diseño y Modelado de la Base de datos

2.1.1 Diseño Conceptual

Al tener la información definida, con la tabla universal, normalizamos los datos usando Entidad – Relación e identificando los atributos que pertenecen a cada una de las relaciones, para eso, hicimos uso de la herramienta draw.io, la cual nos permitió realizar el modelo conceptual.

2.1.2 Diseño Lógico

A partir del modelo conceptual, se toman los atributos correspondientes a cada tabla y se realiza un prototipo de las tablas en base a las entidades y relaciones identificadas, donde en cada tabla, declaramos una PRIMARY KEY y en caso de que corresponda la FOREIGN KEY, con el fin de que sea una base para desarrollar el esquema de nuestra base de datos.

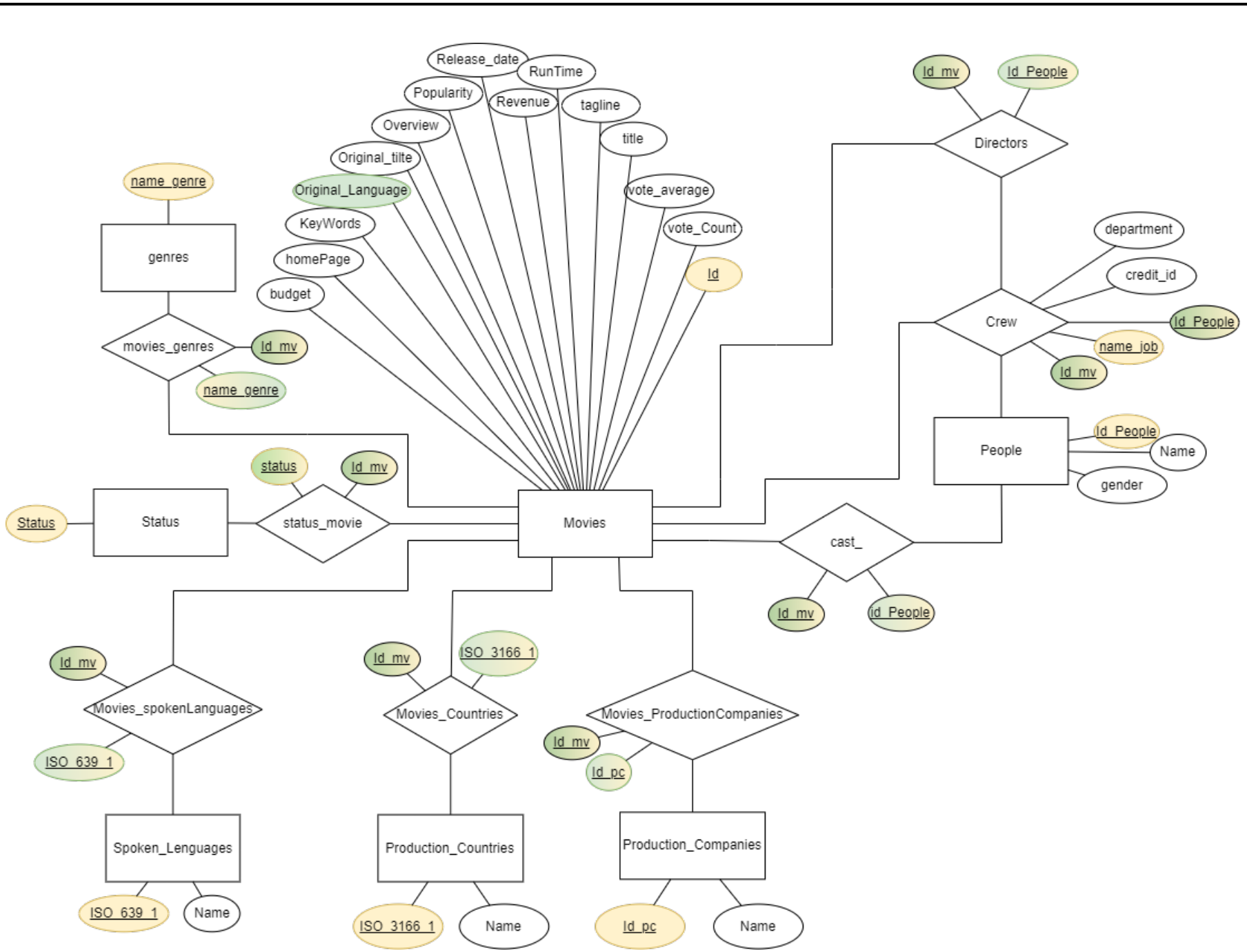


Figure 1.- Modelo Conceptual

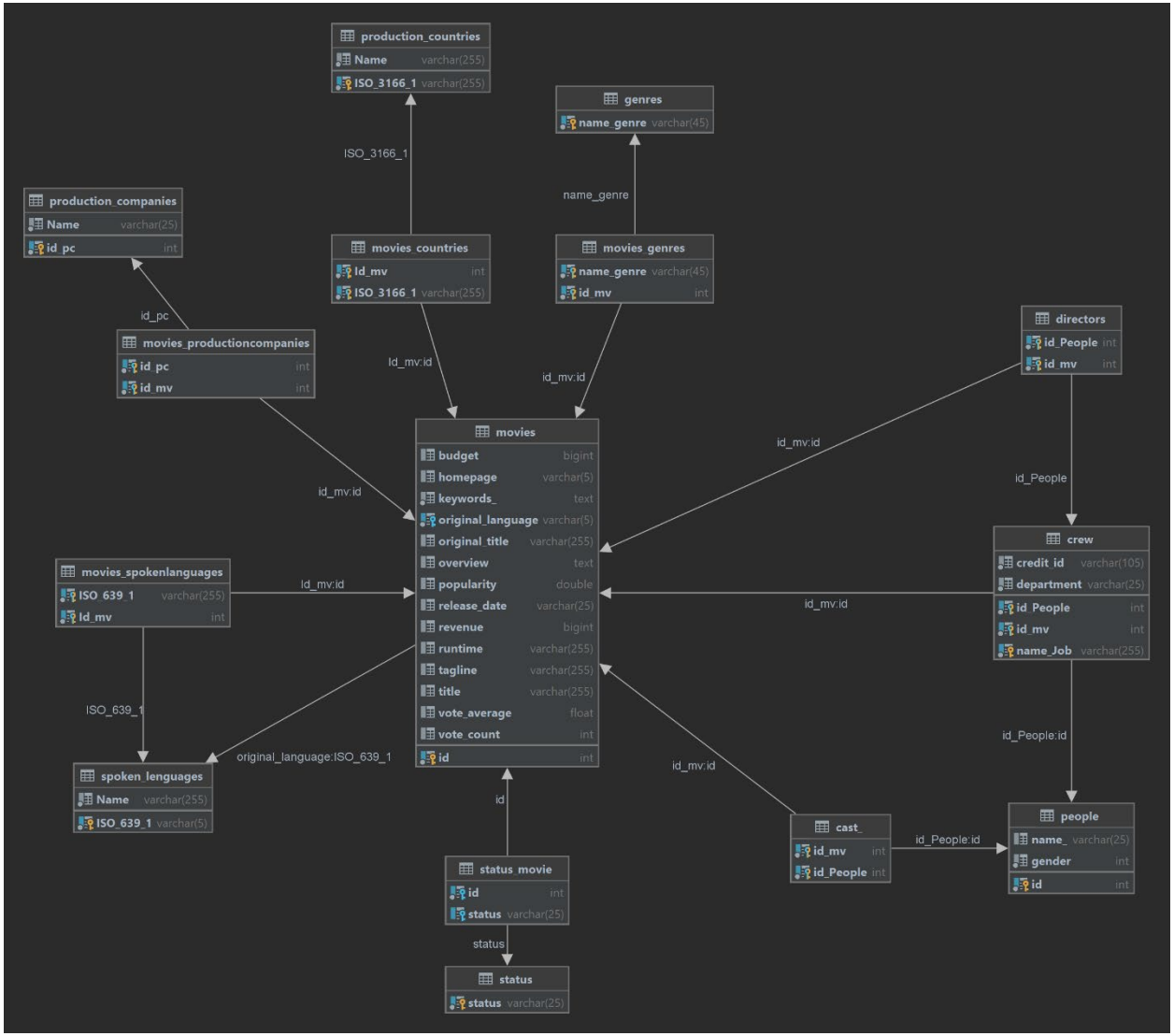


Figure 2.- Modelo Lógico

2.2 Creación del “schema” de la Base de Datos

Definimos la estructura de la base de datos, incluyendo las tablas, relaciones entre ellas, y los atributos de cada tabla. El "schema" actúa como un plan o esqueleto para la organización y almacenamiento de datos.

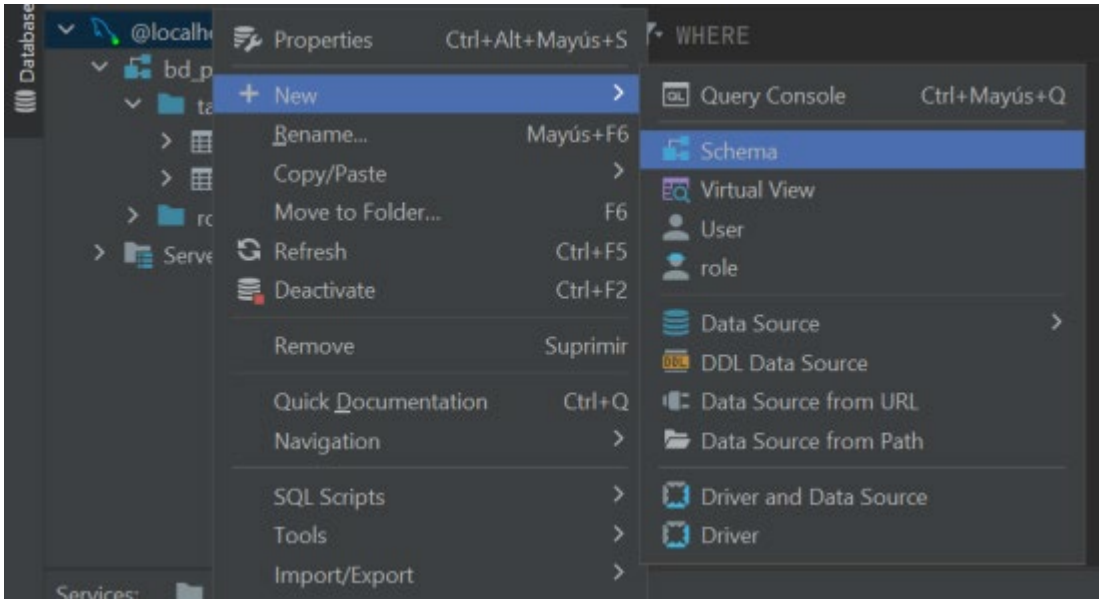


Figure 3.- Creación del "schema"

2.2.1 Conexión a la base de datos

Establecemos una comunicación a la base de datos utilizando MySQL como lenguaje base y DataGrip como DBMS para así poder acceder y manipular los datos que están almacenados en ella.

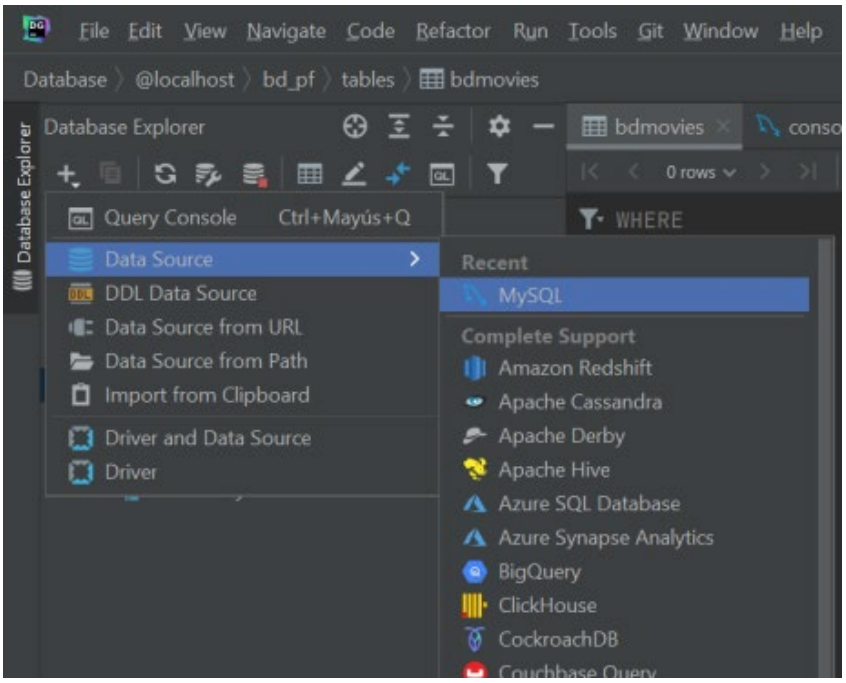


Figure 4.- Conexión a la Base de Datos

2.3 Importación del CSV

Con el archivo CSV descargado llamado “movie_dataset”, el cual fue obtenido de un repositorio de GitHub, entramos a DataGrip y seleccionamos el schema al cual se quiere importar el CSV.

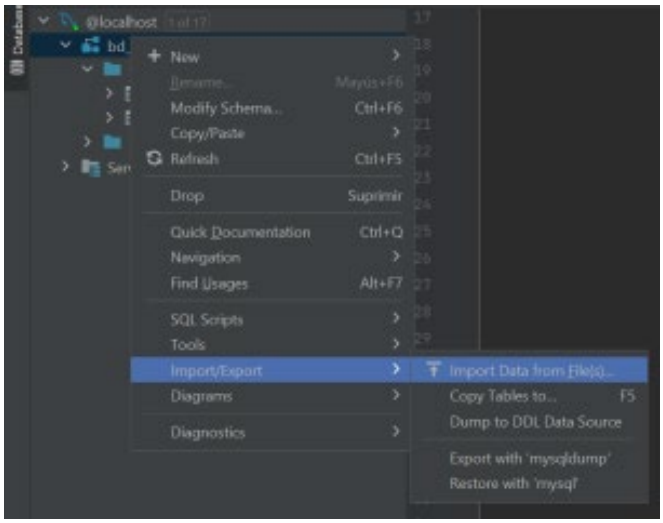


Figure 5.- Importación del CSV

DataGrip nos muestra un preview de la manera que se va a importar el CSV, donde define el tipo de dato que posee cada columna, en este caso en algunas columnas que tenían TEXT, pero su contenido era JSON tales como Crew, Production Companies, Production Countries, Spoken_Languages, se le modifico para que el tipo de dato sea JSON.

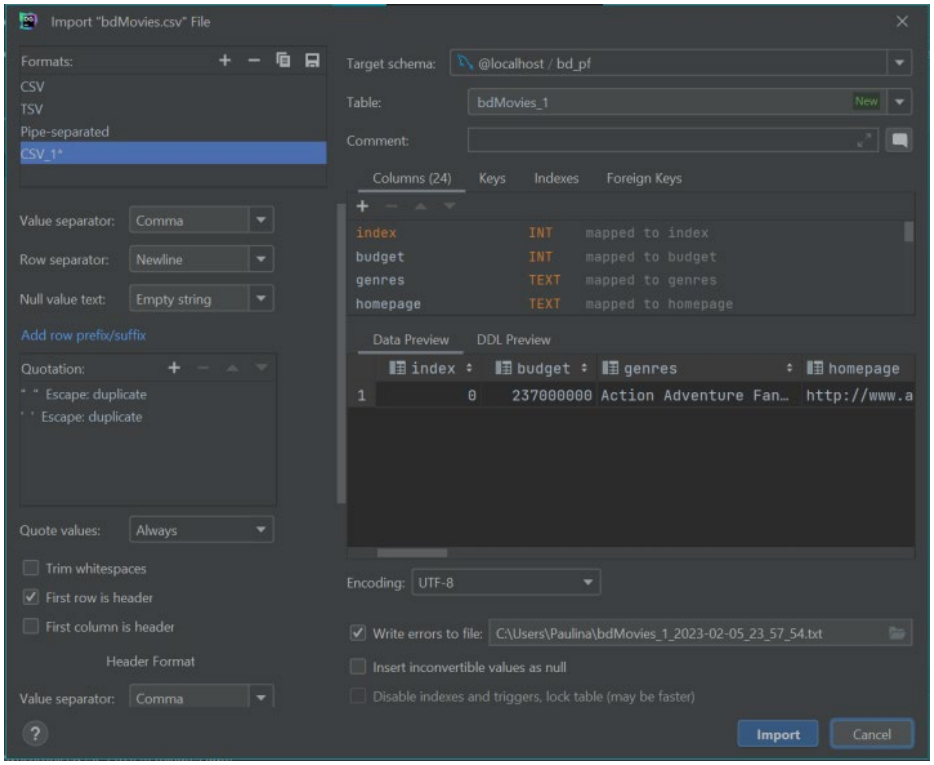


Figure 6.- Preview de Importación CSV

2.4 Creación de Tablas

A partir de la tabla universal que se obtiene al importar el CSV, se genera varias tablas siguiendo el proceso de normalización y partiendo de los modelos realizados anteriormente, los cuales nos dan una vista panorámica de lo que se debe hacer en el DDL a continuación, se observara las tablas creadas de este proceso.

En caso de los JSON se ocuparán tablas temporales, de las cuales obtendremos todos los datos de la columna específicamente, pero en este caso algunos datos saldrían duplicados. Las tablas temporales son muy útiles al momento de realizar las definitivas.

```
DROP TABLE Production_Companies_temp;
CREATE TABLE `Production_Companies_temp` (
    `id_pc` int NOT NULL,
    `Name` varchar(155) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 7.- Creación de la Tabla temporal Production Companies

```
DROP TABLE spoken_languages_temp;
CREATE TABLE `spoken_languages_temp` (
    `iso_639_1` varchar(10) NOT NULL,
    `Name` varchar(155) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 8.- Creación de la Tabla temporal Spoken Languages

```
DROP TABLE Production_Countries_temp;
CREATE TABLE `Production_Countries_temp` (
    `iso_3166_1` varchar(10) NOT NULL,
    `Name` varchar(155) NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 9.- Creación de la Tabla Temporal Production Countries

Con respecto a las tablas definitivas, para las columnas JSON usamos la tabla temporal, pero modificándola con un DISTINCT para obtener los datos únicos, en algunos casos para poder poner la PRIMARY KEY tendremos que hacer un ALTER TABLE. Como en los siguientes casos:

```
CREATE TABLE Production_Companies AS
    SELECT Distinct  id_pc,name
    FROM Production_Companies_temp ;

ALTER TABLE Production_Companies
    ADD PRIMARY KEY (id_pc);
```

Figure 10.- Tabla Definitiva Production Companies

```
CREATE TABLE spoken_languages AS
SELECT Distinct iso_639_1,name
FROM spoken_languages_temp ;

ALTER TABLE spoken_languages
ADD PRIMARY KEY (iso_639_1);
```

Figure 11.- Tabla Definitiva Spoken Languages

```
CREATE TABLE Production_Countries AS
SELECT Distinct iso_3166_1,name
FROM Production_Countries_temp ;

ALTER TABLE Production_Countries
ADD PRIMARY KEY (iso_3166_1);
```

Figure 12.- Tabla Definitiva Production Countries

Para las columnas que no fueron JSON, se procede a hacer la creación de tablas normalmente.

```sql
DROP TABLE IF EXISTS Movies;
CREATE TABLE `Movies` (
  `budget` bigint DEFAULT NULL,
  `homepage` varchar(255) DEFAULT NULL,
  `id` int NOT NULL,
  `keywords_` text DEFAULT NULL,
  `original_language` varchar(255) DEFAULT NULL,
  `original_title` varchar(255) DEFAULT NULL,
  `overview` text,
  `popularity` double DEFAULT NULL,
  `release_date` varchar(25) DEFAULT NULL,
  `revenue` bigint DEFAULT NULL,
  `runtime` varchar(255) DEFAULT NULL,
  `status` varchar(255) DEFAULT NULL,
  `tagline` varchar(255) DEFAULT NULL,
  `title` varchar(255) DEFAULT NULL,
  `vote_average` float DEFAULT NULL,
  `vote_count` int DEFAULT NULL,
  `director` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;
```

Figure 13.- Creación de la Tabla Movies con los atributos atómicos

```sql
CREATE TABLE `genres` (
  `name_genre` varchar(45) NOT NULL,
  PRIMARY KEY (`name_genre`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 14.- Creación de la Tabla Genres

```sql
CREATE TABLE `Status` (
  `status` varchar(25) NOT NULL,
  PRIMARY KEY (`status`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Figure 15.- Creación de Tabla Status

```sql
CREATE TABLE `People` (
  `id` int NOT NULL,
  `name_` varchar(25) DEFAULT NULL,
  `gender` int NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Figure 16.- Creación de la Tabla People

```sql
CREATE TABLE `Jobs` (
  `nameJob` varchar(255) NOT NULL,
  `department` varchar(25) NOT NULL,
  PRIMARY KEY (`nameJob`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Figure 17.- Creación de la Tabla Jobs

```sql
CREATE TABLE `Crew` (
  `id_People` int NOT NULL,
  `name_Job` varchar(255) NOT NULL,
  `id_mv` int NOT NULL,
  `credit_id` varchar(105) NOT NULL,
  PRIMARY KEY (`id_People`,`name_Job`,`id_mv`),
  KEY `name_Job` (`name_Job`),
  KEY `id_mv` (`id_mv`),
  CONSTRAINT `crew_ibfk_1` FOREIGN KEY (`id_People`) REFERENCES `People` (`id`),
  CONSTRAINT `crew_ibfk_2` FOREIGN KEY (`name_Job`) REFERENCES `Jobs` (`nameJob`),
  CONSTRAINT `crew_ibfk_3` FOREIGN KEY (`id_mv`) REFERENCES `Movies` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Figure 18.- Creación de la Tabla Crew

```
CREATE TABLE `Directors` (
  `id_People` int NOT NULL,
  `id_mv` int NOT NULL,
  PRIMARY KEY (`id_People`,`id_mv`),
  KEY `id_mv` (`id_mv`),
  CONSTRAINT `directors_ibfk_1` FOREIGN KEY (`id_People`) REFERENCES `Crew` (`id_People`),
  CONSTRAINT `directors_ibfk_2` FOREIGN KEY (`id_mv`) REFERENCES `Movies` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

Figure 19.- Creación de la Tabla Directors

```
CREATE TABLE genres(genres VARCHAR(100)) AS
    SELECT DISTINCT (
    SUBSTRING_INDEX(SUBSTRING_INDEX(genres,' ', 5), ' ', -1)) AS genres
    FROM movie_dataset;

    DELETE
    FROM genres
    WHERE genres IS NULL;

ALTER TABLE genres
    ADD PRIMARY KEY (genres);
```

Figure 20.- Creación de la Tabla Genres

```
DROP TABLE IF EXISTS cast_;
CREATE TABLE `cast_` (
  `id_mv` int NOT NULL,
  `id_People` int NOT NULL,
  PRIMARY KEY (`id_mv`,`id_People`),
  FOREIGN KEY (`id_mv`) REFERENCES `Movies` (`id`),
  FOREIGN KEY (`id_People`) REFERENCES `People` (`id`)
);
```

Figure 21.- Creación Tabla Cast

Las tablas de relación, las utilizaremos en relaciones de muchos a muchos, en este caso Movies que es la tabla madre, con todas las que mantiene relación, de la cuales, se sacaran las PRIMARY KEY tanto de Movies como de la otra tabla para establecer la relación.

```
CREATE TABLE `movies_sl` (
  `ISO_639_1` varchar(255) NOT NULL,
  `Id_mv` int NOT NULL,
  PRIMARY KEY (`ISO_639_1`,`Id_mv`),
  KEY `Id_mv` (`Id_mv`),
  CONSTRAINT `movies_sl_ibfk_1` FOREIGN KEY (`Id_mv`) REFERENCES `Movies` (`id`),
  CONSTRAINT `movies_sl_ibfk_2` FOREIGN KEY (`ISO_639_1`) REFERENCES spoken_languages (`ISO_639_1`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 22.- Creación de la Tabla Relación (Movies- Spoken Languages)

```
CREATE TABLE `moviesPC` (
  `id_pc` int NOT NULL,
  `id_mv` int NOT NULL,
  PRIMARY KEY (`id_mv`,`id_pc`),
  KEY `id_pc` (`id_pc`),
  CONSTRAINT `moviespc_ibfk_1` FOREIGN KEY (`id_mv`) REFERENCES `Movies` (`id`),
  CONSTRAINT `moviespc_ibfk_2` FOREIGN KEY (`id_pc`) REFERENCES `Production_Companies` (`id_pc`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 23.- Creación de la Tabla Relacional (Movies - Production Companies)

```
CREATE TABLE `Movies_Countrie` (
  `Id_mv` int NOT NULL,
  `ISO_3166_1` varchar(255) NOT NULL,
  PRIMARY KEY (`Id_mv`,`ISO_3166_1`),
  KEY `ISO_3166_1` (`ISO_3166_1`),
  CONSTRAINT `movies_countrie_ibfk_1` FOREIGN KEY (`Id_mv`) REFERENCES `Movies` (`id`),
  CONSTRAINT `movies_countrie_ibfk_2` FOREIGN KEY (`ISO_3166_1`) REFERENCES `Production_Countries` (`ISO_3166_1`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 24.- Creación de la Tabla Relacional (Movies - Production Countries)

```
CREATE TABLE `status_movie` (
  `id` int NOT NULL,
  `status` varchar(25) DEFAULT NULL,
  KEY `id` (`id`),
  KEY `status` (`status`),
  CONSTRAINT `status_movie_ibfk_1` FOREIGN KEY (`id`) REFERENCES `Movies` (`id`),
  CONSTRAINT `status_movie_ibfk_2` FOREIGN KEY (`status`) REFERENCES `Status` (`status`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 25.- Creación de la Tabla Relacional (Status - Movie)

```
CREATE TABLE `movies_genres` (
  `name_genre` varchar(45) NOT NULL,
  `id_mv` int NOT NULL,
  PRIMARY KEY (`name_genre`,`id_mv`),
  KEY `id_mv` (`id_mv`),
  CONSTRAINT `movies_genres_ibfk_1` FOREIGN KEY (`name_genre`) REFERENCES `genres` (`name_genre`),
  CONSTRAINT `movies_genres_ibfk_2` FOREIGN KEY (`id_mv`) REFERENCES `Movies` (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

Figure 26.- Creación de la Tabla Relacional (Movies - Genres)

```sql
CREATE FUNCTION limpieza(valor varchar(255))
RETURNS varchar(255)
BEGIN
    SET @cadena = REPLACE(valor, '\'', '-');
    SET @cadena = REPLACE(@cadena, 'Jean-Claude Van Damme', 'Jean-Claude Van-Damme');
    SET @cadena = REPLACE(@cadena, 'Yan-Kin', 'Yan Kin');
    SET @cadena = REPLACE(@cadena, 'Christopher Evan Welch', 'Christopher Evan');
    SET @cadena = REPLACE(@cadena, 'Freddie Prinze Jr.', 'Freddie PrinzeJr.');
    SET @cadena = REPLACE(@cadena, 'Robert Downey Jr.', 'Robert DowneyJr.');
    SET @cadena = REPLACE(@cadena, 'John C. Reilly', 'John Reilly');
    SET @cadena = REPLACE(@cadena, 'R. D. Call', 'R.D. Call');
    SET @cadena = REPLACE(@cadena, 'Madonna', 'Madonna Ciccone');
    SET @cadena = REPLACE(@cadena, 'Jean-Claude Van Damme', 'Jean-Claude Van-Damme');
    SET @cadena = REPLACE(@cadena, 'Jackie Earle Haley', 'Jackie EarleHaley');
    SET @cadena = REPLACE(@cadena, 'Larry the Cable Guy', 'Larry the-Cable-Guy');
    SET @cadena = REPLACE(@cadena, 'Billy Bob Thornton', 'Billy Bob-Thornton');
    SET @cadena = REPLACE(@cadena, 'T.I. Evan Ross', 'T.I. Evan-Ross');
    SET @cadena = REPLACE(@cadena, 'Brandon T. Jackson', 'Brandon T.Jackson');
    SET @cadena = REPLACE(@cadena, 'Jamie Lee Curtis', ' Jamie Curtis');
    SET @cadena = REPLACE(@cadena, 'Anthony Dileo Jr.', 'Anthony DileoJr.');
    SET @cadena = REPLACE(@cadena, 'Patrick St. Esprit', 'Patrick St.Esprit');
    SET @cadena = REPLACE(@cadena, 'Jeffrey Dean Morgan', 'Jeffrey Morgan');
    SET @cadena = REPLACE(@cadena, 'Cohen David Schwimmer', 'Cohen Schwimmer');
    SET @cadena = REPLACE(@cadena, 'Benicio del Toro', 'Benicio del-Toro');
    SET @cadena = REPLACE(@cadena, 'Rock Cedric the Entertainer', 'Rock-Cedric the-Entertainer');
    SET @cadena = REPLACE(@cadena, 'Jonathan Rhys Meyers', 'Jonathan Rhys-Meyers');
    SET @cadena = REPLACE(@cadena, 'Helena Bonham Carter', 'Helena Bonham-Carter');
    SET @cadena = REPLACE(@cadena, 'Bryce Dallas Howard', 'Bryce Dallas-Howard');
    SET @cadena = REPLACE(@cadena, 'Khan Vincent D-Onofrio', 'Khan-Vincent D-Onofrio');
    SET @cadena = REPLACE(@cadena, 'Rihanna', 'Rihanna Rihanna');
    SET @cadena = REPLACE(@cadena, 'Dakota Blue Richards', 'Dakota Blue-Richards"');
    SET @cadena = REPLACE(@cadena, 'Tommy Lee Jones', 'Tommy Lee-Jones');
    SET @cadena = REPLACE(@cadena, 'Thomas Haden Church', 'Thomas Haden-Church');
    SET @cadena = REPLACE(@cadena, 'K. D. Aubert', 'K.D. Aubert');
    SET @cadena = REPLACE(@cadena, 'Irma P. Hall', 'Irma P.Hall');
    SET @cadena = REPLACE(@cadena, 'Jonathan Silverman Eve', 'Jonathan Silverman-Eve');
    SET @cadena = REPLACE(@cadena, 'Emilie de Ravin', 'Emilie de-Ravin');
    SET @cadena = REPLACE(@cadena, ' Jamie', 'Jamie');
    SET @cadena = REPLACE(@cadena, 'Robert De Niro', 'Robert De-Niro');
    SET @cadena = REPLACE(@cadena, 'Cole Hauser Ludacris', 'Cole Hauser-Ludacris');
    SET @cadena = REPLACE(@cadena, 'Michael B. Jordan"', 'MichaelB. Jordan');
    SET @cadena = REPLACE(@cadena, 'Jennifer Jason Leigh','Jennifer Jason_Leigh');
    SET @cadena = REPLACE(@cadena, '"von","von_"');
    SET @cadena = REPLACE(@cadena, 'Jr.','_Jr');
    SET @cadena = REPLACE(@cadena,' "de la"','de_la_');
    SET @cadena = REPLACE(@cadena, 'Jo','Jo_');
    SET @cadena = REPLACE(@cadena, 'Don','Don_');
    SET @cadena = REPLACE(@cadena, 'LL Cool J','LL Cool_J');
    SET @cadena = REPLACE(@cadena, 'Pelé',' Pelé _____');
    SET @cadena = REPLACE(@cadena, 'Pink',' Pink _____');
    SET @cadena = REPLACE(@cadena, 'van den','van_den_');

    RETURN @cadena;
END;
```

Figure 27.- Limpieza de Cast

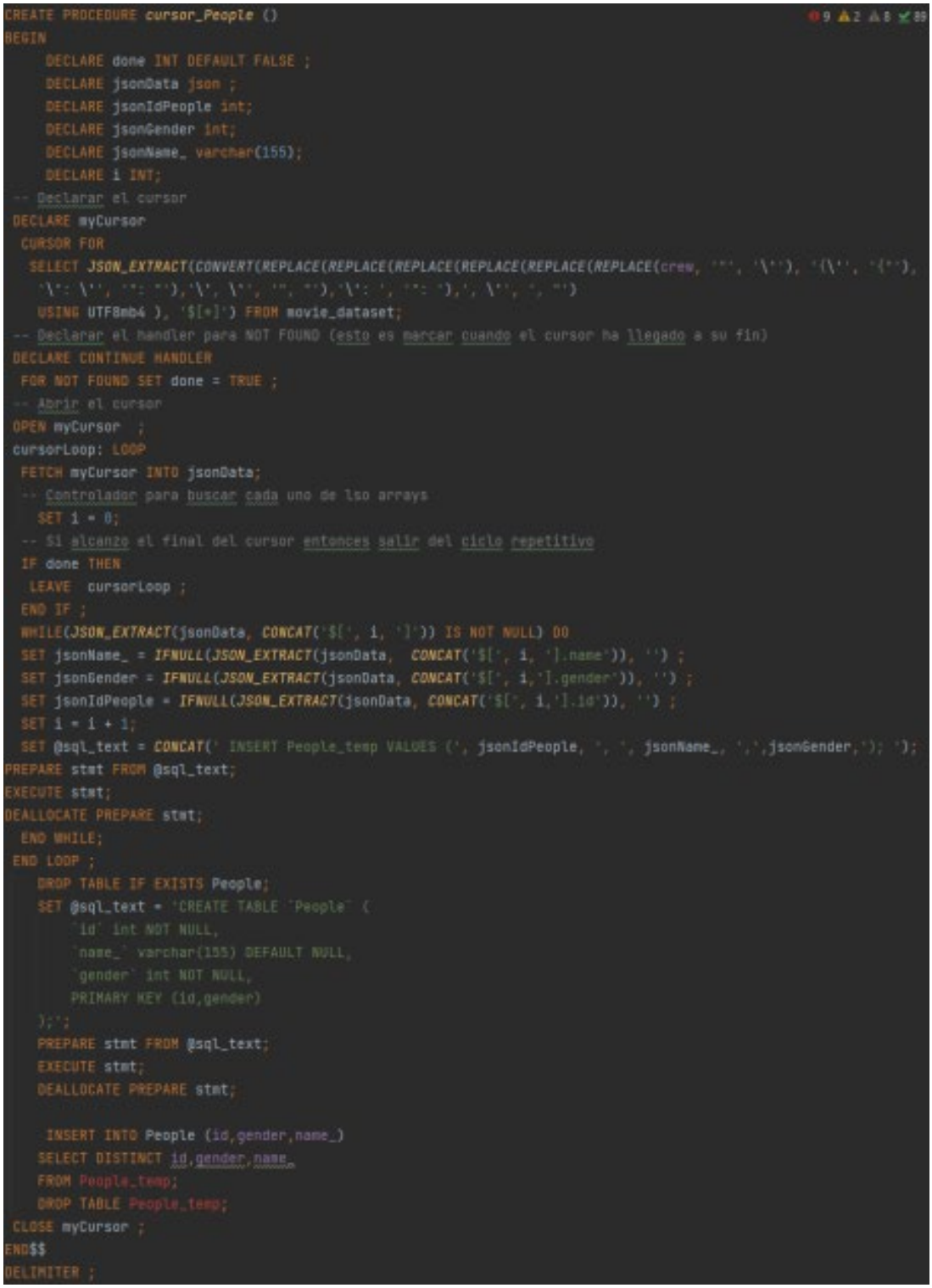
2.5 Creación de procedimientos y limpieza de datos

Un procedure es un proceso cuyo objetivo es que sea llamado en múltiples ocasiones en nuestro código, esto otorga ciertas ventajas relacionadas a rendimiento y optimización.

Para el proyecto, se crearon distintos procedimientos que nos permitieron extraer los datos de las columnas para poder insertar en las tablas correspondientes, en medio de estos procedimientos se limpian aquellos datos que lo necesiten, ya que algunas partes del CSV vienen mal formadas o con caracteres especiales que no son admitidos en la inserción de los datos.

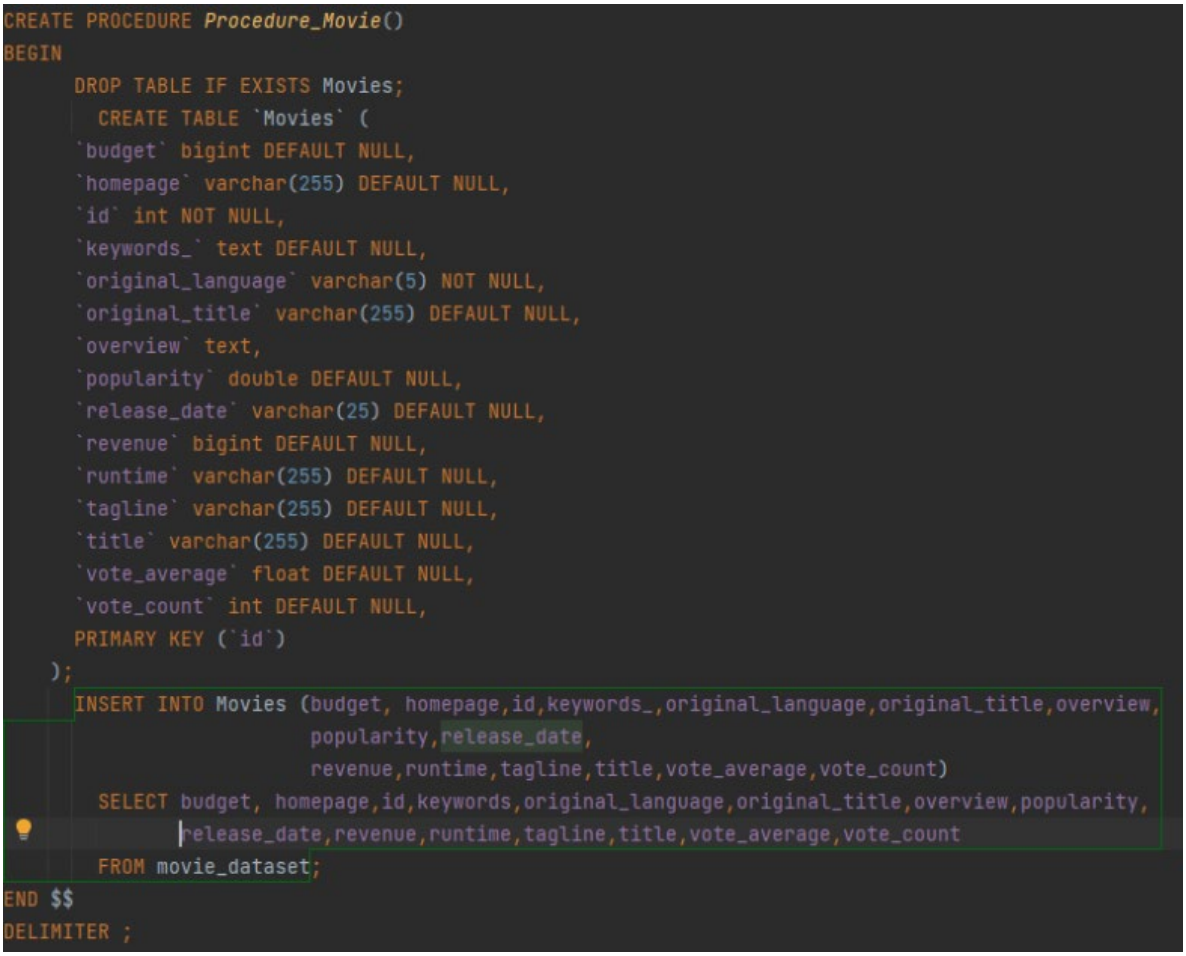
```
CREATE PROCEDURE creacion_tablas ()
BEGIN
    DROP TABLE IF EXISTS Production_Countries_temp;
    SET @sql_text = 'CREATE TABLE Production_Countries_temp (
        iso_3166_1 varchar(10) NOT NULL,
        Name varchar(155) NOT NULL
    );';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DROP TABLE IF EXISTS production_Companies_temp;
    SET @sql_text = 'CREATE TABLE Production_Companies_temp (
        id_pc int NOT NULL,
        Name varchar(155) NOT NULL
    );';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DROP TABLE IF EXISTS spoken_languages_temp;
    SET @sql_text = 'CREATE TABLE spoken_languages_temp (
        iso_639_1 varchar(10) NOT NULL,
        Name varchar(155) DEFAULT NULL
    );';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DROP TABLE IF EXISTS People_temp;
    SET @sql_text = 'CREATE TABLE `People_temp` (
        `id` int NOT NULL,
        `name_` varchar(155) DEFAULT NULL,
        `gender` int NOT NULL
    );';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;

END$$
DELIMITER ;
```

Figure 28.- Creación del Procedimiento "Creación Tablas"



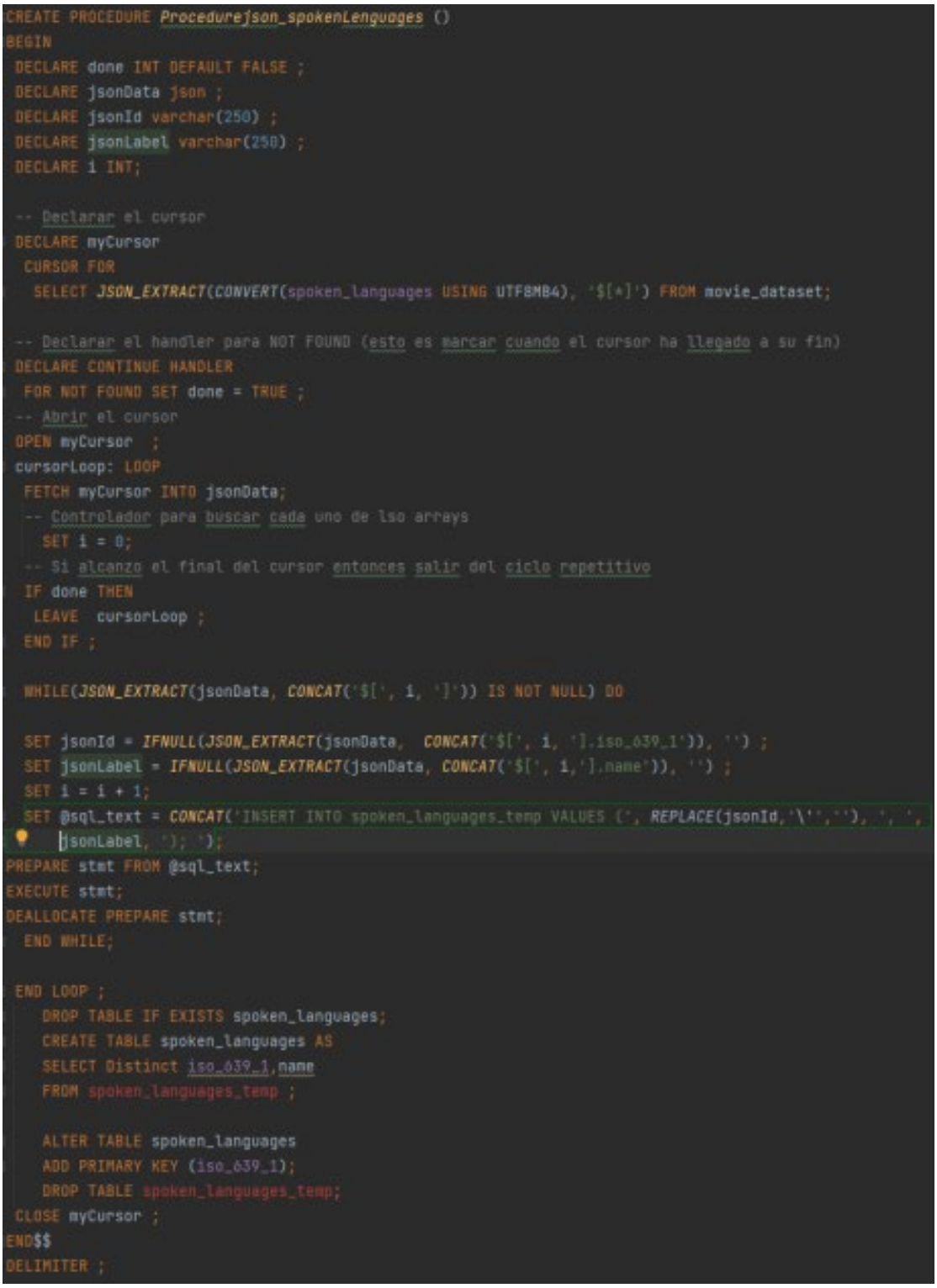
```
CREATE PROCEDURE cursor_People ()
BEGIN
    DECLARE done INT DEFAULT FALSE ;
    DECLARE jsonData json ;
    DECLARE jsonIdPeople int;
    DECLARE jsonGender int;
    DECLARE jsonName_ varchar(155);
    DECLARE i INT;
 -- Declarar el cursor
 DECLARE myCursor
  CURSOR FOR
   SELECT JSON_EXTRACT(CONVERT(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(REPLACE(crew, '"', '\''), '{\'', '{"'),
    '\': \'', '": "'),'\', \'', '", "'),'\': ', '": '),', \'', ', "')
    USING UTF8mb4 ), '$[*]') FROM movie_dataset;
 -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
 DECLARE CONTINUE HANDLER
  FOR NOT FOUND SET done = TRUE ;
 -- Abrir el cursor
 OPEN myCursor  ;
 cursorLoop: LOOP
  FETCH myCursor INTO jsonData;
  -- Controlador para buscar cada uno de lso arrays
    SET i = 0;
  -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
  IF done THEN
   LEAVE  cursorLoop ;
  END IF ;
  WHILE(JSON_EXTRACT(jsonData, CONCAT('$[', i, ']')) IS NOT NULL) DO
  SET jsonName_ = IFNULL(JSON_EXTRACT(jsonData,  CONCAT('$[', i, '].name')), '') ;
  SET jsonGender = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i,'].gender')), '') ;
  SET jsonIdPeople = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i,'].id')), '') ;
  SET i = i + 1;
  SET @sql_text = CONCAT(' INSERT People_temp VALUES (', jsonIdPeople, ', ', jsonName_, ',',jsonGender,'); ');
PREPARE stmt FROM @sql_text;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
  END WHILE;
 END LOOP ;
    DROP TABLE IF EXISTS People;
    SET @sql_text = 'CREATE TABLE `People` (
        `id` int NOT NULL,
        `name_` varchar(155) DEFAULT NULL,
        `gender` int NOT NULL,
       PRIMARY KEY (id,gender)
    );';
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;

     INSERT INTO People (id,gender,name_)
    SELECT DISTINCT id,gender,name_
    FROM People_temp;
    DROP TABLE People_temp;
 CLOSE myCursor ;
END$$
DELIMITER ;
```

Figure 29.- Creación del Procedimiento "Cursor - People"



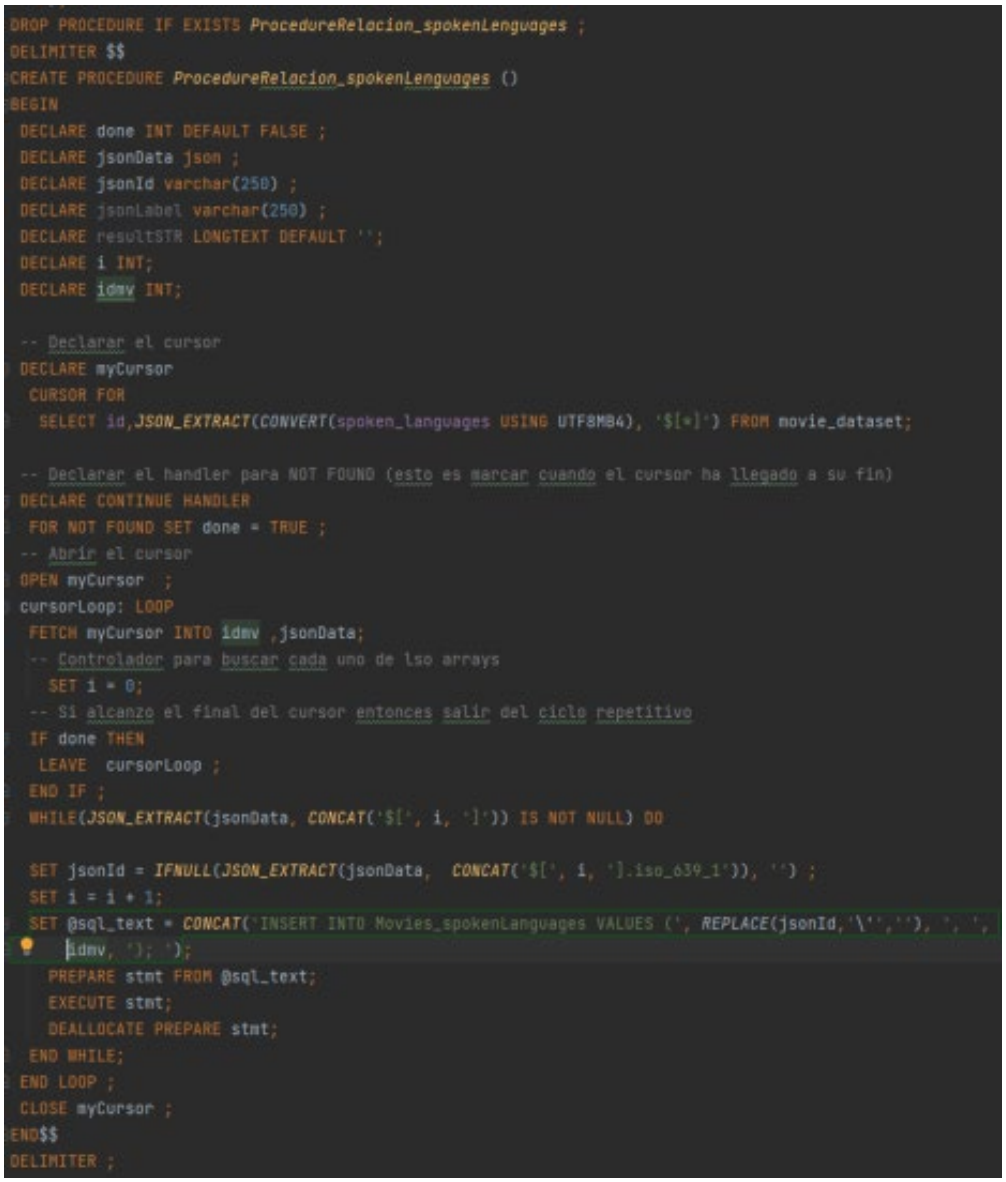
```
CREATE PROCEDURE Procedure_Movie()
BEGIN
    DROP TABLE IF EXISTS Movies;
      CREATE TABLE `Movies` (
    `budget` bigint DEFAULT NULL,
    `homepage` varchar(255) DEFAULT NULL,
    `id` int NOT NULL,
    `keywords_` text DEFAULT NULL,
    `original_language` varchar(5) NOT NULL,
    `original_title` varchar(255) DEFAULT NULL,
    `overview` text,
    `popularity` double DEFAULT NULL,
    `release_date` varchar(25) DEFAULT NULL,
    `revenue` bigint DEFAULT NULL,
    `runtime` varchar(255) DEFAULT NULL,
    `tagline` varchar(255) DEFAULT NULL,
    `title` varchar(255) DEFAULT NULL,
    `vote_average` float DEFAULT NULL,
    `vote_count` int DEFAULT NULL,
    PRIMARY KEY (`id`)
  );
    INSERT INTO Movies (budget, homepage,id,keywords_,original_language,original_title,overview,
                        popularity,release_date,
                        revenue,runtime,tagline,title,vote_average,vote_count)
     SELECT budget, homepage,id,keywords,original_language,original_title,overview,popularity,
            release_date,revenue,runtime,tagline,title,vote_average,vote_count
     FROM movie_dataset;
END $$
DELIMITER ;
```

Figure 30.- Creación del Procedimiento "Procedure Movie"



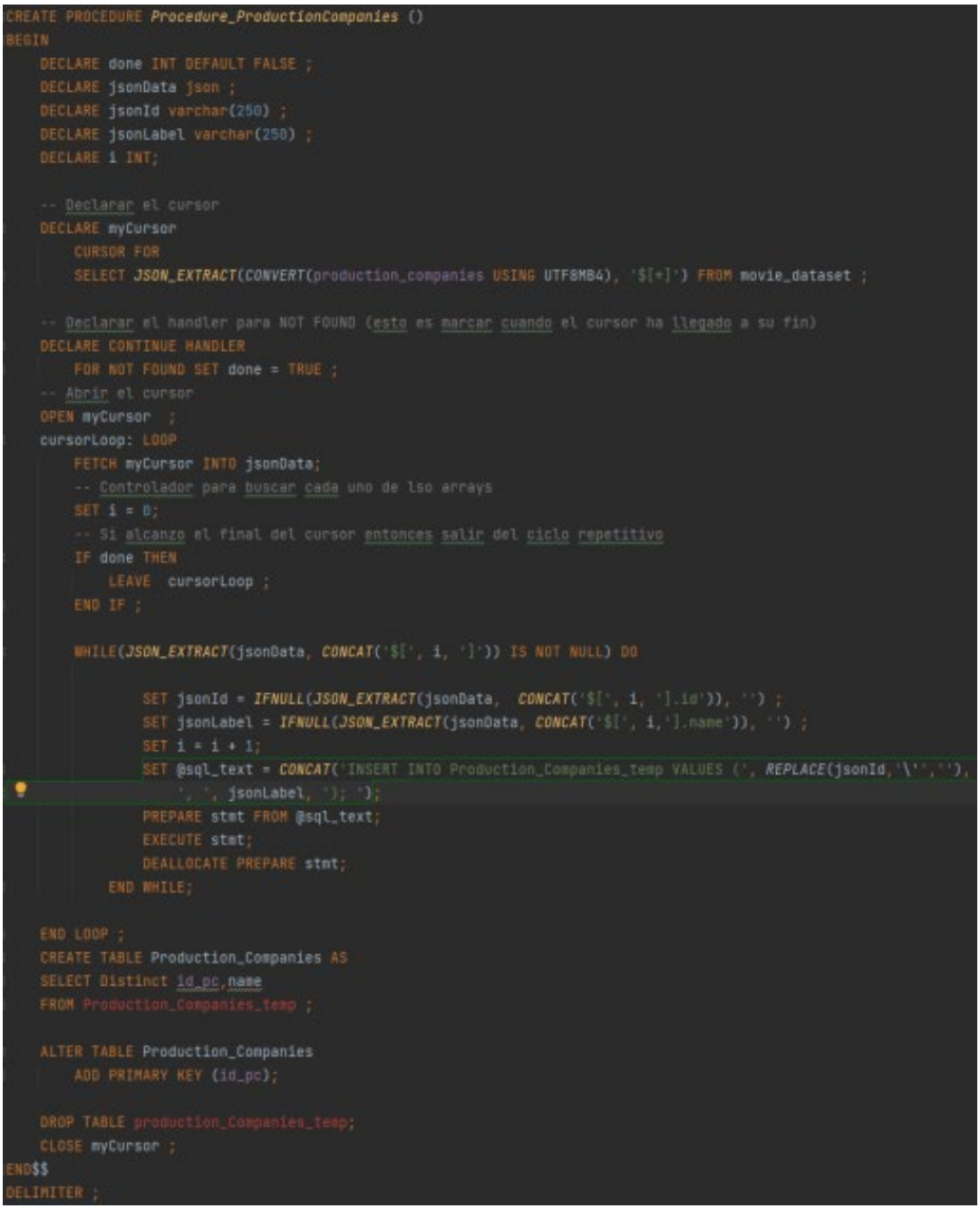
```
CREATE PROCEDURE Procedurejson_spokenLenguages ()
BEGIN
 DECLARE done INT DEFAULT FALSE ;
 DECLARE jsonData json ;
 DECLARE jsonId varchar(250) ;
 DECLARE jsonLabel varchar(250) ;
 DECLARE i INT;

 -- Declarar el cursor
 DECLARE myCursor
  CURSOR FOR
   SELECT JSON_EXTRACT(CONVERT(spoken_languages USING UTF8MB4), '$[*]') FROM movie_dataset;

 -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
 DECLARE CONTINUE HANDLER
  FOR NOT FOUND SET done = TRUE ;
 -- Abrir el cursor
 OPEN myCursor  ;
 cursorLoop: LOOP
  FETCH myCursor INTO jsonData;
  -- Controlador para buscar cada uno de lso arrays
    SET i = 0;
  -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
  IF done THEN
   LEAVE  cursorLoop ;
  END IF ;

  WHILE(JSON_EXTRACT(jsonData, CONCAT('$[', i, ']')) IS NOT NULL) DO

  SET jsonId = IFNULL(JSON_EXTRACT(jsonData,  CONCAT('$[', i, '].iso_639_1')), '') ;
  SET jsonLabel = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i,'].name')), '') ;
  SET i = i + 1;
  SET @sql_text = CONCAT('INSERT INTO spoken_languages_temp VALUES (', REPLACE(jsonId,'\'',''), ', ',
    jsonLabel, '); ');
PREPARE stmt FROM @sql_text;
EXECUTE stmt;
DEALLOCATE PREPARE stmt;
  END WHILE;

 END LOOP ;
    DROP TABLE IF EXISTS spoken_languages;
    CREATE TABLE spoken_languages AS
    SELECT Distinct iso_639_1,name
    FROM spoken_languages_temp ;

    ALTER TABLE spoken_languages
    ADD PRIMARY KEY (iso_639_1);
    DROP TABLE spoken_languages_temp;
 CLOSE myCursor ;
END$$
DELIMITER ;
```

Figure 31.- Creación del Procedimiento " ProcedureJSON Spoken Languages"



```
DROP PROCEDURE IF EXISTS ProcedureRelacion_spokenLenguages ;
DELIMITER $$
CREATE PROCEDURE ProcedureRelacion_spokenLenguages ()
BEGIN
 DECLARE done INT DEFAULT FALSE ;
 DECLARE jsonData json ;
 DECLARE jsonId varchar(250) ;
 DECLARE jsonLabel varchar(250) ;
 DECLARE resultSTR LONGTEXT DEFAULT '';
 DECLARE i INT;
 DECLARE idmv INT;

 -- Declarar el cursor
 DECLARE myCursor
  CURSOR FOR
   SELECT id,JSON_EXTRACT(CONVERT(spoken_languages USING UTF8MB4), '$[*]') FROM movie_dataset;

 -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
 DECLARE CONTINUE HANDLER
  FOR NOT FOUND SET done = TRUE ;
 -- Abrir el cursor
 OPEN myCursor  ;
 cursorLoop: LOOP
  FETCH myCursor INTO idmv ,jsonData;
  -- Controlador para buscar cada uno de lso arrays
    SET i = 0;
  -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
  IF done THEN
   LEAVE  cursorLoop ;
  END IF ;
  WHILE(JSON_EXTRACT(jsonData, CONCAT('$[', i, ']')) IS NOT NULL) DO

  SET jsonId = IFNULL(JSON_EXTRACT(jsonData,  CONCAT('$[', i, '].iso_639_1')), '') ;
  SET i = i + 1;
  SET @sql_text = CONCAT('INSERT INTO Movies_spokenLanguages VALUES (', REPLACE(jsonId,'\'',''), ', ',
    idmv, '); ');
    PREPARE stmt FROM @sql_text;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
  END WHILE;
 END LOOP ;
 CLOSE myCursor ;
END$$
DELIMITER ;
```

Figure 32.- Creación Procedimiento "Procedure Relacion Spoken Languages"



```
CREATE PROCEDURE Procedure_ProductionCompanies ()
BEGIN
    DECLARE done INT DEFAULT FALSE ;
    DECLARE jsonData json ;
    DECLARE jsonId varchar(250) ;
    DECLARE jsonLabel varchar(250) ;
    DECLARE i INT;

    -- Declarar el cursor
    DECLARE myCursor
        CURSOR FOR
        SELECT JSON_EXTRACT(CONVERT(production_companies USING UTF8MB4), '$[*]') FROM movie_dataset ;

    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER
        FOR NOT FOUND SET done = TRUE ;
    -- Abrir el cursor
    OPEN myCursor  ;
    cursorLoop: LOOP
        FETCH myCursor INTO jsonData;
        -- Controlador para buscar cada uno de lso arrays
        SET i = 0;
        -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
        IF done THEN
            LEAVE  cursorLoop ;
        END IF ;

        WHILE(JSON_EXTRACT(jsonData, CONCAT('$[', i, ']')) IS NOT NULL) DO

                SET jsonId = IFNULL(JSON_EXTRACT(jsonData,  CONCAT('$[', i, '].id')), '') ;
                SET jsonLabel = IFNULL(JSON_EXTRACT(jsonData, CONCAT('$[', i,'].name')), '') ;
                SET i = i + 1;
                SET @sql_text = CONCAT('INSERT INTO Production_Companies_temp VALUES (', REPLACE(jsonId,'\'',''),
                    ', ', jsonLabel, '); ');
                PREPARE stmt FROM @sql_text;
                EXECUTE stmt;
                DEALLOCATE PREPARE stmt;
            END WHILE;

    END LOOP ;
    CREATE TABLE Production_Companies AS
    SELECT Distinct id_pc,name
    FROM Production_Companies_temp ;

    ALTER TABLE Production_Companies
        ADD PRIMARY KEY (id_pc);

    DROP TABLE production_Companies_temp;
    CLOSE myCursor ;
END$$
DELIMITER ;
```

Figure 33.- Creación Procedimiento "Procedure Production Companies"

```sql
DROP PROCEDURE IF EXISTS ProcedureRelacion_ProductionCompanies ;
DELIMITER $$
CREATE PROCEDURE ProcedureRelacion_ProductionCompanies ()
BEGIN
    DECLARE done INT DEFAULT FALSE ;
    DECLARE jsonData json ;
    DECLARE jsonId varchar(250) ;
    DECLARE i INT;
    DECLARE idmv INT;

    -- Declarar el cursor
    DECLARE myCursor
        CURSOR FOR
        SELECT id,JSON_EXTRACT(CONVERT(production_companies USING UTF8MB4), '$[*]') FROM movie_dataset;

    -- Declarar el handler para NOT FOUND (esto es marcar cuando el cursor ha llegado a su fin)
    DECLARE CONTINUE HANDLER
        FOR NOT FOUND SET done = TRUE ;
    -- Abrir el cursor
    OPEN myCursor  ;
    cursorLoop: LOOP
        FETCH myCursor INTO idmv ,jsonData;
        -- Controlador para buscar cada uno de lso arrays
        SET i = 0;
        -- Si alcanzo el final del cursor entonces salir del ciclo repetitivo
        IF done THEN
            LEAVE  cursorLoop ;
        END IF ;

        WHILE(JSON_EXTRACT(jsonData, CONCAT('$[', i, ']')) IS NOT NULL) DO

                SET jsonId = IFNULL(JSON_EXTRACT(jsonData,  CONCAT('$[', i, '].id')), '') ;
                SET i = i + 1;
                SET @sql_text = CONCAT('INSERT INTO Movies_ProductionCompanies VALUES (',
                    REPLACE(jsonId,'\'',''), ', ', idmv, '); ');
                PREPARE stmt FROM @sql_text;
                EXECUTE stmt;
                DEALLOCATE PREPARE stmt;
            END WHILE;

    END LOOP ;

    CLOSE myCursor ;
END$$
DELIMITER ;
```

Figure 34.- Creación Procedimiento "Procedure Relacion Production Companies"

2.5.1 Ejecución de procedimientos

Se hace un llamado a los procedimientos creados anteriormente, los cuales tienen funciones de limpieza e inserción de datos de cada columna respectivamente.

```
CALL creacion_tablas();
```

Figure 35.- Llamado al Procedimiento "Creacion tablas"

```
CALL cursor_People();
```

Figure 36.- Llamado al Procedimiento "Cursor People"

```
CALL Procedure_Movie();
```

Figure 37.- Llamado al Procedimiento "Procedure Movie"

```
CALL Procedurejson_spokenLenguages ();
```

Figure 38.- Llamado al Procedimiento "Procedure JSON Spoken Languages"

```
CALL ProcedureRelacion_spokenLenguages ();
```

Figure 39.- Llamado al Procedimiento "Procedure Relacion Spoken Languages"

```
CALL Procedure_ProductionCompanies ();
```

Figure 40.- Llamado al Procedimiento "Procedure Production Companies"

```
CALL ProcedureRelacion_ProductionCompanies ();
```

Figure 41.- Llamado al Procedimiento "Procedure Relacion Production Companies"

3 Conclusiones

- El lenguaje MySQL es muy importante para llevar a cabo cada una de las actividades de corrección, relación, extracción e ingreso de datos del CSV, el cual, si bien presentaba varios errores en la formación de los JSON o la formación de cadenas en varias columnas,

con dichas herramientas fue posible brindar una resolución que nos permita trabajar con dichos datos.

- Existen varias herramientas DBMS, sin embargo, refiriéndonos específicamente a las herramientas a nuestra disposición a lo largo del desarrollo de este proyecto integrador tanto Workbench como DataGrip son dos excelentes herramientas de trabajo, pero DataGrip al contrario de Workbench representa un mayor detalle tanto en la detección de errores, como en las opciones de importación presentadas, este tipo de herramientas facilita el proceso de desarrollo de dicho trabajo al otorgar un mayor detalle respecto a los errores que se comete conforme se avanza en la codificación de las soluciones.