|  |
| --- |
|  |
|  |  |



# C1 DESARROLLO DE CASO

**TERCER PARCIAL**

**INGENIERIA EN SISTEMAS**

**BASE DE DATOS 2**

# C1 DESARROLLO DE CASO

**TERCER PARCIAL**

**INGENIERIA EN SISTEMAS**

**BASE DE DATOS 2**

**UNIVERSIDAD NACIONAL AUTÓNOMA DE HONDURAS**

**CENTRO UNIVERSITARIO REGIONAL DEL LITORAL PACÍFICO**

**(UNAH-CURLP)**

**Bases de Datos II**

**(IS601)**

**Docente: Ing. Oscar Omar Pineda, MSc**

**Integrantes:**

**Ernesto Noe Moncada 20202300046**

**Adair Humberto Flores 20192300102**

**Josué Daniel Henríquez 20202300042**

**Esdras Rigoberto Castillo 20172300218**

**Francisco Josafat Paz Flores 20212300157**

**Kenis Noe Osorto Reyes 20212300177**

**Choluteca abril 2024**

# CONTENIDO

[C1 DESARROLLO DE CASO 1](#_Toc164629009)

[C1 DESARROLLO DE CASO 1](#_Toc164629010)

[1 CONTENIDO 3](#_Toc164629011)

[2 TABLA DE ILUSTRACIONES 6](#_Toc164629012)

[3 INTRODUCCIÓN 7](#_Toc164629013)

[4 OBJETIVOS 8](#_Toc164629014)

[4.1 General 8](#_Toc164629015)

[4.2 Específicos 8](#_Toc164629016)

[5 PostGreSQL 9](#_Toc164629017)

[5.1 ¿Qué es PostgreSQL? 9](#_Toc164629018)

[5.1.1 Nombre de la mascota 9](#_Toc164629019)

[5.1.2 Características: 10](#_Toc164629020)

[5.2 Breve historia de PostgreSQL 12](#_Toc164629021)

[5.2.1 El proyecto POSTGRES de Berkeley 12](#_Toc164629022)

[5.2.2 Postgres95 14](#_Toc164629023)

[5.2.3 PostgreSQL 15](#_Toc164629024)

[5.3 Documentación 16](#_Toc164629025)

[5.3.1 Tipos de datos en PostgreSQL 16](#_Toc164629026)

[5.3.2 Creación de una base de datos 18](#_Toc164629027)

[5.3.3 Creación de una nueva tabla 19](#_Toc164629028)

[5.3.4 Rellenar una tabla con filas 20](#_Toc164629029)

[5.3.5 Consultar una tabla 20](#_Toc164629030)

[5.3.6 Uniones entre tablas 21](#_Toc164629031)

[5.3.7 Funciones agregadas 24](#_Toc164629032)

[5.3.8 Actualizaciones 26](#_Toc164629033)

[5.3.9 Eliminaciones 27](#_Toc164629034)

[5.3.10 Vistas 27](#_Toc164629035)

[5.3.11 Claves foráneas 28](#_Toc164629036)

[5.4 Ranking 30](#_Toc164629037)

[5.5 pgAdmin 4 31](#_Toc164629038)

[6 Modelado de Caso FP 33](#_Toc164629039)

[6.1 Instrucciones 33](#_Toc164629040)

[6.2 Pasos para modelado de la base de datos 40](#_Toc164629041)

[6.2.1 Nombrar la base de datos 40](#_Toc164629042)

[6.2.2 Identificar las Entidades por Schema 40](#_Toc164629043)

[6.2.3 Identificar los atributos de las entidades 41](#_Toc164629044)

[6.3 Diccionario de datos 42](#_Toc164629045)

[6.3.1 Tabla CompaniaProductora 42](#_Toc164629046)

[6.3.2 Tabla EspacioEspecial 43](#_Toc164629047)

[6.3.3 Tabla clasificación 44](#_Toc164629048)

[6.3.4 Tabla genero 45](#_Toc164629049)

[6.3.5 Tabla pelicula 46](#_Toc164629050)

[6.3.6 Tabla programación 47](#_Toc164629051)

[6.3.7 Tabla película\_compania\_productora 48](#_Toc164629052)

[6.3.8 Tabla espacio\_especial\_pelicula 48](#_Toc164629053)

[6.3.9 Tabla profesion 49](#_Toc164629054)

[6.3.10 Tabla nacionalidad 50](#_Toc164629055)

[6.3.11 Tabla artista 51](#_Toc164629056)

[6.3.12 Tabla sitie 52](#_Toc164629057)

[6.3.13 Tabla tipo\_cargo 53](#_Toc164629058)

[6.3.14 Tabla pelicula\_artista\_cargo 54](#_Toc164629059)

[6.3.15 Tabla personaje 55](#_Toc164629060)

[6.4 Implementar el Modelo Conceptual en PostgreSQL 56](#_Toc164629061)

[6.4.1 Crear Base de datos 56](#_Toc164629062)

[6.5 Script DDL 59](#_Toc164629063)

[6.6 DML de Inserciones 63](#_Toc164629064)

[6.7 Funciones almacenadas (CRUD) 87](#_Toc164629065)

[6.8 Funciones almacenadas Adicionales 113](#_Toc164629066)

[6.9 Implementación de las tablas de auditoria(Audilog) con sus respectivos disparadores(Trigger) 134](#_Toc164629067)

[6.10 Disparadores(Triggers) especiales 140](#_Toc164629068)

[7 Diagrama Entidad Relación 143](#_Toc164629069)

[8 Diagrama Relacional 144](#_Toc164629070)

[9 Referencias 145](#_Toc164629071)

# TABLA DE ILUSTRACIONES

[Ilustración 1. Ranking de motores de bases de datos 30](#_Toc164611079)

[Ilustración 2.Ranking de motores de bases de datos 30](#_Toc164611080)

[Ilustración 3. vista diseño de la tabla compania\_productora 43](#_Toc164611081)

[Ilustración 4. Vista diseño de tabla espacio\_especial 44](#_Toc164611082)

[Ilustración 5. Vista diseño de tabla clasificacion 44](#_Toc164611083)

[Ilustración 6. Vista diseño de tabla genero 45](#_Toc164611084)

[Ilustración 7. Vista diseño de tabla pelicula 46](#_Toc164611085)

[Ilustración 8. Vista diseño de tabla pelicula\_compania\_productora 48](#_Toc164611086)

[Ilustración 9. Vista diseño de tabla espacio\_especial\_pelicula 49](#_Toc164611087)

[Ilustración 10. Vista diseño de tabla profesion 50](#_Toc164611088)

[Ilustración 11. Vista diseño de tabla nacionalidad 50](#_Toc164611089)

[Ilustración 12. Vista diseño de tabla artista 51](#_Toc164611090)

[Ilustración 13. Vista diseño de tabla sitie 52](#_Toc164611091)

[Ilustración 14. Vista diseño de tabla tipo\_cargo 53](#_Toc164611092)

[Ilustración 15. Vista diseño de tabla pelicula\_artista\_cargo 54](#_Toc164611093)

[Ilustración 16. Vista diseño de tabla personaje 55](#_Toc164611094)

[Ilustración 17. Paso 1 para crear base de datos 56](#_Toc164611095)

[Ilustración 18. Paso 2 para crear base de datos 57](#_Toc164611096)

[Ilustración 19. Paso 3 para crear base de datos 57](#_Toc164611097)

[Ilustración 20. Paso 4 para crear base de datos 58](#_Toc164611098)

[Ilustración 21. Paso 5 para crear base de datos 58](#_Toc164611099)

[Ilustración 22. Paso 6 para crear base de datos 59](#_Toc164611100)

[Ilustración 23. Diagrama Entidad Relacion 143](#_Toc164611101)

[Ilustración 24. Diagrama Relacional 144](#_Toc164611102)

# INTRODUCCIÓN

Desde su lanzamiento PostgreSQL ha evolucionado como un sistema de gestión de bases de datos relacional de código abierto, destacándose por su flexibilidad y su habilidad para adaptarse a una diversidad de entornos y necesidades de datos. Gracias a su arquitectura altamente modular y una rica variedad de características, PostgreSQL sobresale en la gestión tanto de transacciones críticas como en el análisis de datos complejos.

La comunidad activa que rodea a PostgreSQL asegura un proceso de desarrollo sólido y continuo, lo que garantiza que la base de datos se mantenga relevante y actualizada frente a las demandas cambiantes del mundo de la gestión de datos. Esta colaboración continua también se refleja en el extenso soporte ofrecido, que garantiza la confiabilidad y la estabilidad de PostgreSQL a largo plazo.

PostgreSQL no solo se destaca por proporcionar una base sólida para el modelado de datos y la aplicación de restricciones de integridad referencial, sino que también impulsa una cultura de colaboración y contribución. Este enfoque promueve el desarrollo de un ecosistema de software dinámico y accesible. En proyectos como la programación mensual del canal de TV por cable CINEX con el propósito de publicar la programación en Internet, PostgreSQL se convierte en una opción confiable, permitiendo el diseño de una estructura de base de datos eficiente y segura, adaptable a las necesidades específicas del negocio.

# OBJETIVOS

## General

* Diseñar e implementar una base de datos normalizada en PostgreSQL para la programación mensual del canal de TV por cable CINEX, con el propósito de publicar la programación en Internet, asegurando una gestión eficiente y precisa de la información relacionada con los programas, horarios, géneros y fechas de emisión.

## Específicos

* Ingresar datos de prueba que representen diversos escenarios de programación, garantizando su conformidad con las restricciones y relaciones establecidas.
* Realizar consultas de validación para asegurar que la integridad referencial se mantenga intacta, incluso con datos simulados de muestra.

# PostGreSQL

## ¿Qué es PostgreSQL?

PostgreSQL es un potente sistema de base de datos relacional de objetos de código abierto que utiliza y amplía el lenguaje SQL combinado con muchas funciones que almacenan y escalan de forma segura las cargas de trabajo de datos más complicadas. Los orígenes de PostgreSQL se remontan a 1986 como parte del proyecto POSTGRES de la Universidad de California en Berkeley y cuenta con más de 35 años de desarrollo activo en la plataforma central (PostgreSQL, PostgreSQL, 2024).

PostgreSQL se ha ganado una sólida reputación por su arquitectura probada, confiabilidad, integridad de datos, conjunto de características robustas, extensibilidad y la dedicación de la comunidad de código abierto detrás del software para ofrecer constantemente soluciones innovadoras y de alto rendimiento. PostgreSQL se ejecuta en los principales sistemas operativos, ha sido compatible con ACID desde 2001 y tiene potentes complementos como el popular extensor de base de datos geoespacial PostGIS. No es de extrañar que PostgreSQL se haya convertido en la base de datos relacional de código abierto elegida por muchas personas y organizaciones (PostgreSQL, PostgreSQL, 2024).

Comenzar a usar PostgreSQL nunca ha sido tan fácil: elija un proyecto que desee crear y deje que PostgreSQL almacene sus datos de forma segura y sólida (PostgreSQL, PostgreSQL, 2024).

### Nombre de la mascota

La mascota de PostgreSQL se llama Slonik (Herrera, 2007).

### Características:

#### Tipos de datos

* Primitivas: Entero, Numérico, Cadena, Booleano
* Estructurado: Fecha/Hora, Matriz, Rango/Multirango, UUID
* Documento: JSON/JSONB, XML, Clave-valor (Hstore)
* Geometría: Punto, Línea, Círculo, Polígono
* Personalizaciones: Compuesto, Tipos personalizados

#### Integridad de los datos

* ÚNICO, NO NULO
* Claves primarias
* Claves foráneas
* Restricciones de exclusión
* Bloqueos explícitos, bloqueos de asesoramiento

#### Simultaneidad, rendimiento

* Indexación: Árbol B, Multicolumna, Expresiones, Parcial
* Indexación avanzada: GiST, SP-Gist, KNN Gist, GIN, BRIN, índices de cobertura, filtros de Bloom
* Planificador/optimizador de consultas sofisticado, análisis de solo índice, estadísticas de varias columnas
* Transacciones, transacciones anidadas (a través de puntos de guardado)
* Control de simultaneidad de varias versiones (MVCC)
* Paralelización de consultas de lectura y creación de índices de árbol B
* Particionamiento de tablas
* Todos los niveles de aislamiento de transacciones definidos en el estándar SQL, incluido Serializable
* Compilación Just-In-Time (JIT) de expresiones

#### Confiabilidad, recuperación ante desastres

* Registro de escritura anticipada (WAL)
* Replicación: asíncrona, síncrona, lógica
* Recuperación a un momento dado (PITR), en espera activa
* Espacios de tablas

#### Seguridad

* Autenticación: GSSAPI, SSPI, LDAP, SCRAM-SHA-256, certificado y más
* Robusto sistema de control de acceso
* Seguridad a nivel de columna y fila
* Autenticación multifactor con certificados y un método adicional

#### Extensibilidad

* Funciones y procedimientos almacenados
* Lenguajes procedimentales: PL/pgSQL, Perl, Python y Tcl. Hay otros lenguajes disponibles a través de extensiones, por ejemplo, Java, JavaScript (V8), R, Lua y Rust
* Constructores SQL/JSON y expresiones de ruta de acceso
* Contenedores de datos externos: conéctese a otras bases de datos o flujos con una interfaz SQL estándar
* Interfaz de almacenamiento personalizable para tablas
* Muchas extensiones que proporcionan funcionalidad adicional, incluyendo PostGIS

#### Internacionalización, Búsqueda de texto

* Compatibilidad con conjuntos de caracteres internacionales, por ejemplo, a través de intercalaciones de ICU
* Intercalaciones que no distinguen entre mayúsculas y minúsculas ni acentos
* Búsqueda de texto completo

(PostgreSQL, PostgreSQL, 2024)

## Breve historia de PostgreSQL

El sistema de gestión de bases de datos relacionales de objetos, ahora conocido como PostgreSQL, se deriva del paquete POSTGRES escrito en la Universidad de California en Berkeley. Con décadas de desarrollo a sus espaldas, PostgreSQL es ahora la base de datos de código abierto más avanzada disponible en cualquier lugar (PostgreSQL, postgresql documentacion, 2024).

### El proyecto POSTGRES de Berkeley

El proyecto POSTGRES, dirigido por el profesor Michael Stonebraker, fue patrocinado por la Agencia de Proyectos de Investigación Avanzada de Defensa (DARPA), la Oficina de Investigación del Ejército (ARO), la Fundación Nacional de Ciencias (NSF) y ESL, Inc. La implementación de POSTGRES comenzó en 1986. Los conceptos iniciales del sistema se presentaron en [ston86] y la definición del modelo de datos inicial apareció en [rowe87]. El diseño del sistema de reglas en ese momento fue descrito en [ston87a]. La lógica y la arquitectura del administrador de almacenamiento se detallaron en [ston87b] (PostgreSQL, postgresql documentacion, 2024).

POSTGRES ha sufrido varios lanzamientos importantes desde entonces. El primer sistema "demoware" entró en funcionamiento en 1987 y se mostró en la Conferencia ACM-SIGMOD de 1988. La versión 1, descrita en [ston90a], fue lanzada a unos pocos usuarios externos en junio de 1989. En respuesta a una crítica del primer sistema de reglas ([ston89]), el sistema de reglas fue rediseñado ([ston90b]), y la versión 2 fue lanzada en junio de 1990 con el nuevo sistema de reglas. La versión 3 apareció en 1991 y agregó soporte para múltiples administradores de almacenamiento, un ejecutor de consultas mejorado y un sistema de reglas reescrito. En su mayor parte, las versiones posteriores hasta Postgres95 (ver más abajo) se centraron en la portabilidad y la confiabilidad (PostgreSQL, postgresql documentacion, 2024).

POSTGRES se ha utilizado para implementar muchas aplicaciones diferentes de investigación y producción. Estos incluyen: un sistema de análisis de datos financieros, un paquete de monitoreo del rendimiento del motor a reacción, una base de datos de seguimiento de asteroides, una base de datos de información médica y varios sistemas de información geográfica. POSTGRES también se ha utilizado como herramienta educativa en varias universidades. Finalmente, Illustra Information Technologies (más tarde fusionada con Informix, que ahora es propiedad de IBM) recogió el código y lo comercializó. A finales de 1992, POSTGRES se convirtió en el principal gestor de datos para el proyecto de computación científica Sequoia 2000 (PostgreSQL, postgresql documentacion, 2024).

El tamaño de la comunidad de usuarios externos casi se duplicó en 1993. Se hizo cada vez más evidente que el mantenimiento del código y el soporte del prototipo estaba ocupando grandes cantidades de tiempo que deberían haberse dedicado a la investigación de bases de datos. En un esfuerzo por reducir esta carga de soporte, el proyecto Berkeley POSTGRES finalizó oficialmente con la versión 4.2. (PostgreSQL, postgresql documentacion, 2024)

### Postgres95

En 1994, Andrew Yu y Jolly Chen añadieron un intérprete de lenguaje SQL a POSTGRES. Bajo un nuevo nombre, Postgres95 fue lanzado posteriormente a la web para encontrar su propio camino en el mundo como un descendiente de código abierto del código original de POSTGRES Berkeley (PostgreSQL, postgresql documentacion, 2024).

El código Postgres95 era completamente ANSI C y se había recortado su tamaño en un 25%. Muchos cambios internos mejoraron el rendimiento y la capacidad de mantenimiento. La versión 1.0.x de Postgres95 se ejecutó entre un 30 y un 50 % más rápido en Wisconsin Benchmark en comparación con POSTGRES, versión 4.2. Aparte de las correcciones de errores, las siguientes fueron las principales mejoras:

* El lenguaje de consulta PostQUEL fue reemplazado por SQL (implementado en el servidor). (La biblioteca de interfaz libpq lleva el nombre de PostQUEL). Las subconsultas no eran compatibles hasta PostgreSQL (ver más abajo), pero podían ser imitadas en Postgres95 con funciones SQL definidas por el usuario. Se volvieron a implementar las funciones agregadas. También se agregó soporte para la cláusula de consulta GROUP BY
* Se proporcionó un nuevo programa (psql) para consultas SQL interactivas, que utilizaba GNU Readline. Esto reemplazó en gran medida al antiguo programa de monitores.
* Una nueva biblioteca front-end, , admite clientes basados en Tcl. Un shell de ejemplo, , proporcionó nuevos comandos Tcl para interconectar los programas Tcl con el servidor Postgres95.libpgtclpgtclsh
* Se revisó la interfaz de objetos grandes. La inversión de objetos grandes era el único mecanismo para almacenar objetos grandes. (Se ha eliminado el sistema de archivos de inversión).
* Se ha eliminado el sistema de reglas de nivel de instancia. Las reglas seguían estando disponibles como reglas de reescritura.
* Con el código fuente se distribuyó un breve tutorial que introducía las características regulares de SQL, así como las de Postgres95
* Se utilizó GNU make (en lugar de BSD make) para la compilación. Además, Postgres95 se podía compilar con un GCC sin parches (se corrigió la alineación de datos de los dobles).

(PostgreSQL, postgresql documentacion, 2024)

### PostgreSQL

En 1996, quedó claro que el nombre "Postgres95" no resistiría la prueba del tiempo. Elegimos un nuevo nombre, PostgreSQL, para reflejar la relación entre el POSTGRES original y las versiones más recientes con capacidad SQL. Al mismo tiempo, establecimos la numeración de la versión para que comience en 6.0, volviendo a colocar los números en la secuencia originalmente iniciada por el proyecto POSTGRES de Berkeley.

Mucha gente sigue refiriéndose a PostgreSQL como "Postgres" (ahora rara vez en mayúsculas) debido a la tradición o porque es más fácil de pronunciar. Este uso es ampliamente aceptado como apodo o alias.

Durante el desarrollo de Postgres95, el énfasis estuvo en identificar y comprender los problemas existentes en el código del servidor. Con PostgreSQL, el énfasis se ha desplazado hacia el aumento de características y capacidades, aunque el trabajo continúa en todas las áreas.

Los detalles sobre lo que ha sucedido en PostgreSQL desde entonces se pueden encontrar en el Apéndice E.

## Documentación

### Tipos de datos en PostgreSQL

Tabla 1. Tipos de datos de PostgreSQL (PostgreSQL, PostgreSQL, 2024)

| **Nombre** | **Alias** | **Descripción** |
| --- | --- | --- |
| **bigint** | **int8** | **entero de ocho bytes con signo** |
| **bigserial** | **serial8** | **entero de ocho bytes autoincremental** |
| **bit [ (*n*) ]** |  | **cadena de bits de longitud fija** |
| **bit varying [ (*n*) ]** | **varbit** | **cadena de bits de longitud variable** |
| **boolean** | **bool** | **Booleano lógico (verdadero/falso)** |
| **box** |  | **caja rectangular en el plano** |
| **bytea** |  | **datos binarios ("matriz de bytes")** |
| **character varying [ (*n*) ]** | **varchar [ (*n*) ]** | **cadena de caracteres de longitud variable** |
| **character [ (*n*) ]** | **char [ (*n*) ]** | **cadena de caracteres de longitud fija** |
| **cidr** |  | **Dirección de red IPv4 o IPv6** |
| **circle** |  | **círculo en el plano** |
| **date** |  | **fecha del calendario (año, mes, día)** |
| **double precision** | **float8** | **número de punto flotante de doble precisión** |
| **inet** |  | **Dirección de host IPv4 o IPv6** |
| **integer** | **int, int4** | **entero de cuatro bytes con signo** |
| **interval [ (*p*) ]** |  | **periodo de tiempo** |
| **line** |  | **línea infinita en el plano** |
| **lseg** |  | **segmento de recta en el plano** |
| **macaddr** |  | **Dirección MAC** |
| **money** |  | **monto de dinero** |
| **numeric [ (*p*, *s*) ]** | **decimal [ (*p*, *s*) ]** | **número exacto de precisión seleccionable** |
| **path** |  | **trayectoria geométrica en el plano** |
| **point** |  | **punto geométrico en el plano** |
| **polygon** |  | **trayectoria geométrica cerrada en el plano** |
| **real** | **float4** | **número de punto flotante de precisión simple** |
| **smallint** | **int2** | **entero de dos bytes con signo** |
| **serial** | **serial4** | **entero de cuatro bytes autoincremental** |
| **text** |  | **cadena de caracteres de longitud variable** |
| **time [ (*p*) ] [ without time zone ]** |  | **hora del día** |
| **time [ (*p*) ] with time zone** | **timetz** | **hora del día, incluyendo zona horaria** |
| **timestamp [ (*p*) ] [ without time zone ]** |  | **fecha y hora** |
| **timestamp [ (*p*) ] with time zone** | **timestamptz** | **fecha y hora, incluyendo zona horaria** |

### Creación de una base de datos

La primera prueba para ver si puede acceder al servidor de bases de datos es intentar crear una base de datos. Un servidor PostgreSQL en ejecución puede administrar muchas bases de datos. Normalmente, se utiliza una base de datos independiente para cada proyecto o para cada usuario.

Para crear una nueva base de datos, en este ejemplo denominada mydb, utilice el siguiente comando:

$ createdb mydb

También puede crear bases de datos con otros nombres. PostgreSQL le permite crear cualquier número de bases de datos en un sitio determinado. Los nombres de las bases de datos deben tener un primer carácter alfabético y están limitados a 63 bytes de longitud. Una opción conveniente es crear una base de datos con el mismo nombre que su nombre de usuario actual. Muchas herramientas asumen que el nombre de la base de datos es el predeterminado, por lo que puede ahorrarle algo de escritura. Para crear esa base de datos, simplemente escriba:

$ createdb

Si ya no desea utilizar su base de datos, puede eliminarla. Por ejemplo, si usted es el propietario (creador) de la base de datos, puede destruirla mediante el siguiente comando:

$ dropdb mydb

(Para este comando, el nombre de la base de datos no se establece de forma predeterminada en el nombre de la cuenta de usuario. Siempre es necesario especificarlo). Esta acción elimina físicamente todos los archivos asociados con la base de datos y no se puede deshacer, por lo que esto solo debe hacerse con mucha previsión (PostgreSQL, postgresql documentacion, 2024).

### Creación de una nueva tabla

Puede crear una nueva tabla especificando el nombre de la tabla, junto con todos los nombres de columna y sus tipos:

CREATE TABLE weather (

city varchar(80),

temp\_lo int, -- low temperature

temp\_hi int, -- high temperature

prcp real, -- precipitation

date date

);

Por último, cabe mencionar que si ya no necesitas una tabla o quieres volver a crearla de forma diferente puedes eliminarla utilizando el siguiente comando:

DROP TABLE tablename;

(PostgreSQL, postgresql documentacion, 2024)

### Rellenar una tabla con filas

La instrucción INSERT se utiliza para rellenar una tabla con filas:

INSERT INTO weather VALUES ('San Francisco', 46, 50, 0.25, '1994-11-27');

Tenga en cuenta que todos los tipos de datos utilizan formatos de entrada bastante obvios. Las constantes que no son valores numéricos simples normalmente deben ir entre comillas simples (‘’), como en el ejemplo.

El tipo point requiere un par de coordenadas como entrada, como se muestra aquí:

INSERT INTO cities VALUES ('San Francisco', '(-194.0, 53.0)');

La sintaxis utilizada hasta ahora requiere que recuerde el orden de las columnas. Una sintaxis alternativa le permite enumerar las columnas explícitamente:

INSERT INTO weather (city, temp\_lo, temp\_hi, prcp, date)

VALUES ('San Francisco', 43, 57, 0.0, '1994-11-29');

Puede enumerar las columnas en un orden diferente si lo desea o incluso omitir algunas columnas, por ejemplo, si se desconoce la precipitación:

INSERT INTO weather (date, city, temp\_hi, temp\_lo)

VALUES ('1994-11-29', 'Hayward', 54, 37);

(PostgreSQL, postgresql documentacion, 2024)

### Consultar una tabla

Para recuperar datos de una tabla, se consulta la tabla. Para ello, se utiliza una sentencia SQL. La instrucción se divide en una lista de selección (la parte que enumera las columnas que se van a devolver), una lista de tablas (la parte que enumera las tablas de las que se van a recuperar los datos) y una calificación opcional (la parte que especifica las restricciones). Por ejemplo, para recuperar todas las filas de la tabla, escriba:

SELECT \* FROM weather;

Aquí hay una abreviatura para "todas las columnas". Por lo tanto, el mismo resultado se obtendría con:

SELECT city, temp\_lo, temp\_hi, prcp, date FROM weather;

Puede escribir expresiones, no solo referencias de columna simples, en la lista de selección. Por ejemplo, puedes hacer lo siguiente:

SELECT city, (temp\_hi+temp\_lo)/2 AS temp\_avg, date FROM weather;

Otra forma de hacer las consultas es la siguiente:

SELECT DISTINCT city

FROM weather

ORDER BY city;

(PostgreSQL, postgresql documentacion, 2024)

### Uniones entre tablas

Hasta ahora, nuestras consultas solo han accedido a una tabla a la vez. Las consultas pueden tener acceso a varias tablas a la vez o acceder a la misma tabla de tal manera que se procesen varias filas de la tabla al mismo tiempo. Las consultas que tienen acceso a varias tablas (o a varias instancias de la misma tabla) a la vez se denominan consultas de combinación. Combinan filas de una tabla con filas de una segunda tabla, con una expresión que especifica qué filas se van a emparejar. Por ejemplo, para devolver todos los registros meteorológicos junto con la ubicación de la ciudad asociada, la base de datos debe comparar la columna de cada fila de la tabla con la columna de todas las filas de la tabla y seleccionar los pares de filas en los que coinciden estos valores. cityweathernamecities Esto se lograría mediante la siguiente consulta:

SELECT \* FROM weather JOIN cities ON city = name;

Observe dos cosas sobre el conjunto de resultados:

No hay ninguna fila de resultados para la ciudad de Hayward. Esto se debe a que no hay ninguna entrada coincidente en la tabla de Hayward, por lo que la combinación ignora las filas no coincidentes de la tabla. Veremos en breve cómo se puede arreglar esto.

Hay dos columnas que contienen el nombre de la ciudad. Esto es correcto porque las listas de columnas de las tablas y están concatenadas. Sin embargo, en la práctica, esto no es deseable, por lo que probablemente querrá enumerar las columnas de salida explícitamente en lugar de usar:

SELECT city, temp\_lo, temp\_hi, prcp, date, location

FROM weather JOIN cities ON city = name;

Dado que todas las columnas tenían nombres diferentes, el analizador encontró automáticamente a qué tabla pertenecen. Si hubiera nombres de columna duplicados en las dos tablas, tendría que calificar los nombres de columna para mostrar a cuál se refería, como en:

SELECT weather.city, weather.temp\_lo, weather.temp\_hi,

weather.prcp, weather.date, cities.location

FROM weather JOIN cities ON weather.city = cities.name;

En general, se considera un buen estilo calificar todos los nombres de columna de una consulta de combinación, de modo que no se produzca un error en la consulta si posteriormente se agrega un nombre de columna duplicado a una de las tablas.

Las consultas de combinación del tipo visto hasta ahora también se pueden escribir de esta forma:

SELECT \*

FROM weather, cities

WHERE city = name;

Esta sintaxis es anterior a la sintaxis /, que se introdujo en SQL-92. Las tablas simplemente se enumeran en la cláusula y la expresión de comparación se agrega a la cláusula. Los resultados de esta sintaxis implícita anterior y la sintaxis explícita / más reciente son idénticos. Pero para un lector de la consulta, la sintaxis explícita hace que su significado sea más fácil de entender: la condición de combinación se introduce por su propia palabra clave, mientras que anteriormente la condición se mezclaba en la cláusula junto con otras condiciones.

Ahora vamos a ver cómo podemos recuperar los registros de Hayward. Lo que queremos que haga la consulta es escanear la tabla y que cada fila encuentre la(s) fila(s) coincidente(s). Si no se encuentra ninguna fila coincidente, queremos que se sustituyan algunas columnas de la tabla por algunos "valores vacíos". Este tipo de consulta se denomina combinación externa. (Las uniones que hemos visto hasta ahora son uniones internas). El comando tiene el siguiente aspecto:

SELECT \*

FROM weather LEFT OUTER JOIN cities ON weather.city = cities.name;

(PostgreSQL, postgresql documentacion, 2024)

### Funciones agregadas

Al igual que la mayoría de los otros productos de bases de datos relacionales, PostgreSQL admite funciones agregadas. Una función de agregado calcula un único resultado a partir de varias filas de entrada. Por ejemplo, hay agregados para calcular , , (promedio), (máximo) y (mínimo) en un conjunto de filas.

A modo de ejemplo, podemos encontrar la lectura de baja temperatura más alta en cualquier lugar con:

SELECT max(temp\_lo) FROM weather;

Si quisiéramos saber en qué ciudad (o ciudades) se produjo esa lectura, podríamos intentar:

SELECT city FROM weather WHERE temp\_lo = max(temp\_lo);

Pero esto no funcionará, ya que el agregado no se puede usar en la cláusula. (Esta restricción existe porque la cláusula determina qué filas se incluirán en el cálculo agregado; por lo tanto, obviamente tiene que evaluarse antes de que se calculen las funciones agregadas). Sin embargo, como suele ser el caso, la consulta se puede reformular para lograr el resultado deseado, aquí mediante una subconsulta:

SELECT city FROM weather

WHERE temp\_lo = (SELECT max(temp\_lo) FROM weather);

city

Esto es correcto porque la subconsulta es un cálculo independiente que calcula su propio agregado por separado de lo que está sucediendo en la consulta externa.

Los agregados también son muy útiles en combinación con cláusulas. Por ejemplo, podemos obtener el número de lecturas y la temperatura mínima máxima observada en cada ciudad con:

SELECT city, count(\*), max(temp\_lo)

FROM weather

GROUP BY city;

Lo que nos da una fila de salida por ciudad. Cada resultado agregado se calcula sobre las filas de la tabla que coinciden con esa ciudad. Podemos filtrar estas filas agrupadas usando:

SELECT city, count(\*), max(temp\_lo)

FROM weather

GROUP BY city

HAVING max(temp\_lo) < 40;

lo que nos da los mismos resultados solo para las ciudades que tienen todos los valores por debajo de 40. Finalmente, si solo nos preocupamos por las ciudades cuyos nombres comienzan con "S", podríamos hacer:

SELECT city, count(\*), max(temp\_lo)

FROM weather

WHERE city LIKE 'S%' -- (1)

GROUP BY city;

Es importante comprender la interacción entre los agregados y las cláusulas y SQL. La diferencia fundamental entre y es la siguiente: selecciona las filas de entrada antes de que se calculen los grupos y los agregados (por lo tanto, controla qué filas entran en el cálculo del agregado), mientras que selecciona las filas de grupo después de que se calculan los grupos y los agregados. Por lo tanto, la cláusula no debe contener funciones agregadas; No tiene sentido intentar usar un agregado para determinar qué filas serán entradas para los agregados. Por otro lado, la cláusula siempre contiene funciones agregadas. (Estrictamente hablando, se le permite escribir una cláusula que no use agregados, pero rara vez es útil. La misma condición podría usarse de manera más eficiente en la etapa).

En el ejemplo anterior, podemos aplicar la restricción del nombre de la ciudad en , ya que no necesita ningún agregado. Esto es más eficaz que agregar la restricción a , porque evitamos realizar los cálculos de agrupación y agregación para todas las filas que no superan la comprobación.

Otra forma de seleccionar las filas que entran en un cálculo agregado es usar , que es una opción por agregado:

SELECT city, count(\*) FILTER (WHERE temp\_lo < 45), max(temp\_lo)

FROM weather

GROUP BY city;

FILTER es muy similar a , excepto que elimina filas solo de la entrada de la función de agregado particular a la que está asociada. Aquí, el agregado solo cuenta las filas con menos de 45; Pero el agregado se sigue aplicando a todas las filas, por lo que sigue encontrando la lectura de 46 (PostgreSQL, postgresql documentacion, 2024).

### Actualizaciones

Puede actualizar las filas existentes mediante el comando. Supongamos que descubres que las lecturas de temperatura están desviadas en 2 grados después del 28 de noviembre. Puede corregir los datos de la siguiente manera:

UPDATE weather

SET temp\_hi = temp\_hi - 2, temp\_lo = temp\_lo - 2

WHERE date > '1994-11-28';

(PostgreSQL, postgresql documentacion, 2024)

### Eliminaciones

Las filas se pueden eliminar de una tabla mediante el comando. Supongamos que ya no está interesado en el clima de Hayward. A continuación, puede hacer lo siguiente para eliminar esas filas de la tabla:

DELETE FROM weather WHERE city = 'Hayward';

Se eliminan todos los registros meteorológicos que pertenecen a Hayward.

SELECT \* FROM weather;

Hay que tener cuidado con las declaraciones de la forma

DELETE FROM tablename;

Sin una calificación, eliminará todas las filas de la tabla dada, dejándola vacía. ¡El sistema no solicitará confirmación antes de hacer esto!

(PostgreSQL, postgresql documentacion, 2024)

### Vistas

Supongamos que la lista combinada de registros meteorológicos y la ubicación de la ciudad es de particular interés para la aplicación, pero no desea escribir la consulta cada vez que la necesite. Puede crear una vista sobre la consulta, que le da un nombre a la consulta al que puede hacer referencia como una tabla normal:

CREATE VIEW myview AS

SELECT name, temp\_lo, temp\_hi, prcp, date, location

FROM weather, cities

WHERE city = name;

SELECT \* FROM myview;

Hacer un uso liberal de las vistas es un aspecto clave de un buen diseño de base de datos SQL. Las vistas permiten encapsular los detalles de la estructura de las tablas, que pueden cambiar a medida que evoluciona la aplicación, detrás de interfaces coherentes.

Las vistas se pueden usar en casi cualquier lugar donde se pueda usar una mesa real. No es raro construir puntos de vista sobre otros puntos de vista (PostgreSQL, postgresql documentacion, 2024).

### Claves foráneas

Considere el siguiente problema: desea asegurarse de que nadie pueda insertar filas en la tabla que no tengan una entrada coincidente en la tabla. A esto se le llama mantener la integridad referencial de sus datos. En los sistemas de bases de datos simplistas, esto se implementaría (si es que se implementa) mirando primero la tabla para verificar si existe un registro coincidente y luego insertando o rechazando los nuevos registros. Este enfoque tiene una serie de problemas y es muy inconveniente, por lo que PostgreSQL puede hacer esto por usted.

La nueva declaración de las tablas quedaría así:

CREATE TABLE cities (

name varchar(80) primary key,

location point

);

CREATE TABLE weather (

city varchar(80) references cities(name),

temp\_lo int,

temp\_hi int,

prcp real,

date date

);

Ahora intente insertar un registro no válido:

INSERT INTO weather VALUES ('Berkeley', 45, 53, 0.0, '1994-11-28');

ERROR: insert or update on table "weather" violates foreign key constraint "weather\_city\_fkey"

DETAIL: Key (city)=(Berkeley) is not present in table "cities".

(PostgreSQL, postgresql documentacion, 2024)

## Ranking

Ilustración 1. Ranking de motores de bases de datos

Para los desarrolladores profesionales, PostgreSQL apenas le arrebató el primer lugar a MySQL. Los desarrolladores profesionales son más propensos que aquellos aprender a codificar para usar Redis, PostgreSQL, Microsoft SQL Server y Elasticsearch. MongoDB es utilizado por un porcentaje similar tanto de desarrolladores profesionales como de aquellos que están aprendiendo a programar, y es la segunda base de datos más popular para los que aprenden a programar (detrás de MySQL). Esto tiene sentido, ya que es compatible con una gran cantidad de lenguajes y plataformas de desarrollo de aplicaciones.

Ilustración 2.Ranking de motores de bases de datos

## pgAdmin 4

pgAdmin es la herramienta de gestión de código abierto líder para Postgres, la base de datos de código abierto más avanzada del mundo. pgAdmin 4 está diseñado para satisfacer las necesidades tanto de los usuarios novatos como de los experimentados de Postgres, proporcionando Una potente interfaz gráfica que simplifica la creación, el mantenimiento y el uso de objetos de base de datos (Team, 2024).

Los paquetes de instalación precompilados y configurados para pgAdmin 4 están disponibles para una serie de entornos de escritorio; Recomendamos utilizar un instalador siempre que sea posible (Team, 2024).

En una implementación de servidor, la aplicación pgAdmin se implementa detrás de un servidor web o con la interfaz WSGI. Si instala pgAdmin en modo servidor, se le pedirá que proporcione un rol name y la contraseña de pgAdmin cuando se conecta inicialmente a pgAdmin. La primera el rol registrado en pgAdmin será un usuario administrativo; el administrative puede utilizar el cuadro de diálogo de gestión de usuarios pgAdmin para crear y administrar cuentas de usuario pgAdmin adicionales. Cuando un usuario se autentica con pgAdmin, el control de árbol pgAdmin muestra las definiciones de servidor asociado a ese rol de inicio de sesión (Team, 2024).

En una implementación de escritorio, la aplicación pgAdmin está configurada para utilizar la función Entorno de ejecución de escritorio para alojar el programa en una plataforma compatible. Normalmente, los usuarios instalarán un paquete precompilado para ejecutar pgAdmin en el escritorio modo, pero se puede instalar una implementación de escritorio manual y, aunque es más difícil de configurar, puede ser útil para los desarrolladores interesados en comprender cómo funciona pgAdmin (Team, 2024).

También es posible utilizar un Container Deployment de pgAdmin, en el que el servidor El modo está preconfigurado para la seguridad (Team, 2024).

# Modelado de Caso FP

## Instrucciones

Diseñe un modelo de base de datos normalizado utilizando el modelo Entidad/Relación a partir de la información que se le da a continuación:

“Se desea crear un sistema de información para registrar la programación mensual del canal de TV por cable CINEX con el propósito de publicar la programación en Internet.

Un ejemplo de listado de programación que podrá ser enviado por correo electrónico cuando el usuario solicite que se le envíe por correo electrónico la programación mensual es el siguiente:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Lunes 1 de Marzo de 2003** | | |  | |  |  |
| **Hora** | **Película** | **Descripción** | **Duración** | **Género** | **Clasificación** | **Director** |
| 06:00 | Big | Cuando un niño pide un deseo a una máquina de juego… | 2:00 | Comedia | PG | Penny  Marshall |
| 08:00 | The Fast and the  Furious | Un policía encubierto se  introduce en… | 1:45 | Acción | PG-13 | Rob Cohen |
| 10:00 | When Harry met Sally | Una pareja de amigos descubre… | 1:55 | Comedia | PG-13 | Rob Reiner |
| … | … |  | … | … | … | … |
| **Martes 2 de Marzo** | | |  | |  |  |
| **Hora** | **Película** | **Descripción** | **Duración** | **Género** | **Clasificación** | **Director** |
| 06:00 | The  Princess  Bride | Cuando su  nieto permanece en cama | 1:40 | Comedia | PG | Rob Reiner |
|  |  | enfermo, un abuelo… |  |  |  |  |
| 08:00 | Jerry  Maguire | Un agente  deportivo descubre súbitamente que… | 2:05 | Drama | PG-13 | Cameron Crowe |
| 10:15 | Moulin Rouge! | Cuando  Christian, un escritor sin un centavo en la bolsa… | 2:00 | Musical | PG-13 | Baz  Luhrman |
| 12:30 | The Fast and the  Furious | Un policía encubierto se  introduce en… | 1:45 | Acción | PG-13 | Rob Cohen |
| 2:30 | The  Matrix  Reloaded | La última  ciudad humana,  Zion, se ve en peligro ante el inminente ataque… | 2:20 | Sci-Fi | R | Andy  Wachowski  Larry  Wachowski |

Atendiendo las sugerencias de los televidentes se colocará en el sitio de Internet una consulta que permita consultar cuales días y a cuáles horas se transmitirá una determinada película. Por ejemplo:

|  |  |  |
| --- | --- | --- |
| **Película**: | **The Fast and the Furious** | |
| Día de Transmisión | | Hora de Transmisión |
| Lunes 1 de marzo | | 08:00 |
| Martes 2 de marzo | | 12:30 |
| Sábado 6 de marzo | | 13:00 |
| Domingo 14 de marzo | | 17:15 |

Otra clasificación que se da a las películas es ubicarlas en un espacio para que el público pueda identificar las películas de su interés. Cuando las películas se presentan dentro de este espacio, suelen ser acompañadas de documentales cortos y entrevistas con los artistas.

Algunos de los espacios especiales que tiene el canal CINEX son:

|  |  |  |
| --- | --- | --- |
| **Nombre** | **Descripción** | **Horario** |
| Adrenalina al máximo | Espacio reservado para las películas de acción y suspenso al límite | Todos los sábados a las 13:00 |
| Ganadoras de  Oscar | Este espacio está reservado exclusivamente para las películas que han logrado el máximo  galardón de la Industria | Todos los domingos a las 15:00 |
| CINEX  Independiente | En este espacio se presentan películas innovadoras de cineastas que trabajan fuera del ambiente de Hollywood | Todos los jueves a las 20:00 |

Por ejemplo, las películas programadas para el espacio CINEX Independiente para el mes de Marzo son las siguientes:

|  |  |  |  |
| --- | --- | --- | --- |
| **Espacio:** |  | **CINEX independiente** | |
| **Fecha** | **Película** | **Director** | **Descripción** |
| 4 de marzo 2004 | Memento | Christopher Notan | Leonard Shelby is a man on a mission to find and kill the man who raped and murdered his wife. He's also a man with a problem… |
| 11 de marzo 2004 | Donnie Darko | Richard Kelly | Un joven estudiante, el mejor de su clase, comienza a tener extrañas visiones… |
| 18 de marzo 2004 | Requiem for a dream | Darren  Aronofsky | Cuatro vidas entran en una espiral descendente que parece no tener fin debido a su adicción… |
| 25 de marzo 2004 | Whale Rider | Niki Caro | Cuando en lugar del esperado descendiente varón de una tribu maorí (su futuro líder) llega en la forma de una niña… |

Nótese que los nombres de las películas aparecen subrayados en todos los lugares donde aparece. Esto es así porque cuando el usuario haga clic sobre el nombre de una película apareceré una ficha técnica de la misma conteniendo la siguiente información:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Memento** |  | | |  |
| Director:  Christopher Nolan Productores:  Jennifer Todd, Suzanne  Todd, Aaron Ryder Guionista:  Jonathan Nolan,  Christopher Nolan Música Original: David Julian Editor:  Dody Dorn | **Protagonistas principales:** | | **Personaje** | |
| Guy Pearce | | Leonard Shelby | |
| Carrie Ann-Moss | | Natalie | |
| Joe Pantoliano | | John Gammel | |
| **Otros Datos Generales** | | | |
| Año de Release: | 2000 | | |
| Duración: | 1:50 | | |
| Género: | Suspense | | |
| Clasificación: | R | | |
| Compañía Productora: | Newmarket Films | | | |
| Descripción de la Película: | Leonard Shelby is a man on a mission - to find and kill the man who raped and murdered his wife. He's also a man with a problem - the savage attack also gave him brain damage, and he now suffers from a rare and untreatable form of memory loss. He knows who he is and recalls everything up to the incident, but now he can no longer create new memories, meaning he can't remember people he's met, places he's been or things he's done even fifteen minutes ago. He's | | | |
|  | instinctually learned a system to help him - scribbled notes and photos in his pockets, tattoos all over his body for the really important clues he discovers - as he struggles through his frustrating handicap to find vengeance, always unsure of who to trust. | | | |

Ténganse en cuenta las siguientes condiciones:

Una película puede tener más de un director, más de un productor, más de un guionista, más de un compositor de música original y más de un editor. Además, puede ser producida por más de una compañía productora.

Una misma persona podría estar asociada a la película en más de un trabajo (véase en este ejemplo que Christopher Nolan es a la vez director y Guionista de la película. Incluso los actores pueden desempeñar otros trabajos (como Mel Gibson que fue director y Protagonista a la vez de *Braveheart*).

Los géneros en los que se clasifican las películas están predefinidos ya y son los siguientes:

Drama, Comedia, Romance, Suspense, Terror, Acción, Musical, Sci-Fi, Fantasía e Infantil. 4. Las clasificaciones también están predefinidas y son las siguientes: G, PG, PG-13, R, NC17 y X.

|  |  |  |
| --- | --- | --- |
| **Christopher Nolan** | |  |
| **Datos Generales** | | |
| Fecha de Nacimiento: | 30 de julio 1970 | |
| Nacionalidad: | Británico | |
| **Biografía** | | |
| Noted for the innovative structure of both his noirish, cerebral debut film Following  (1998) and its follow-up, the equally unconventional and heady Memento (2000), London-born filmmaker Christopher Nolan has shown a unique talent for creating involving films containing concepts based on abstract breaks with conventional behavior and idealism. Dubbed meta-noir by critics at a loss for words to describe its psychologically demanding, high-concept yet low-key journey into the mind of a man seeking revenge but lacking the ability to create new memories, Memento became the basis of lively discussion and debate among critics and audiences hungering for something thoughtful among a flurry of countless computer-generated pseudo-thrills and all-toofamiliar gross-out comedies. | | |
| **Filmografía** | | |
| Como Director: | Memento(2000)  Insomnia(2002)  Batman:Intimidation(2005) | |
| Como Guionista: | Memento(2000) Insomnia(2002**)** | |

Nótese que también los artistas asociados a las películas aparecen subrayados. Esto es así porque se requiere que al hacer clic sobre el nombre de un artista aparezca una ficha biográfica con la siguiente información:

Ténganse en cuenta las siguientes condiciones:

Cuando el artista a consultar sea un actor, deberá agregarse a su ficha bibliográfica la dirección web de su Fan-Site oficial y la dirección web de su Sitio oficial personal

Cuando el artista a consultar sea un director o un Actor, deberá registrarse el nombre de su representante (manager).

Interesa tener los datos generales y biografías de todos los tipos de artistas que se registren en la base de datos.

## Pasos para modelado de la base de datos

### Nombrar la base de datos

El nombre de la base de datos fue proporcionado en el nombre del documento de la descripción e instrucciones del caso, el cual es: **CasoFP.**

### Identificar las Entidades por Schema

#### Schema catalogo\_programacion

* **CompaniaProductora**
* **EspacioEspecial**
* **Clasificacion**
* **Genero**
* **Pelicula**
* **Programacion**
* **PeliculaCompaniaProductora**
* **EspacioEspecialPelicula**

#### Schema elenco

* **Profesion**
* **Nacionalidad**
* **Artista**
* **Sitie**
* **TipoCargo**
* **PeliculaArtistaCargo**
* **Personaje**

### Identificar los atributos de las entidades

Tabla 2 Atributos de entidades

|  |  |
| --- | --- |
| **Entidad** | **Atributos** |
| CompaniaProductora | IdCompaniaProductora, NombreCompaniaProductora |
| EspacioEspecial | IdEspacioEspecial, Nombre, Descripcion, Horario |
| Clasificacion | IdClasificacion, NombreClasificacion |
| Genero | IdGenero, NombreGenero |
| Pelicula | IdPelicula, IdGenero, IdClasificacion, NombrePelicula, AnioRelace, Duracion, DescripcionPelicula |
| Programacion | IdProgramacion, IdPelicula, Hora, Fecha |
| PeliculaCompaniaProductora | IdPelicula, IdCompaniaProductora |
| EspacioEspecialPelicula | IdEspacioEspecial, IdPelicula, Fecha |
| Profesion | IdProfesion, NombreProfesion, |
| Nacionalidad | IdNacionalidad, NombreNacionalidad, Pais |
| Artista | IdArtista, IdNacionalidad, Manager, Nombres, Apellidos, Biografia, FechaNacimiento, IdProfesion |
| Sitie | IdArtista, FanSiteURL, PersonalSiteURL |
| TipoCargo | IdTipoCargo, NombreTipoCargo |
| PeliculaArtistaCargo | IdPelicula, IdArtista, IdTipoCargo |
| Personaje | IdPersonaje, IdArtista, IdPelicula, Personaje, PersonajePrincipal |

## Diccionario de datos

### Tabla CompaniaProductora

#### Propósito

Esta tabla llevara los registros de todas las compañías productoras que participaran en la producción de una o más películas, de estas compañías solo se precisa almacenar sus nombres.

#### Vista diseño

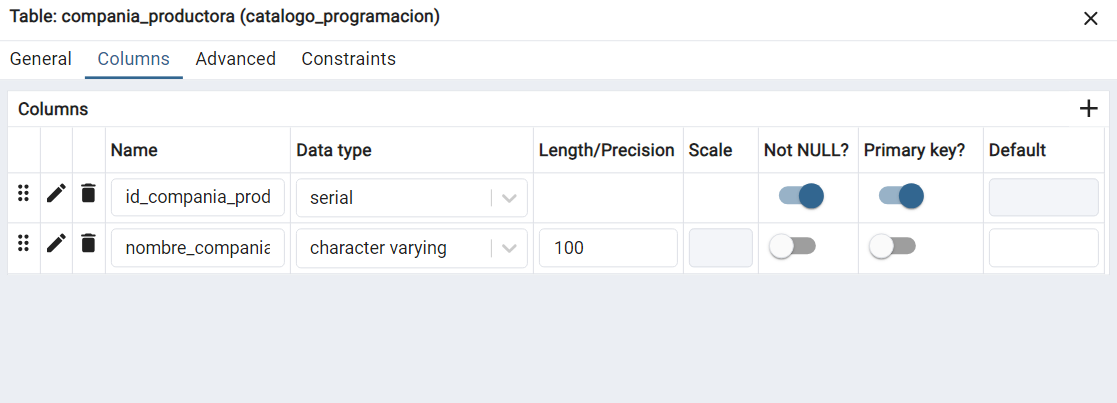


Ilustración 3. vista diseño de la tabla compania\_productora

#### Instrucciones SQL

CREATE TABLE catalogo\_programacion.compania\_productora

(

    id\_compania\_productora SERIAL PRIMARY KEY,

    nombre\_compania\_productora varchar(100) UNIQUE NOT NULL

);

### Tabla EspacioEspecial

#### Propósito

Hay diferentes tipos de espacios especiales y esta tabla se encarga de llevar el registro de estos. Estos espacios tienen un horario que deja especificado el día de la semana y la hora que las películas se van a programar y una descripción del propósito del espacio.

#### Vista diseño

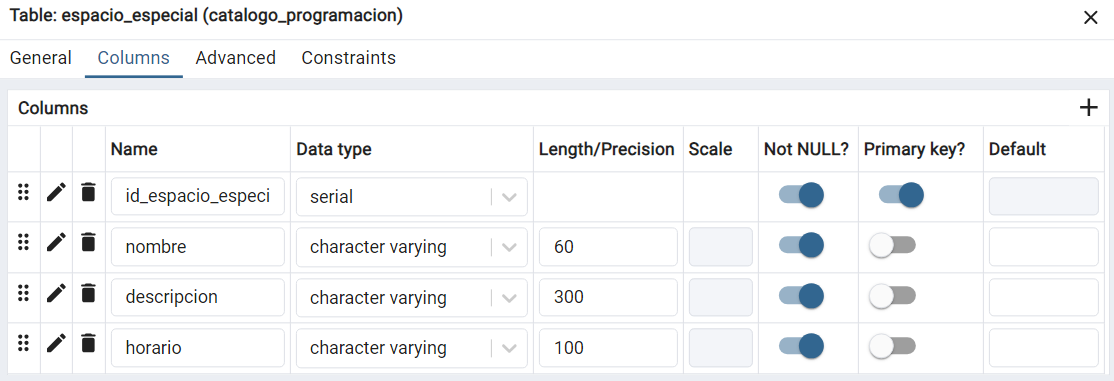


Ilustración 4. Vista diseño de tabla espacio\_especial

#### Instrucciones SQL

CREATE TABLE catalogo\_programacion.espacio\_especial

(

    id\_espacio\_especial SERIAL PRIMARY KEY,

    nombre varchar(60) NOT NULL,

    descripcion varchar(300) NOT NULL,

    horario varchar(100) NOT NULL

);

### Tabla clasificación

#### Propósito

Esta tabla tiene como finalidad normalizar la asignación de una clasificación a cada película que se requiera registrar, para ello solo almacena su propio id y el nombre de dicha clasificación.

#### Vista diseño

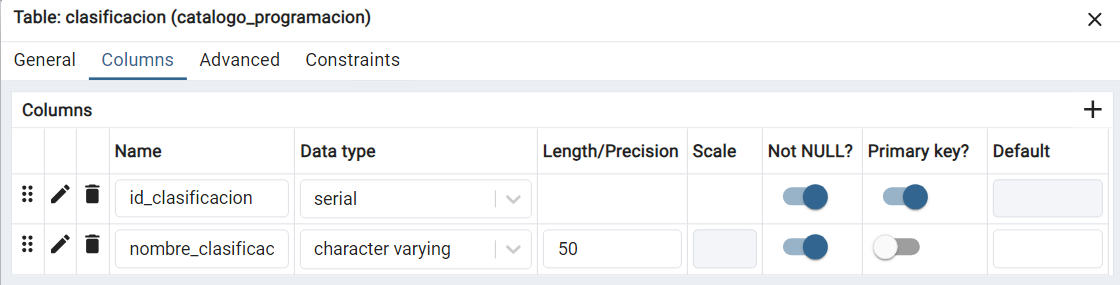


Ilustración 5. Vista diseño de tabla clasificacion

#### Instrucciones SQL

CREATE TABLE catalogo\_programacion.clasificacion

(

    id\_clasificacion SERIAL PRIMARY KEY,

    nombre\_clasificacion varchar(50) NOT NULL UNIQUE

);

### Tabla genero

#### Propósito

Esta tabla tiene como finalidad normalizar la asignación de un género a cada película que se requiera registrar, para ello solo almacena su propio id y el nombre de dicho género.

#### Vista diseño

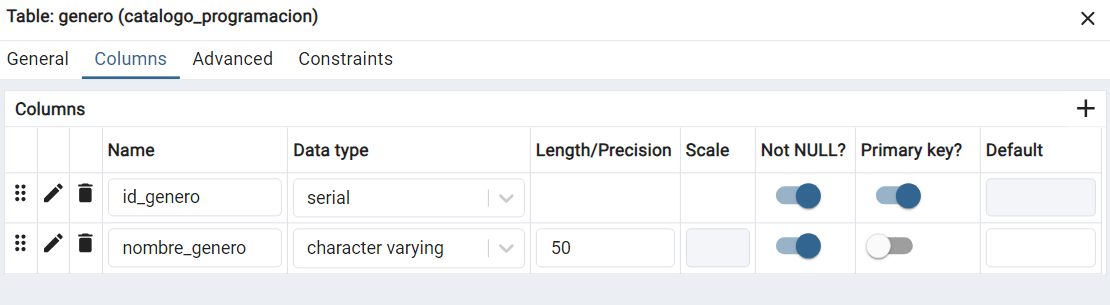


Ilustración 6. Vista diseño de tabla genero

#### Instrucciones SQL

CREATE TABLE catalogo\_programacion.genero

(

    id\_genero SERIAL PRIMARY KEY,

    nombre\_genero varchar(50) UNIQUE NOT NULL

);

### Tabla pelicula

#### Propósito

Esta tabla tiene como objetivo llevar registro de todas las películas que se tengan en catálogo, con interés en sus datos como nombre, género (guardando el id del género registrado en la tabla genero), clasificación (guardando el id de la clasificación registrada en la tabla clasificación), el año en que se estrenó, la duración (en horas: minutos: segundos), y una descripción de la trama como resumen de la película.

#### Vista diseño

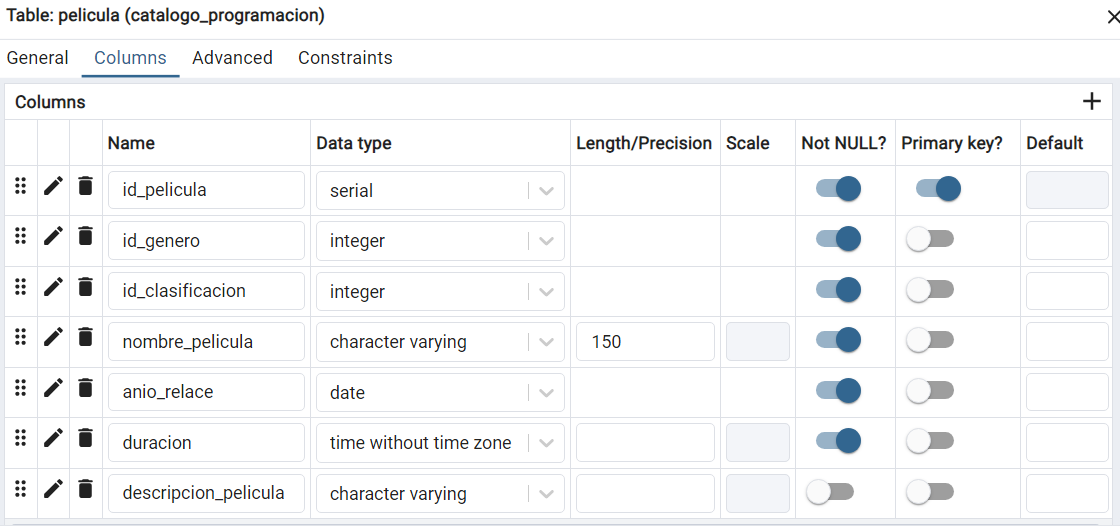


Ilustración 7. Vista diseño de tabla pelicula

#### Instrucciones SQL

CREATE TABLE catalogo\_programacion.pelicula

(

    id\_pelicula SERIAL PRIMARY KEY,

    id\_genero int NOT NULL,

    id\_clasificacion int NOT NULL,

    nombre\_pelicula varchar(150) UNIQUE NOT NULL,

    anio\_relace date NOT NULL,

    duracion time NOT NULL,

    descripcion\_pelicula varchar(300),

    CONSTRAINT fk\_pelicula\_genero FOREIGN KEY (id\_genero)

        REFERENCES catalogo\_programacion.genero (id\_genero),

    CONSTRAINT fk\_pelicula\_clasificacion FOREIGN KEY (id\_clasificacion)

        REFERENCES catalogo\_programacion.clasificacion (id\_clasificacion)

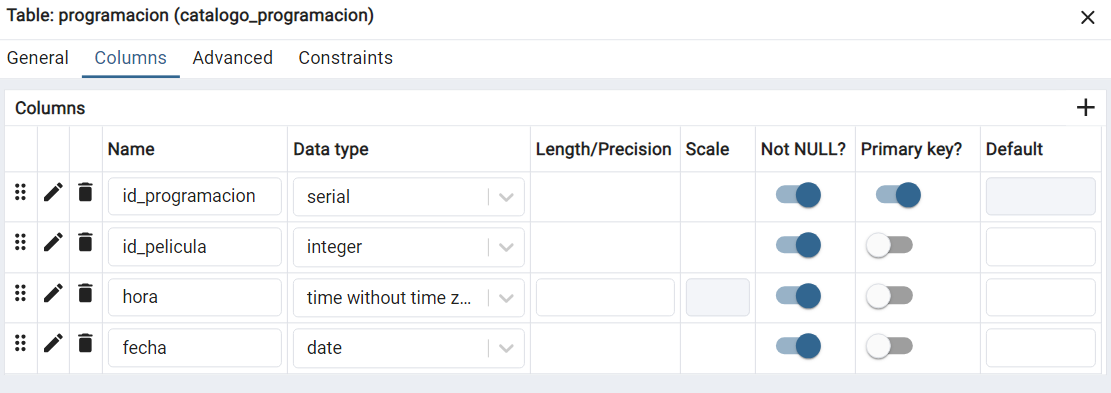
);

### Tabla programación

#### Propósito

Esta tabla lleva la programación de las películas, asignándoles una fecha y una hora determinada. Una misma película puede programarse varias veces para fechas u horas diferentes.

#### Vista diseño



#### Instrucciones SQL

CREATE TABLE catalogo\_programacion.programacion

(

    id\_programacion SERIAL PRIMARY KEY,

    id\_pelicula int NOT NULL,

    hora time NOT NULL,

    fecha date NOT NULL,

    CONSTRAINT fk\_programacion\_pelicula FOREIGN KEY (id\_pelicula)

        REFERENCES catalogo\_programacion.pelicula (id\_pelicula)

);

### Tabla película\_compania\_productora

#### Propósito

Esta es la tabla de asignación de muchos a muchos entre las compañías productoras y las películas.

#### Vista diseño

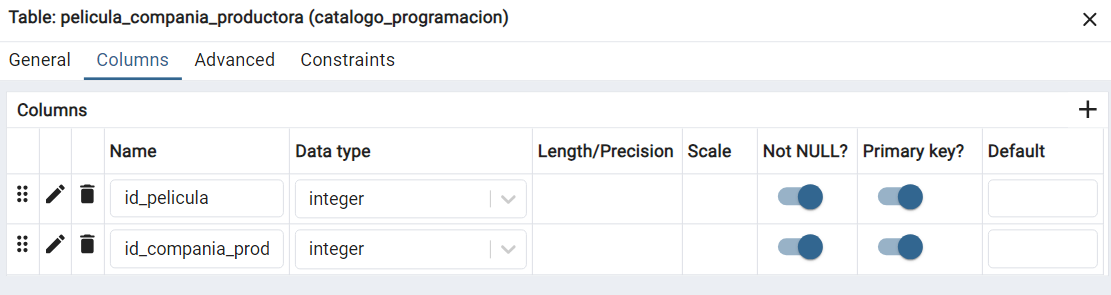


Ilustración 8. Vista diseño de tabla pelicula\_compania\_productora

#### Instrucciones SQL

CREATE TABLE catalogo\_programacion.pelicula\_compania\_productora

(

    id\_pelicula int NOT NULL,

    id\_compania\_productora int NOT NULL,

    CONSTRAINT pk\_pelicula\_compania\_productora PRIMARY KEY (id\_pelicula, id\_compania\_productora),

    CONSTRAINT fk\_pelicula\_compania\_productora\_pelicula FOREIGN KEY (id\_pelicula)

        REFERENCES catalogo\_programacion.pelicula (id\_pelicula),

    CONSTRAINT fk\_pelicula\_compania\_productora\_compania\_productora FOREIGN KEY (id\_compania\_productora)

        REFERENCES catalogo\_programacion.compania\_productora (id\_compania\_productora)

);

### Tabla espacio\_especial\_pelicula

#### Propósito

Esta es la tabla de asignación de muchos a muchos entre la programación de los espacios especiales y las películas. Debemos recordar que las películas se programan en los espacios especiales una fecha determinada basándonos en la descripción del horario que tiene el espacio especial.

#### Vista diseño

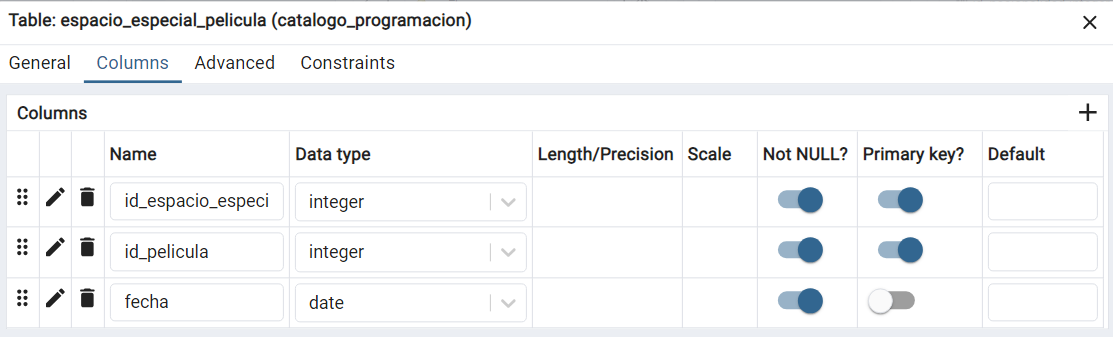


Ilustración 9. Vista diseño de tabla espacio\_especial\_pelicula

#### Instrucciones SQL

CREATE TABLE catalogo\_programacion.espacio\_especial\_pelicula

(

    id\_espacio\_especial int NOT NULL,

    id\_pelicula int NOT NULL,

    fecha date NOT NULL,

    CONSTRAINT pk\_espacio\_especial\_pelicula PRIMARY KEY (id\_espacio\_especial, id\_pelicula),

    CONSTRAINT fk\_espacio\_especial\_pelicula\_espacio\_especial FOREIGN KEY (id\_espacio\_especial)

        REFERENCES catalogo\_programacion.espacio\_especial (id\_espacio\_especial),

    CONSTRAINT fk\_espacio\_especial\_pelicula\_pelicula FOREIGN KEY (id\_pelicula)

        REFERENCES catalogo\_programacion.pelicula (id\_pelicula)

);

### Tabla profesion

#### Propósito

Esta es una tabla catalogo para la normalización de la asignación de una profesión a los artistas.

#### Vista diseño

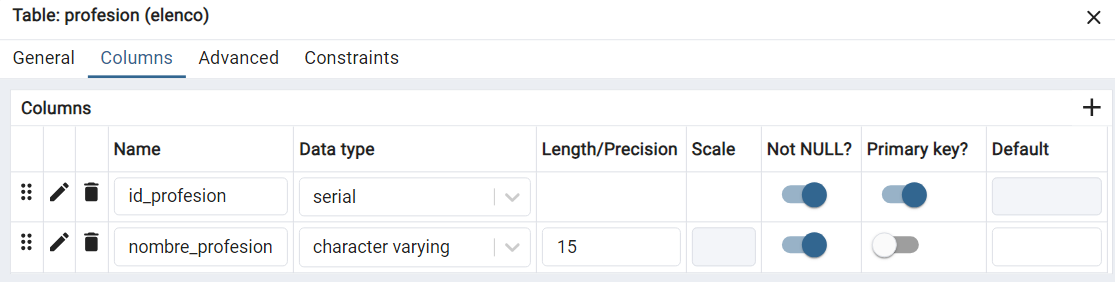


Ilustración 10. Vista diseño de tabla profesion

#### Instrucciones SQL

CREATE TABLE elenco.profesion

(

    id\_profesion SERIAL PRIMARY KEY,

    nombre\_profesion varchar(15) UNIQUE NOT NULL

);

### Tabla nacionalidad

#### Propósito

Esta es una tabla catalogo para la normalización de la asignación de una nacionalidad a los artistas. Almacena el nombre de la nacionalidad y el nombre del país perteneciente.

#### Vista diseño

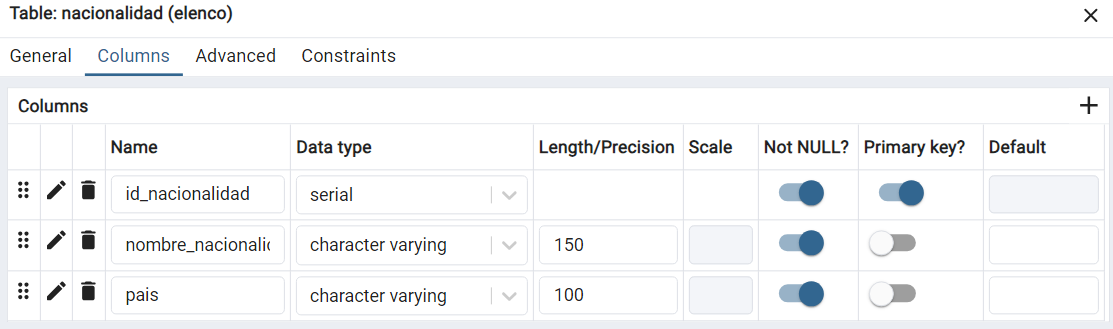


Ilustración 11. Vista diseño de tabla nacionalidad

#### Instrucciones SQL

CREATE TABLE elenco.nacionalidad

(

    id\_nacionalidad SERIAL PRIMARY KEY,

    nombre\_nacionalidad varchar(150) UNIQUE NOT NULL,

    pais varchar (100) UNIQUE NOT NULL

);

### Tabla artista

#### Propósito

Esta tabla tiene como propósito llevar el registro de todos los artistas, ya sean actores, directores, guionistas, productores, editores o compositores, que participen en la producción de una película.

Para esta tabla será necesario el registro de su nacionalidad, manager(únicamente para los actores y directores), nombres, apellidos, biografía, fecha de nacimiento y su profesión (almacenando el id de la profesión que se encuentra en la tabla ya normalizada).

#### Vista diseño

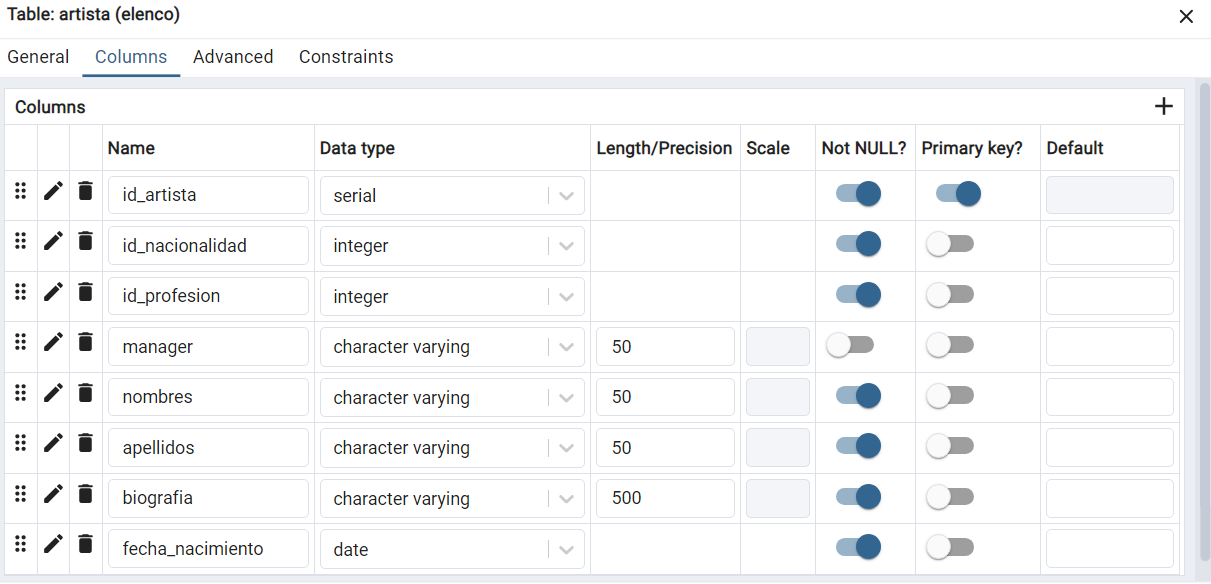


Ilustración 12. Vista diseño de tabla artista

#### Instrucciones SQL

CREATE TABLE elenco.artista

(

    id\_artista SERIAL PRIMARY KEY,

    id\_nacionalidad int NOT NULL,

    id\_profesion int NOT NULL,

    manager varchar(50),

    nombres varchar(50) NOT NULL,

    apellidos varchar(50) NOT NULL,

    biografia varchar(500) NOT NULL,

    fecha\_nacimiento date NOT NULL,

    CONSTRAINT fk\_artista\_nacionalidad FOREIGN KEY (id\_nacionalidad)

        REFERENCES elenco.nacionalidad (id\_nacionalidad),

    CONSTRAINT fk\_artista\_profesion FOREIGN KEY (id\_profesion)

        REFERENCES elenco.profesion (id\_profesion)

);

### Tabla sitie

#### Propósito

En esta tabla se registran las URL del Fansite y el PersonalSite, teniendo como llave primaria un id del artista al que le pertenecen esos sitios asi creando una relación de uno a uno con artista. Pero se debe destacar que solo se puede almacenar los sitios de los actores únicamente.

#### Vista diseño

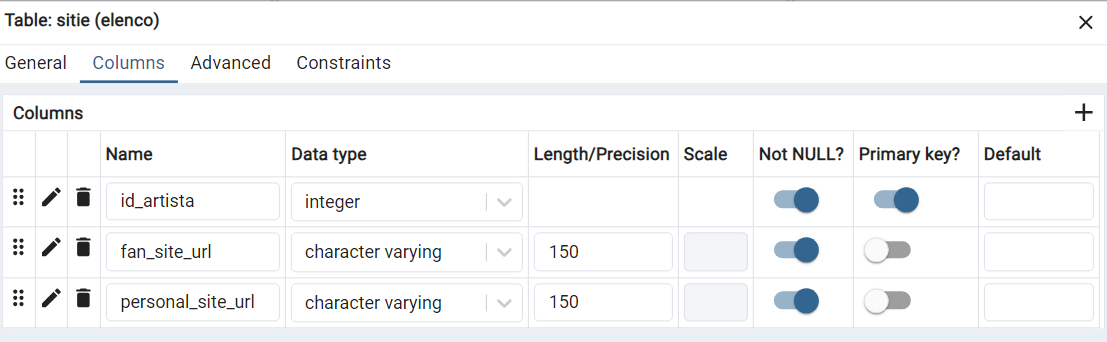


Ilustración 13. Vista diseño de tabla sitie

#### Instrucciones SQL

CREATE TABLE elenco.sitie

(

    id\_artista int NOT NULL,

    fan\_site\_url varchar(150) UNIQUE NOT NULL,

    personal\_site\_url varchar(150) UNIQUE NOT NULL,

    CONSTRAINT pk\_sitie PRIMARY KEY (id\_artista),

    CONSTRAINT fk\_sitie\_artista FOREIGN KEY (id\_artista)

        REFERENCES elenco.artista (id\_artista)

);

### Tabla tipo\_cargo

#### Propósito

Esta tabla sirve para la normalización de la asignación de un cargo que tuvo un artista en la producción de una película.

#### Vista diseño

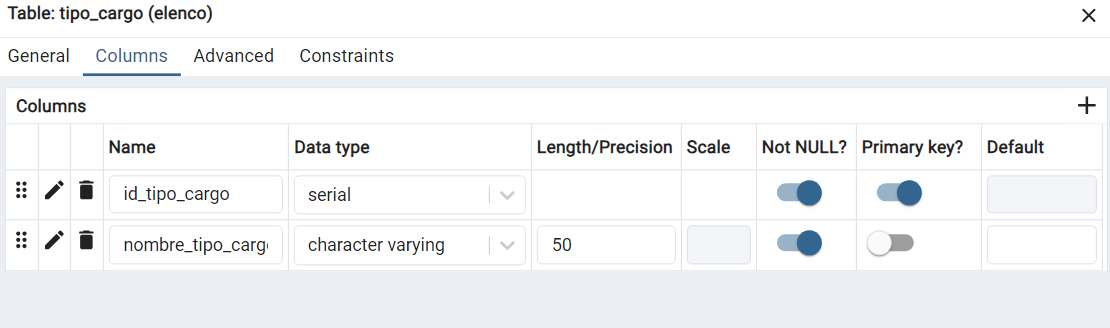


Ilustración 14. Vista diseño de tabla tipo\_cargo

#### Instrucciones SQL

CREATE TABLE elenco.tipo\_cargo

(

    id\_tipo\_cargo SERIAL PRIMARY KEY,

    nombre\_tipo\_cargo varchar(50) UNIQUE NOT NULL

);

### Tabla pelicula\_artista\_cargo

#### Propósito

Esta tabla almacena una triple clave primaria con las llaves foranes de los id de las tablas pelicula, artista y tipo\_cargo. Esto con el propósito de asignarle muchos actores a muchas películas, pero cumpliendo que un actor pueda desempeñar varios cargos para una misma película.

#### Vista diseño

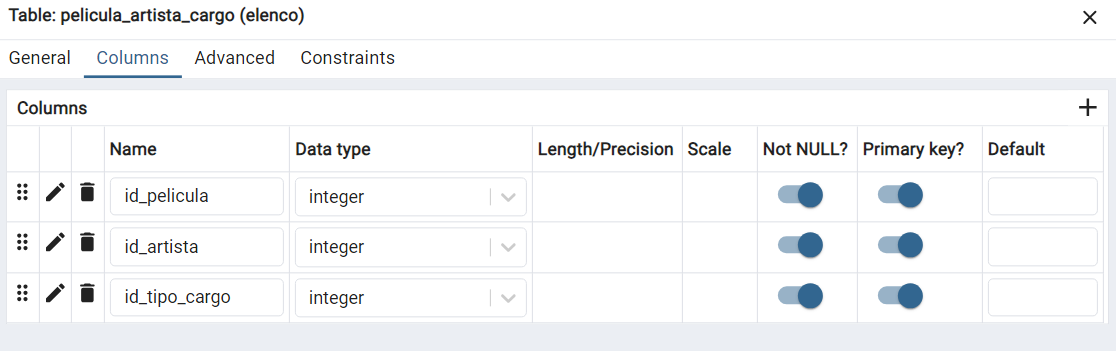


Ilustración 15. Vista diseño de tabla pelicula\_artista\_cargo

#### Instrucciones SQL

CREATE TABLE elenco.pelicula\_artista\_cargo

(

    id\_pelicula int NOT NULL,

    id\_artista int NOT NULL,

    id\_tipo\_cargo int NOT NULL,

    CONSTRAINT pk\_pelicula\_artista\_cargo PRIMARY KEY (id\_pelicula, id\_artista, id\_tipo\_cargo),

    CONSTRAINT fk\_pelicula\_artista\_cargo\_pelicula FOREIGN KEY (id\_pelicula)

        REFERENCES catalogo\_programacion.pelicula (id\_pelicula),

    CONSTRAINT fk\_pelicula\_artista\_cargo\_artista FOREIGN KEY (id\_artista)

        REFERENCES elenco.artista (id\_artista),

    CONSTRAINT fk\_pelicula\_artista\_cargo\_tipo\_cargo FOREIGN KEY (id\_tipo\_cargo)

        REFERENCES elenco.tipo\_cargo (id\_tipo\_cargo)

);

### Tabla personaje

#### Propósito

Esta tabla almacena todos los personajes que han interpretado los artistas en una película (debemos recordar que no solo los artistas cuya profesión sea ser actor pueden llegar a actuar) que hayan actuado como actores. Se necesita almacenar su propio id, dos claves foráneas tanto artista como de la película, también el nombre del personaje que interpreto, y un atributo de tipo booleano para identificar los personajes principales o secundarios.

También debemos recordar que un actor puede interpretar varios personajes en una misma película.

#### Vista diseño

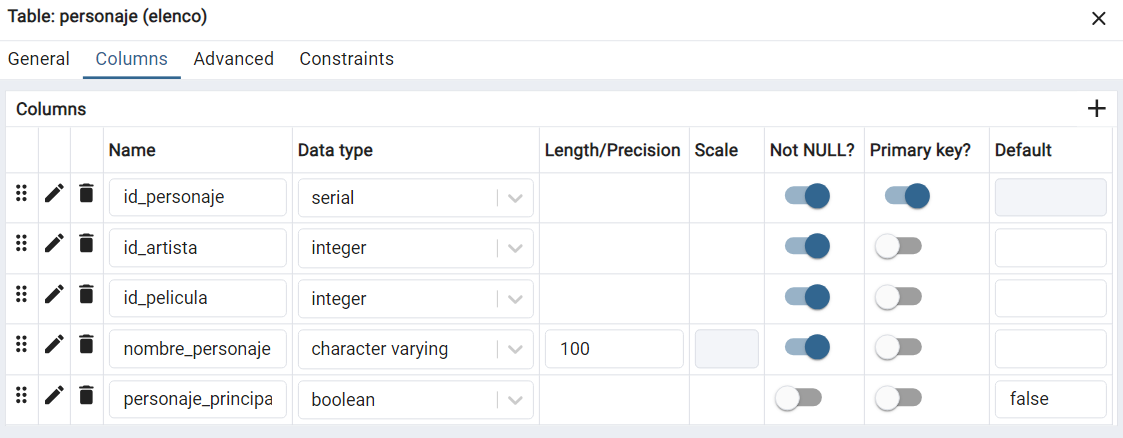


Ilustración 16. Vista diseño de tabla personaje

#### Instrucciones SQL

CREATE TABLE elenco.personaje

(

    id\_personaje SERIAL PRIMARY KEY,

    id\_artista int NOT NULL,

    id\_pelicula int NOT NULL,

    nombre\_personaje VARCHAR(100) NOT NULL,

    personaje\_principal BOOLEAN DEFAULT(FALSE),

    CONSTRAINT fk\_personaje\_artista FOREIGN KEY (id\_artista)

        REFERENCES elenco.artista (id\_artista),

    CONSTRAINT fk\_personaje\_pelicula FOREIGN KEY (id\_pelicula)

        REFERENCES catalogo\_programacion.pelicula (id\_pelicula)

);

## Implementar el Modelo Conceptual en PostgreSQL

### Crear Base de datos

Entramos a pgAdmin 4 iniciamos sesión en el servidor de PostgreSQL que vayamos a crear la base de datos, en la carpeta de **DATABASE** como se puede ver en la ilustración 17 damos click derecho sobre la carpeta, vamos a la opción **Create**, y damos click izquierdo sobre la opción **database** tal y como se puede ver en la Ilustración 17.

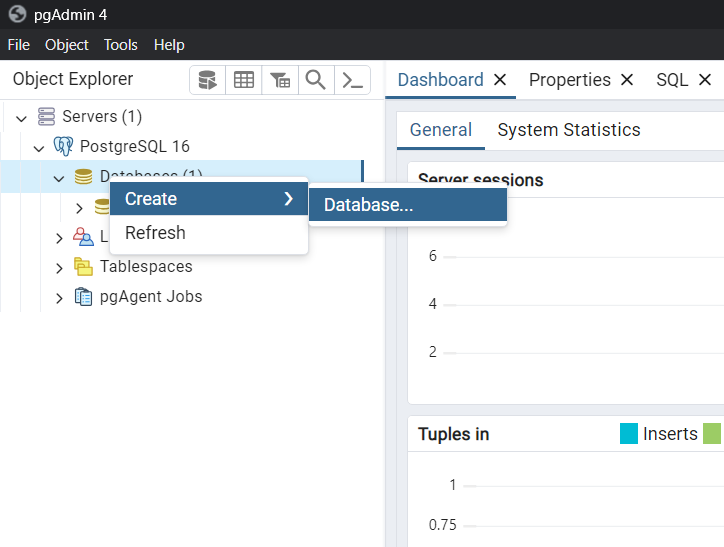


Ilustración 17. Paso 1 para crear base de datos

Una vez hecho esto se nos abrirá una nueva ventana en la cual tendremos que darle un nombre a nuestra base de datos en el cuadro de texto al lado de **Database** y que en este caso nuestro nombre de base de datos es **Caso\_FP**, después de esto debemos dar click izquierdo en el botón **Save** en la esquina inferior derecha de la ventana nueva, tal y como se puede apreciar en la Ilustración 18.

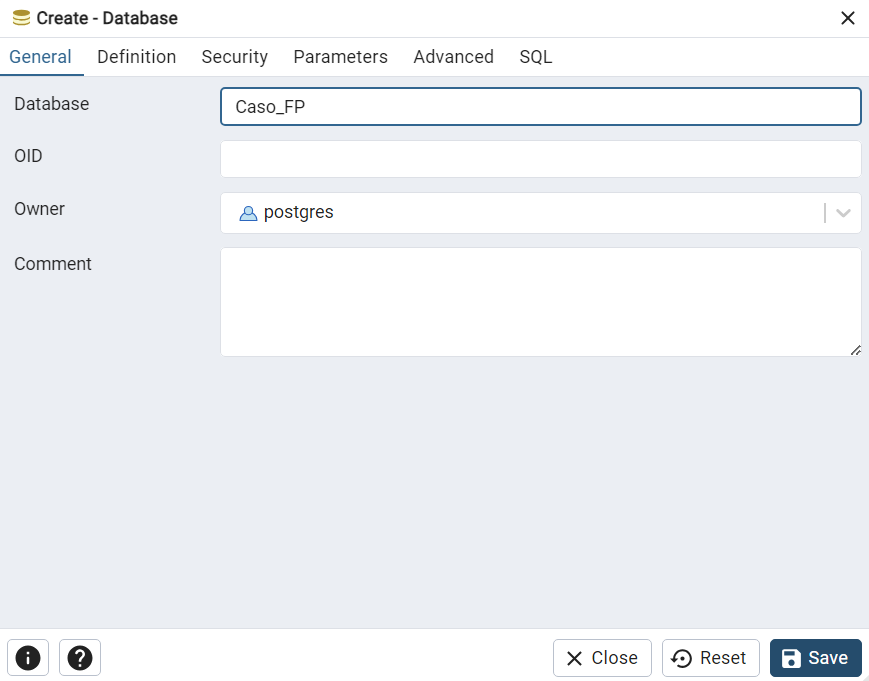


Ilustración 18. Paso 2 para crear base de datos

Una vez hecho lo anterior, nos aparecerá una nueva carpeta dentro de la carpeta de **Databases**, con el nombre que le pusimos a nuestra base de datos, ahora debemos darle click derecho a la carpeta mencionada recién y nos aparecerá un menú como se puede ver en la Ilustración 19, ahora la opción que nos interesa es la que dice Query Tool, le daremos click izquierdo sobre esta opción.

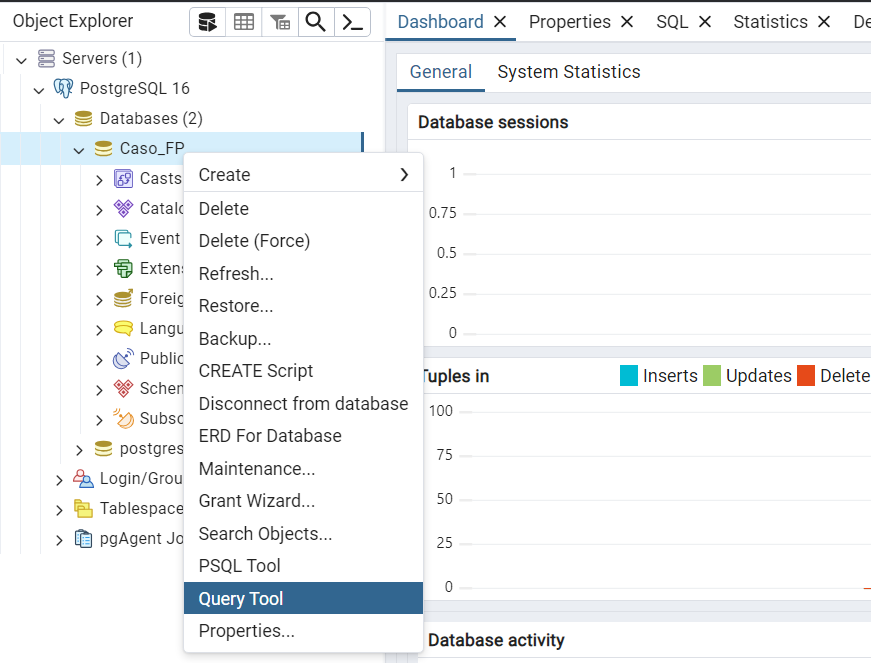


Ilustración 19. Paso 3 para crear base de datos

Se nos abrirá un área de trabajo para ejecutar consultas SQL, pero nuestro Script tenemos que abrirlo con la opción que se puede ver en la Ilustración 20.

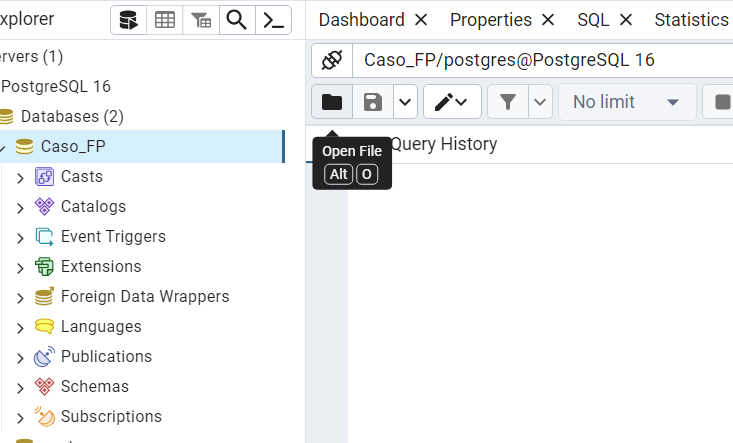


Ilustración 20. Paso 4 para crear base de datos

Se nos abrirá la ventana de la Ilustración 21 en la que tendremos que buscar el Script entre nuestros archivos seleccionarlo y darle click izquierdo al botón de **Abrir** de esa misma ventana.

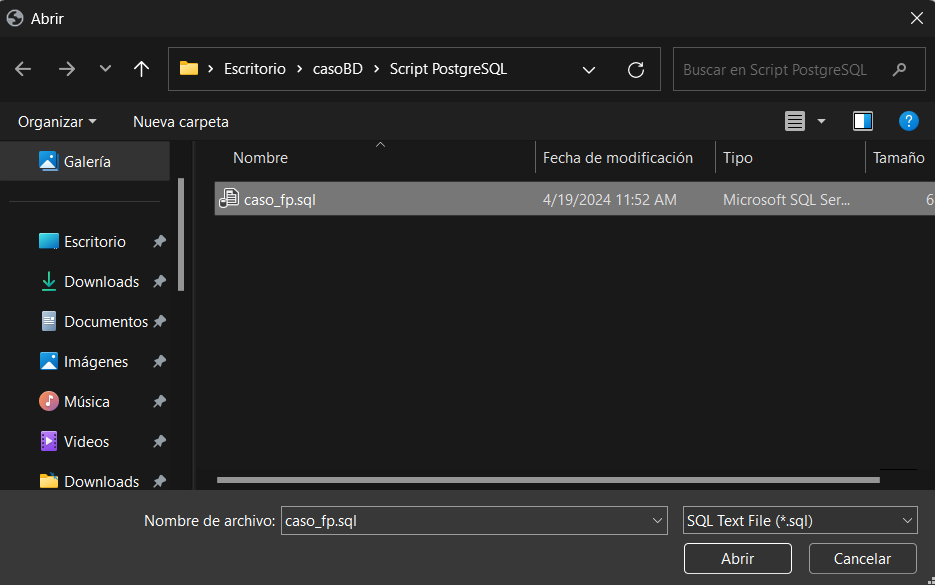


Ilustración 21. Paso 5 para crear base de datos

Una vez hecho esto se nos abrirá el Script con nuestras consultas como puede ver en la Ilustración 22 y en la que tendremos que dar click izquierdo sobre el botón Execute Script señalado en la Ilustración 22.

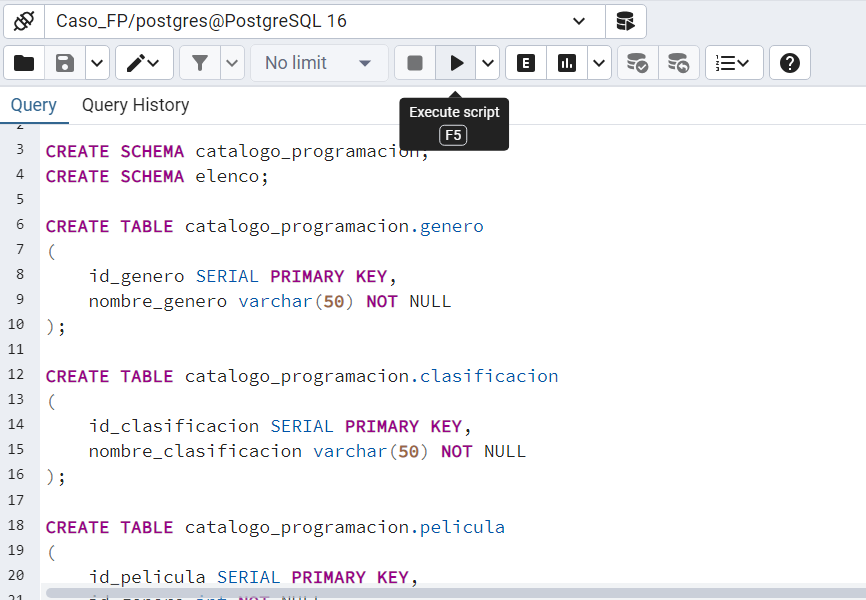


Ilustración 22. Paso 6 para crear base de datos

## Script DDL

CREATE SCHEMA catalogo\_programacion;

CREATE SCHEMA elenco;

CREATE TABLE catalogo\_programacion.genero

(

    id\_genero SERIAL PRIMARY KEY,

    nombre\_genero varchar(50) UNIQUE NOT NULL

);

CREATE TABLE catalogo\_programacion.clasificacion

(

    id\_clasificacion SERIAL PRIMARY KEY,

    nombre\_clasificacion varchar(50) NOT NULL UNIQUE

);

CREATE TABLE catalogo\_programacion.pelicula

(

    id\_pelicula SERIAL PRIMARY KEY,

    id\_genero int NOT NULL,

    id\_clasificacion int NOT NULL,

    nombre\_pelicula varchar(150) UNIQUE NOT NULL,

    anio\_relace date NOT NULL,

    duracion time NOT NULL,

    descripcion\_pelicula varchar(300),

    CONSTRAINT fk\_pelicula\_genero FOREIGN KEY (id\_genero)

        REFERENCES catalogo\_programacion.genero (id\_genero),

    CONSTRAINT fk\_pelicula\_clasificacion FOREIGN KEY (id\_clasificacion)

        REFERENCES catalogo\_programacion.clasificacion (id\_clasificacion)

);

CREATE TABLE catalogo\_programacion.compania\_productora

(

    id\_compania\_productora SERIAL PRIMARY KEY,

    nombre\_compania\_productora varchar(100) UNIQUE NOT NULL

);

CREATE TABLE catalogo\_programacion.pelicula\_compania\_productora

(

    id\_pelicula int NOT NULL,

    id\_compania\_productora int NOT NULL,

    CONSTRAINT pk\_pelicula\_compania\_productora PRIMARY KEY (id\_pelicula, id\_compania\_productora),

    CONSTRAINT fk\_pelicula\_compania\_productora\_pelicula FOREIGN KEY (id\_pelicula)

        REFERENCES catalogo\_programacion.pelicula (id\_pelicula),

    CONSTRAINT fk\_pelicula\_compania\_productora\_compania\_productora FOREIGN KEY (id\_compania\_productora)

        REFERENCES catalogo\_programacion.compania\_productora (id\_compania\_productora)

);

CREATE TABLE catalogo\_programacion.programacion

(

    id\_programacion SERIAL PRIMARY KEY,

    id\_pelicula int NOT NULL,

    hora time NOT NULL,

    fecha date NOT NULL,

    CONSTRAINT fk\_programacion\_pelicula FOREIGN KEY (id\_pelicula)

        REFERENCES catalogo\_programacion.pelicula (id\_pelicula)

);

CREATE TABLE catalogo\_programacion.espacio\_especial

(

    id\_espacio\_especial SERIAL PRIMARY KEY,

    nombre varchar(60) NOT NULL,

    descripcion varchar(300) NOT NULL,

    horario varchar(100) NOT NULL

);

CREATE TABLE catalogo\_programacion.espacio\_especial\_pelicula

(

    id\_espacio\_especial int NOT NULL,

    id\_pelicula int NOT NULL,

    fecha date NOT NULL,

    CONSTRAINT pk\_espacio\_especial\_pelicula PRIMARY KEY (id\_espacio\_especial, id\_pelicula),

    CONSTRAINT fk\_espacio\_especial\_pelicula\_espacio\_especial FOREIGN KEY (id\_espacio\_especial)

        REFERENCES catalogo\_programacion.espacio\_especial (id\_espacio\_especial),

    CONSTRAINT fk\_espacio\_especial\_pelicula\_pelicula FOREIGN KEY (id\_pelicula)

        REFERENCES catalogo\_programacion.pelicula (id\_pelicula)

);

CREATE TABLE elenco.nacionalidad

(

    id\_nacionalidad SERIAL PRIMARY KEY,

    nombre\_nacionalidad varchar(150) UNIQUE NOT NULL,

    pais varchar (100) UNIQUE NOT NULL

);

CREATE TABLE elenco.profesion

(

    id\_profesion SERIAL PRIMARY KEY,

    nombre\_profesion varchar(15) UNIQUE NOT NULL

);

CREATE TABLE elenco.artista

(

    id\_artista SERIAL PRIMARY KEY,

    id\_nacionalidad int NOT NULL,

    id\_profesion int NOT NULL,

    manager varchar(50),

    nombres varchar(50) NOT NULL,

    apellidos varchar(50) NOT NULL,

    biografia varchar(500) NOT NULL,

    fecha\_nacimiento date NOT NULL,

    CONSTRAINT fk\_artista\_nacionalidad FOREIGN KEY (id\_nacionalidad)

        REFERENCES elenco.nacionalidad (id\_nacionalidad),

    CONSTRAINT fk\_artista\_profesion FOREIGN KEY (id\_profesion)

        REFERENCES elenco.profesion (id\_profesion)

);

CREATE TABLE elenco.personaje

(

    id\_personaje SERIAL PRIMARY KEY,

    id\_artista int NOT NULL,

    id\_pelicula int NOT NULL,

    nombre\_personaje VARCHAR(100) NOT NULL,

    personaje\_principal BOOLEAN DEFAULT(FALSE),

    CONSTRAINT fk\_personaje\_artista FOREIGN KEY (id\_artista)

        REFERENCES elenco.artista (id\_artista),

    CONSTRAINT fk\_personaje\_pelicula FOREIGN KEY (id\_pelicula)

        REFERENCES catalogo\_programacion.pelicula (id\_pelicula)

);

CREATE TABLE elenco.sitie

(

    id\_artista int NOT NULL,

    fan\_site\_url varchar(150) UNIQUE NOT NULL,

    personal\_site\_url varchar(150) UNIQUE NOT NULL,

    CONSTRAINT pk\_sitie PRIMARY KEY (id\_artista),

    CONSTRAINT fk\_sitie\_artista FOREIGN KEY (id\_artista)

        REFERENCES elenco.artista (id\_artista)

);

CREATE TABLE elenco.tipo\_cargo

(

    id\_tipo\_cargo SERIAL PRIMARY KEY,

    nombre\_tipo\_cargo varchar(50) UNIQUE NOT NULL

);

CREATE TABLE elenco.pelicula\_artista\_cargo

(

    id\_pelicula int NOT NULL,

    id\_artista int NOT NULL,

    id\_tipo\_cargo int NOT NULL,

    CONSTRAINT pk\_pelicula\_artista\_cargo PRIMARY KEY (id\_pelicula, id\_artista, id\_tipo\_cargo),

    CONSTRAINT fk\_pelicula\_artista\_cargo\_pelicula FOREIGN KEY (id\_pelicula)

        REFERENCES catalogo\_programacion.pelicula (id\_pelicula),

    CONSTRAINT fk\_pelicula\_artista\_cargo\_artista FOREIGN KEY (id\_artista)

        REFERENCES elenco.artista (id\_artista),

    CONSTRAINT fk\_pelicula\_artista\_cargo\_tipo\_cargo FOREIGN KEY (id\_tipo\_cargo)

        REFERENCES elenco.tipo\_cargo (id\_tipo\_cargo)

);

## DML de Inserciones

-- Insert de datos por defecto de la tabla compania\_productora

INSERT INTO catalogo\_programacion.compania\_productora (nombre\_compania\_productora)

VALUES

    ('Warner Bros.'),

    ('Universal Pictures'),

    ('Paramount Pictures'),

    ('Walt Disney Pictures'),

    ('20th Century Studios'),

    ('Sony Pictures Entertainment'),

    ('Lionsgate'),

    ('Metro-Goldwyn-Mayer (MGM)'),

    ('New Line Cinema'),

    ('DreamWorks Pictures'),

    ('Columbia Pictures'),

    ('Focus Features'),

    ('Miramax Films'),

    ('Studio Ghibli'),

    ('A24'),

    ('Legendary Entertainment'),

    ('Fox Searchlight Pictures'),

    ('Participant Media'),

    ('Sony Pictures Classics'),

    ('United Artists');

-- Insert de datos por defecto de la tabla espacio\_espcial

INSERT INTO catalogo\_programacion.espacio\_especial (nombre, descripcion, horario)

VALUES

    ('Adrenalina al máximo', 'Espacio reservado para las películas de acción y suspenso al límite', 'Todos los sábados a las 13:00'),

    ('Ganadoras de Oscar', 'Este espacio está reservado exclusivamente para las películas que han logrado el máximo galardón de la Industria', 'Todos los domingos a las 15:00'),

    ('CINEX Independiente', 'En este espacio se presentan películas innovadoras de cineastas que trabajan fuera del ambiente de Hollywood', 'Todos los jueves a las 20:00');

-- Insert de datos por defecto de la tabla catalogo clasificacion

INSERT INTO catalogo\_programacion.clasificacion (nombre\_clasificacion)

VALUES ('G'), ('PG'), ('PG-13'), ('R'), ('NC-17'), ('X');

-- Insert de datos por defecto de la tabla genero

INSERT INTO catalogo\_programacion.genero (nombre\_genero)

VALUES

    ('Drama'),

    ('Comedia'),

    ('Romance'),

    ('Suspense'),

    ('Terror'),

    ('Acción'),

    ('Musical'),

    ('Sci-Fi'),

    ('Fantasía'),

    ('Infantil');

-- Insert de datos por defecto de la tabla pelicula

-- Películas de Drama

INSERT INTO catalogo\_programacion.pelicula (id\_genero, id\_clasificacion, nombre\_pelicula, anio\_relace, duracion, descripcion\_pelicula)

VALUES

    (1, 3, 'El Club de la Pelea', '1999-10-15', '02:19:00', 'Un hombre insomne y un vendedor de jabón forman un club de lucha clandestino.'),

    (1, 3, 'El Renacido', '2015-12-25', '02:36:00', 'Un hombre de la frontera lucha por sobrevivir después de ser atacado por un oso y dejado por muerto.'),

    (1, 3, 'El Discurso del Rey', '2010-12-25', '01:58:00', 'La historia real del rey Jorge VI, quien supera su tartamudez con la ayuda de un terapeuta del habla poco ortodoxo.'),

    (1, 3, 'La Lista de Schindler', '1993-12-15', '03:15:00', 'La historia de Oskar Schindler, quien salvó a más de mil judíos durante el Holocausto nazi.');

-- Películas de Comedia

INSERT INTO catalogo\_programacion.pelicula (id\_genero, id\_clasificacion, nombre\_pelicula, anio\_relace, duracion, descripcion\_pelicula)

VALUES

    (2, 3, 'Loco por Mary', '1998-07-15', '01:59:00', 'Un hombre busca reavivar su amor de la escuela secundaria, Mary, después de muchos años.'),

    (2, 3, 'La Máscara', '1994-07-29', '01:41:00', 'Un empleado de banco tímido se transforma en un superhéroe excéntrico después de encontrar una máscara antigua.'),

    (2, 3, 'Supercool', '2007-08-17', '01:53:00', 'Dos amigos organizan una fiesta de graduación para ser populares, pero las cosas se descontrolan rápidamente.'),

    (2, 3, 'Scary Movie', '2000-07-07', '01:28:00', 'Una parodia de películas de terror populares, donde un grupo de adolescentes intenta descubrir el misterio detrás de una serie de asesinatos.'),

    (2, 3, 'Dumb and Dumber', '1994-12-16', '01:47:00', 'Dos amigos incompetentes se embarcan en un viaje por carretera para devolver un maletín a su legítimo propietario.');

-- Películas de Romance

INSERT INTO catalogo\_programacion.pelicula (id\_genero, id\_clasificacion, nombre\_pelicula, anio\_relace, duracion, descripcion\_pelicula)

VALUES

    (3, 3, 'Orgullo y Prejuicio', '2005-11-23', '02:09:00', 'La historia de amor entre Elizabeth Bennet y el señor Darcy en la Inglaterra del siglo XIX.'),

    (3, 3, 'Titanic', '1997-12-19', '03:14:00', 'Un romance entre un pasajero y una joven aristócrata en el fatídico viaje inaugural del RMS Titanic.'),

    (3, 3, 'Antes del Amanecer', '1995-01-27', '01:41:00', 'Dos extraños se encuentran en un tren en Europa y pasan una noche juntos explorando Viena.'),

    (3, 3, 'El Diario de una Pasión', '2004-06-25', '02:04:00', 'La historia de amor entre un hombre y una mujer de diferentes clases sociales en la década de 1940.'),

    (3, 3, 'Un Paseo para Recordar', '2002-01-25', '01:41:00', 'La historia de amor entre un chico popular y una chica impopular en la escuela secundaria.');

-- Películas de Suspense

INSERT INTO catalogo\_programacion.pelicula (id\_genero, id\_clasificacion, nombre\_pelicula, anio\_relace, duracion, descripcion\_pelicula)

VALUES

    (4, 3, 'El Silencio de los Inocentes', '1991-02-14', '01:58:00', 'Una joven agente del FBI se une a un psicópata encarcelado para atrapar a otro asesino en serie.'),

    (4, 3, 'Sé Lo Que Hicieron el Verano Pasado', '1997-10-17', '01:41:00', 'Un grupo de amigos es acosado por un misterioso asesino un año después de un accidente automovilístico fatal.'),

    (4, 3, 'Psicosis', '1960-09-08', '01:49:00', 'Una secretaria emprende un viaje y se aloja en un motel dirigido por un propietario misterioso y su madre dominante.'),

    (4, 3, 'El Sexto Sentido', '1999-08-06', '01:47:00', 'Un niño que puede ver y comunicarse con los muertos busca la ayuda de un psicólogo atormentado.'),

    (4, 3, 'Seven', '1995-09-22', '02:07:00', 'Dos detectives investigan una serie de asesinatos relacionados con los siete pecados capitales.');

-- Películas de Terror

INSERT INTO catalogo\_programacion.pelicula (id\_genero, id\_clasificacion, nombre\_pelicula, anio\_relace, duracion, descripcion\_pelicula)

VALUES

    (5, 4, 'El Exorcista', '1973-12-26', '02:12:00', 'Una madre busca la ayuda de dos sacerdotes después de que su hija es poseída por una entidad demoníaca.'),

    (5, 4, 'El Resplandor', '1980-06-13', '02:26:00', 'Un hombre acepta un trabajo como cuidador de un hotel durante el invierno y comienza a experimentar sucesos paranormales.'),

    (5, 4, 'Actividad Paranormal', '2007-09-25', '01:26:00', 'Una pareja es acosada por una presencia demoníaca en su hogar y decide documentar sus experiencias.'),

    (5, 4, 'La Noche de Halloween', '1978-10-25', '01:31:00', 'Un asesino enmascarado acecha a los residentes de un pequeño pueblo en Halloween.'),

    (5, 4, 'El Conjuro', '2013-07-19', '01:52:00', 'Una pareja de investigadores paranormales ayuda a una familia a lidiar con la presencia maligna en su hogar.');

-- Películas de Acción

INSERT INTO catalogo\_programacion.pelicula (id\_genero, id\_clasificacion, nombre\_pelicula, anio\_relace, duracion, descripcion\_pelicula)

VALUES

    (6, 3, 'Duro de Matar', '1988-07-15', '02:12:00', 'Un policía de Nueva York lucha contra un grupo de terroristas que toman como rehenes a un rascacielos durante una fiesta navideña.'),

    (6, 3, 'Terminator 2: El Juicio Final', '1991-07-03', '02:32:00', 'Un cyborg enviado desde el futuro intenta proteger a un niño de un modelo avanzado de cyborg enviado para matarlo.'),

    (6, 3, 'Mad Max: Furia en el Camino', '2015-05-15', '02:00:00', 'En un mundo post-apocalíptico, un hombre solitario se une a un grupo de mujeres para escapar de un tirano y sus seguidores.'),

    (6, 3, 'Rápido y Furioso', '2001-06-22', '01:46:00', 'Un policía encubierto se infiltra en una banda de ladrones de autos y se ve obligado a elegir entre su deber y su lealtad.');

-- Películas de Musical

INSERT INTO catalogo\_programacion.pelicula (id\_genero, id\_clasificacion, nombre\_pelicula, anio\_relace, duracion, descripcion\_pelicula)

VALUES

    (7, 3, 'Cantando Bajo la Lluvia', '1952-04-11', '01:43:00', 'Un dúo de actores intenta hacer la transición del cine mudo al sonoro mientras se enamoran.'),

    (7, 3, 'El Mago de Oz', '1939-08-25', '01:42:00', 'Una joven es transportada a un mundo mágico y emprende un viaje para encontrar al Mago de Oz y volver a casa.'),

    (7, 3, 'La La Land', '2016-12-25', '02:08:00', 'Un pianista de jazz y una actriz se enamoran mientras persiguen sus sueños en Los Ángeles.'),

    (7, 3, 'Chicago', '2002-12-27', '01:53:00', 'Dos mujeres luchan por la fama y la libertad mientras intentan escapar de la cárcel en la década de 1920.'),

    (7, 3, 'Mamma Mia!', '2008-07-18', '01:49:00', 'Una joven se prepara para casarse y busca a su padre verdadero entre tres posibles candidatos.');

-- Películas de Sci-Fi

INSERT INTO catalogo\_programacion.pelicula (id\_genero, id\_clasificacion, nombre\_pelicula, anio\_relace, duracion, descripcion\_pelicula)

VALUES

    (8, 3, 'Blade Runner', '1982-06-25', '01:57:00', 'Un ex policía de Los Ángeles se convierte en cazador de androides fugitivos en un futuro distópico.'),

    (8, 3, 'Interestelar', '2014-11-07', '02:49:00', 'Un grupo de astronautas viaja a través de un agujero de gusano en busca de un nuevo hogar para la humanidad.'),

    (8, 3, '2001: Una Odisea del Espacio', '1968-04-02', '02:29:00', 'Un equipo de astronautas investiga un monolito misterioso mientras viaja hacia Júpiter.'),

    (8, 3, 'Matrix', '1999-03-31', '02:16:00', 'Un hacker descubre que el mundo que percibe es una simulación y se une a una rebelión para liberar a la humanidad de las máquinas.'),

    (8, 3, 'Avatar', '2009-12-18', '02:42:00', 'Un ex marine es enviado a una luna alienígena para infiltrarse en una tribu indígena y asegurar valiosos minerales.');

-- Películas de Fantasía

INSERT INTO catalogo\_programacion.pelicula (id\_genero, id\_clasificacion, nombre\_pelicula, anio\_relace, duracion, descripcion\_pelicula)

VALUES

    (9, 3, 'El Señor de los Anillos: La Comunidad del Anillo', '2001-12-19', '02:58:00', 'Un joven hobbit se embarca en una misión para destruir un anillo maligno y salvar la Tierra Media.'),

    (9, 3, 'Harry Potter y la Piedra Filosofal', '2001-11-16', '02:32:00', 'Un joven mago descubre que es el niño que vivió y asiste a la Escuela Hogwarts de Magia y Hechicería.'),

    (9, 3, 'La Historia Interminable', '1984-04-06', '01:42:00', 'Un niño se sumerge en un libro mágico y se embarca en una aventura para salvar el mundo de Fantasía.'),

    (9, 3, 'El Laberinto del Fauno', '2006-10-11', '01:58:00', 'Una niña se adentra en un mundo de fantasía mientras escapa de la España franquista de posguerra.');

-- Películas Infantiles

INSERT INTO catalogo\_programacion.pelicula (id\_genero, id\_clasificacion, nombre\_pelicula, anio\_relace, duracion, descripcion\_pelicula)

VALUES

    (10, 3, 'Toy Story', '1995-11-22', '01:21:00', 'Un grupo de juguetes cobra vida y se embarca en una aventura para encontrar a su dueño perdido.'),

    (10, 3, 'Frozen', '2013-11-27', '01:42:00', 'Una joven princesa busca a su hermana que ha sumido al reino en un invierno eterno con sus poderes de hielo.'),

    (10, 3, 'Buscando a Nemo', '2003-05-30', '01:40:00', 'Un pez payaso padre se embarca en un viaje para encontrar a su hijo desaparecido en el océano.'),

    (10, 3, 'El Rey León', '1994-06-15', '01:28:00', 'Un joven león debe enfrentar su destino y reclamar su lugar como rey después de la muerte de su padre.'),

    (10, 3, 'Monsters, Inc.', '2001-11-02', '01:32:00', 'Dos monstruos trabajan en una fábrica de sustos y se encuentran con una niña humana que cambia sus vidas.');

-- Insert de datos por defecto de la tabla programacion

INSERT INTO catalogo\_programacion.programacion (id\_pelicula, hora, fecha)

VALUES

    -- Miércoles 3 de Marzo de 2003

    (8, '06:00', '2003-03-03'),

    (9, '08:00', '2003-03-03'),

    (10, '10:00', '2003-03-03'),

    (11, '12:30', '2003-03-03'),

    -- Jueves 4 de Marzo de 2003

    (12, '06:00', '2003-03-04'),

    (13, '08:00', '2003-03-04'),

    (14, '10:15', '2003-03-04'),

    (15, '12:30', '2003-03-04'),

    (16, '14:30', '2003-03-04'),

    -- Viernes 5 de Marzo de 2003

    (17, '06:00', '2003-03-05'),

    (18, '08:00', '2003-03-05'),

    (19, '10:00', '2003-03-05'),

    (20, '12:30', '2003-03-05'),

    -- Sábado 6 de Marzo de 2003

    (1, '06:00', '2003-03-06'),

    (2, '08:00', '2003-03-06'),

    (3, '10:00', '2003-03-06'),

    (4, '12:30', '2003-03-06'),

    -- Domingo 7 de Marzo de 2003

    (5, '06:00', '2003-03-07'),

    (6, '08:00', '2003-03-07'),

    (7, '10:15', '2003-03-07'),

    (8, '12:30', '2003-03-07');

-- Insert de datos por defecto de la tabla espacio\_espacial\_pelicula

-- Espacio especial: Adrenalina al máximo

INSERT INTO catalogo\_programacion.espacio\_especial\_pelicula (id\_espacio\_especial, id\_pelicula, fecha)

VALUES

    (1, 5, '2024-04-19'),

    (1, 15, '2024-04-20'),

    (1, 20, '2024-04-21');

-- Espacio especial: Ganadoras de Oscar

INSERT INTO catalogo\_programacion.espacio\_especial\_pelicula (id\_espacio\_especial, id\_pelicula, fecha)

VALUES

    (2, 13, '2024-04-19'),

    (2, 14, '2024-04-20'),

    (2, 19, '2024-04-21');

-- Espacio especial: CINEX Independiente

INSERT INTO catalogo\_programacion.espacio\_especial\_pelicula (id\_espacio\_especial, id\_pelicula, fecha)

VALUES

    (3, 7, '2024-04-19'),

    (3, 10, '2024-04-20'),

    (3, 11, '2024-04-21');

-- Insert de datos por defecto de la tabla pelicula\_compania\_productora

INSERT INTO catalogo\_programacion.pelicula\_compania\_productora (id\_pelicula, id\_compania\_productora)

VALUES

    (1, 1),

    (2, 2),

    (3, 3),

    (4, 4),

    (5, 5),

    (6, 6),

    (7, 7),

    (8, 1),

    (9, 8),

    (10, 9),

    (11, 10),

    (12, 11),

    (13, 12),

    (14, 13),

    (15, 14),

    (16, 15),

    (17, 16),

    (18, 17),

    (19, 18),

    (20, 19),

    (21, 1),

    (22, 2),

    (23, 3),

    (24, 4),

    (25, 5),

    (26, 6),

    (27, 7),

    (28, 8),

    (29, 9),

    (30, 20),

    (31, 19),

    (32, 15),

    (33, 14),

    (34, 16),

    (35, 6),

    (36, 4),

    (37, 1),

    (38, 2),

    (39, 3),

    (40, 10),

    (41, 11),

    (42, 12),

    (43, 13),

    (44, 4),

    (45, 5),

    (46, 9),

    (47, 11);

-- Insert de datos por defecto de la tabla profesion

INSERT INTO elenco.profesion (nombre\_profesion)

VALUES

    ('director'),

    ('productor'),

    ('guionista'),

    ('compositor'),

    ('editor'),

    ('actor');

-- Insert de datos por defecto de la tabla nacionalidad

INSERT INTO elenco.nacionalidad (nombre\_nacionalidad, pais)

VALUES

    ('Afgana', 'Afganistán'),

    ('Albanesa', 'Albania'),

    ('Alemana', 'Alemania'),

    ('Andorrana', 'Andorra'),

    ('Angoleña', 'Angola'),

    ('Antiguana y Barbudana', 'Antigua y Barbuda'),

    ('Argentina', 'Argentina'),

    ('Armenia', 'Armenia'),

    ('Arubeña', 'Aruba'),

    ('Australiana', 'Australia'),

    ('Austríaca', 'Austria'),

    ('Azerbaiyana', 'Azerbaiyán'),

    ('Bahameña', 'Bahamas'),

    ('Bahreiní', 'Baréin'),

    ('Bangladesí', 'Bangladés'),

    ('Barbadense', 'Barbados'),

    ('Belga', 'Bélgica'),

    ('Beliceña', 'Belice'),

    ('Beninesa', 'Benín'),

    ('Bermudeña', 'Bermudas'),

    ('Bielorrusa', 'Bielorrusia'),

    ('Birmana', 'Birmania'),

    ('Boliviana', 'Bolivia'),

    ('Bosnia', 'Bosnia y Herzegovina'),

    ('Botsuana', 'Botsuana'),

    ('Brasileña', 'Brasil'),

    ('Británica', 'Reino Unido'),

    ('Bruneana', 'Brunéi'),

    ('Búlgara', 'Bulgaria'),

    ('Burkinesa', 'Burkina Faso'),

    ('Burundesa', 'Burundi'),

    ('Butanesa', 'Bután'),

    ('Cabo Verdiana', 'Cabo Verde'),

    ('Camboyana', 'Camboya'),

    ('Camerunesa', 'Camerún'),

    ('Canadiense', 'Canadá'),

    ('Centroafricana', 'República Centroafricana'),

    ('Chadiana', 'Chad'),

    ('Checa', 'República Checa'),

    ('Chilena', 'Chile'),

    ('China', 'China'),

    ('Chipriota', 'Chipre'),

    ('Comorense', 'Comoras'),

    ('Congoleña', 'República del Congo'),

    ('Costarricense', 'Costa Rica'),

    ('Croata', 'Croacia'),

    ('Cubana', 'Cuba'),

    ('Danés', 'Dinamarca'),

    ('Dominicana', 'República Dominicana'),

    ('Ecuatoriana', 'Ecuador'),

    ('Egipcia', 'Egipto'),

    ('Emiratí', 'Emiratos Árabes Unidos'),

    ('Eritrea', 'Eritrea'),

    ('Eslovaca', 'Eslovaquia'),

    ('Eslovena', 'Eslovenia'),

    ('Española', 'España'),

    ('Estadounidense', 'Estados Unidos'),

    ('Estonia', 'Estonia'),

    ('Etiopía', 'Etiopía'),

    ('Feroesa', 'Islas Feroe'),

    ('Filipina', 'Filipinas'),

    ('Finlandesa', 'Finlandia'),

    ('Fiyiana', 'Fiyi'),

    ('Francesa', 'Francia'),

    ('Gabonesa', 'Gabón'),

    ('Gambiana', 'Gambia'),

    ('Georgiana', 'Georgia'),

    ('Ghanesa', 'Ghana'),

    ('Gibraltareña', 'Gibraltar'),

    ('Granadina', 'Granada'),

    ('Griega', 'Grecia'),

    ('Groenlandesa', 'Groenlandia'),

    ('Guatemalteca', 'Guatemala'),

    ('Guineana', 'Guinea'),

    ('Guineana Ecuatorial', 'Guinea Ecuatorial'),

    ('Guyanesa', 'Guyana'),

    ('Haitiana', 'Haití'),

    ('Hondureña', 'Honduras'),

    ('Hongkonesa', 'Hong Kong'),

    ('Húngara', 'Hungría'),

    ('India', 'India'),

    ('Indonesa', 'Indonesia'),

    ('Iraní', 'Irán'),

    ('Iraquí', 'Irak'),

    ('Irlandesa', 'Irlanda'),

    ('Islandesa', 'Islandia'),

    ('Israelí', 'Israel'),

    ('Italiana', 'Italia'),

    ('Jamaicana', 'Jamaica'),

    ('Japonesa', 'Japón'),

    ('Jordana', 'Jordania'),

    ('Kazaja', 'Kazajistán'),

    ('Keniata', 'Kenia'),

    ('Kirguisa', 'Kirguistán'),

    ('Kiribatiana', 'Kiribati'),

    ('Kuwaití', 'Kuwait'),

    ('Laosiana', 'Laos'),

    ('Lesotense', 'Lesoto'),

    ('Letona', 'Letonia'),

    ('Libanesa', 'Líbano'),

    ('Liberiana', 'Liberia'),

    ('Libia', 'Libia'),

    ('Liechtensteiniana', 'Liechtenstein'),

    ('Lituana', 'Lituania'),

    ('Luxemburguesa', 'Luxemburgo'),

    ('Macao', 'Macao'),

    ('Macedonia', 'Macedonia del Norte'),

    ('Malasia', 'Malasia'),

    ('Malauí', 'Malaui'),

    ('Maldiva', 'Maldivas'),

    ('Malgache', 'Madagascar'),

    ('Maliense', 'Malí'),

    ('Maltesa', 'Malta'),

    ('Marfileña', 'Costa de Marfil'),

    ('Marroquí', 'Marruecos'),

    ('Mauriciana', 'Mauricio'),

    ('Mauritana', 'Mauritania'),

    ('Mexicana', 'México'),

    ('Micronesia', 'Micronesia'),

    ('Moldava', 'Moldavia'),

    ('Monegasca', 'Mónaco'),

    ('Mongola', 'Mongolia'),

    ('Montenegrina', 'Montenegro'),

    ('Mozambiqueña', 'Mozambique'),

    ('Namibia', 'Namibia'),

    ('Nauruana', 'Nauru'),

    ('Neerlandesa', 'Países Bajos'),

    ('Neozelandesa', 'Nueva Zelanda'),

    ('Nepalí', 'Nepal'),

    ('Nicaragüense', 'Nicaragua'),

    ('Nigeriana', 'Nigeria'),

    ('Nortecoreana', 'Corea del Norte'),

    ('Noruega', 'Noruega'),

    ('Omana', 'Omán'),

    ('Pakistani', 'Pakistán'),

    ('Palaos', 'Palaos'),

    ('Palestina', 'Estado de Palestina'),

    ('Panameña', 'Panamá'),

    ('Papú', 'Papúa Nueva Guinea'),

    ('Paraguaya', 'Paraguay'),

    ('Peruana', 'Perú'),

    ('Polaca', 'Polonia'),

    ('Portuguesa', 'Portugal'),

    ('Qatarí', 'Catar'),

    ('Ruandesa', 'Ruanda'),

    ('Rumana', 'Rumania'),

    ('Rusa', 'Rusia'),

    ('Salomonense', 'Islas Salomón'),

    ('Salvadoreña', 'El Salvador'),

    ('Samoa', 'Samoa'),

    ('Sanmarinense', 'San Marino'),

    ('Santaluciana', 'Santa Lucía'),

    ('Sanvicentina', 'San Vicente y las Granadinas'),

    ('Santomense', 'Santo Tomé y Príncipe'),

    ('Saudí', 'Arabia Saudita'),

    ('Senegalesa', 'Senegal'),

    ('Serbia', 'Serbia'),

    ('Seychellense', 'Seychelles'),

    ('Sierra Leonesa', 'Sierra Leona'),

    ('Singapurense', 'Singapur'),

    ('Siria', 'Siria'),

    ('Somalí', 'Somalia'),

    ('Srilanka', 'Sri Lanka'),

    ('Sudafricana', 'Sudáfrica'),

    ('Sudanesa', 'Sudán'),

    ('Sueca', 'Suecia'),

    ('Suiza', 'Suiza'),

    ('Surinamesa', 'Surinam'),

    ('Esvatini', 'Esvatini'),

    ('Tailandesa', 'Tailandia'),

    ('Tanzana', 'Tanzania'),

    ('Tayika', 'Tayikistán'),

    ('Timorense', 'Timor Oriental'),

    ('Togolesa', 'Togo'),

    ('Tongana', 'Tonga'),

    ('Trinitense', 'Trinidad y Tobago'),

    ('Tunecina', 'Túnez'),

    ('Turca', 'Turquía'),

    ('Turkmenistana', 'Turkmenistán'),

    ('Tuvaluana', 'Tuvalu'),

    ('Ucraniana', 'Ucrania'),

    ('Ugandesa', 'Uganda'),

    ('Uruguaya', 'Uruguay'),

    ('Uzbeka', 'Uzbekistán'),

    ('Vanuatense', 'Vanuatu'),

    ('Vaticana', 'Ciudad del Vaticano'),

    ('Venezolana', 'Venezuela'),

    ('Vietnamita', 'Vietnam'),

    ('Yemení', 'Yemen'),

    ('Yibutiana', 'Yibuti'),

    ('Zambiana', 'Zambia'),

    ('Zimbabuense', 'Zimbabue');

-- Insert de datos por defecto de la tabla artista

-- Directores

INSERT INTO elenco.artista (id\_nacionalidad, id\_profesion, manager, nombres, apellidos, biografia, fecha\_nacimiento)

VALUES

    (1, 1, 'Manager1', 'Christopher', 'Nolan', 'Biografía de Christopher Nolan...', '1970-07-30'),

    (2, 1, 'Manager2', 'Quentin', 'Tarantino', 'Biografía de Quentin Tarantino...', '1963-03-27'),

    (3, 1, 'Manager3', 'Martin', 'Scorsese', 'Biografía de Martin Scorsese...', '1942-11-17');

-- Productores

INSERT INTO elenco.artista (id\_nacionalidad, id\_profesion, nombres, apellidos, biografia, fecha\_nacimiento)

VALUES

    (4, 2, 'Kathleen', 'Kennedy', 'Biografía de Kathleen Kennedy...', '1953-06-05'),

    (5, 2, 'Jerry', 'Bruckheimer', 'Biografía de Jerry Bruckheimer...', '1945-09-21'),

    (6, 2, 'Brian', 'Grazer', 'Biografía de Brian Grazer...', '1951-07-12');

-- Guionistas

INSERT INTO elenco.artista (id\_nacionalidad, id\_profesion, nombres, apellidos, biografia, fecha\_nacimiento)

VALUES

    (7, 3, 'Aaron', 'Sorkin', 'Biografía de Aaron Sorkin...', '1961-06-09'),

    (8, 3, 'Greta', 'Gerwig', 'Biografía de Greta Gerwig...', '1983-08-04'),

    (9, 3, 'Charlie', 'Kaufman', 'Biografía de Charlie Kaufman...', '1958-11-01');

-- Compositores de música original

INSERT INTO elenco.artista (id\_nacionalidad, id\_profesion, nombres, apellidos, biografia, fecha\_nacimiento)

VALUES

    (10, 4, 'Hans', 'Zimmer', 'Biografía de Hans Zimmer...', '1957-09-12'),

    (11, 4, 'Ennio', 'Morricone', 'Biografía de Ennio Morricone...', '1928-11-10'),

    (12, 4, 'John', 'Williams', 'Biografía de John Williams...', '1932-02-08');

-- Editores

INSERT INTO elenco.artista (id\_nacionalidad, id\_profesion, nombres, apellidos, biografia, fecha\_nacimiento)

VALUES

    (13, 5, 'Thelma', 'Schoonmaker', 'Biografía de Thelma Schoonmaker...', '1940-01-03'),

    (14, 5, 'Michael', 'Kahn', 'Biografía de Michael Kahn...', '1935-12-08'),

    (15, 5, 'Sally', 'Menke', 'Biografía de Sally Menke...', '1953-12-17');

-- Actores

INSERT INTO elenco.artista (id\_nacionalidad, id\_profesion, manager, nombres, apellidos, biografia, fecha\_nacimiento)

VALUES

    (16, 6, 'Manager16', 'Tom', 'Hanks', 'Biografía de Tom Hanks...', '1956-07-09'),

    (17, 6, 'Manager17', 'Meryl', 'Streep', 'Biografía de Meryl Streep...', '1949-06-22'),

    (18, 6, 'Manager18', 'Leonardo', 'DiCaprio', 'Biografía de Leonardo DiCaprio...', '1974-11-11');

-- Insert de datos por defecto de la tabla site

-- Insertar sitios para artistas actores

INSERT INTO elenco.sitie (id\_artista, fan\_site\_url, personal\_site\_url)

VALUES

    (16, 'https://www.tomhanksfans.com', 'https://www.tomhanks.com'),

    (17, 'https://www.merylstreepfans.com', 'https://www.merylstreep.com'),

    (18, 'https://www.leonardodicapriofans.com', 'https://www.leonardodicaprio.com');

-- Insert de datos por defecto de la tabla tipo\_cargo

INSERT INTO elenco.tipo\_cargo (nombre\_tipo\_cargo)

VALUES

    ('Director'),

    ('Productor'),

    ('Guionista'),

    ('Compositor'),

    ('Editor'),

    ('Actor');

-- Insert de datos por defecto de la tabla pelicula\_artista\_cargo

-- Películas de Drama

INSERT INTO elenco.pelicula\_artista\_cargo (id\_pelicula, id\_artista, id\_tipo\_cargo)

VALUES

    -- El Club de la Pelea

    (1, 1, 1), -- Director

    (1, 2, 6), -- Actor principal

    -- El Renacido

    (2, 3, 1), -- Director

    (2, 4, 6), -- Actor principal

    -- El Discurso del Rey

    (3, 5, 1), -- Director

    (3, 6, 6), -- Actor principal

    -- La Lista de Schindler

    (4, 7, 1), -- Director

    (4, 8, 6); -- Actor principal

-- Películas de Comedia

INSERT INTO elenco.pelicula\_artista\_cargo (id\_pelicula, id\_artista, id\_tipo\_cargo)

VALUES

    -- Loco por Mary

    (5, 9, 1), -- Director

    (5, 10, 6), -- Actor principal

    -- La Máscara

    (6, 11, 1), -- Director

    (6, 12, 6), -- Actor principal

    -- Supercool

    (7, 13, 1), -- Director

    (7, 14, 6), -- Actor principal

    -- Scary Movie

    (8, 15, 1), -- Director

    (8, 16, 6), -- Actor principal

    -- Dumb and Dumber

    (9, 17, 1), -- Director

    (9, 18, 6); -- Actor principal

-- Películas de Romance

INSERT INTO elenco.pelicula\_artista\_cargo (id\_pelicula, id\_artista, id\_tipo\_cargo)

VALUES

    -- Orgullo y Prejuicio

    (10, 9, 1), -- Director

    (10, 2, 6), -- Actor principal

    -- Titanic

    (11, 1, 1), -- Director

    (11, 2, 6), -- Actor principal

    -- Antes del Amanecer

    (12, 3, 1), -- Director

    (12, 14, 6), -- Actor principal

    -- El Diario de una Pasión

    (13, 5, 1), -- Director

    (13, 2, 6), -- Actor principal

    -- Un Paseo para Recordar

    (14, 2, 1), -- Director

    (14, 16, 6); -- Actor principal

-- Películas de Suspense

INSERT INTO elenco.pelicula\_artista\_cargo (id\_pelicula, id\_artista, id\_tipo\_cargo)

VALUES

    -- El Silencio de los Inocentes

    (15, 2, 1), -- Director

    (15, 18, 6), -- Actor principal

    -- Sé Lo Que Hicieron el Verano Pasado

    (16, 1, 1), -- Director

    (16, 3, 6), -- Actor principal

    -- Psicosis

    (17, 3, 1), -- Director

    (17, 5, 6), -- Actor principal

    -- El Sexto Sentido

    (18, 5, 1), -- Director

    (18, 3, 6), -- Actor principal

    -- Seven

    (19, 7, 1), -- Director

    (19, 3, 6); -- Actor principal

-- Películas de Terror

INSERT INTO elenco.pelicula\_artista\_cargo (id\_pelicula, id\_artista, id\_tipo\_cargo)

VALUES

    -- El Exorcista

    (20, 3, 1), -- Director

    (20, 4, 6), -- Actor principal

    -- El Resplandor

    (21, 1, 1), -- Director

    (21, 12, 6), -- Actor principal

    -- Actividad Paranormal

    (22, 3, 1), -- Director

    (22, 4, 6), -- Actor principal

    -- La Noche de Halloween

    (23, 5, 1), -- Director

    (23, 6, 6), -- Actor principal

    -- El Conjuro

    (24, 7, 1), -- Director

    (24, 1, 6); -- Actor principal

-- Películas de Acción

INSERT INTO elenco.pelicula\_artista\_cargo (id\_pelicula, id\_artista, id\_tipo\_cargo)

VALUES

    -- Duro de Matar

    (25, 2, 1), -- Director

    (25, 3, 6), -- Actor principal

    -- Terminator 2: El Juicio Final

    (26, 4, 1), -- Director

    (26, 5, 6), -- Actor principal

    -- Mad Max: Furia en el Camino

    (27, 6, 1), -- Director

    (27, 7, 6), -- Actor principal

    -- Rápido y Furioso

    (28, 8, 1), -- Director

    (28, 9, 6); -- Actor principal

-- Películas de Musical

INSERT INTO elenco.pelicula\_artista\_cargo (id\_pelicula, id\_artista, id\_tipo\_cargo)

VALUES

    -- Cantando Bajo la Lluvia

    (29, 10, 1), -- Director

    (29, 11, 6), -- Actor principal

    -- El Mago de Oz

    (30, 12, 1), -- Director

    (30, 13, 6), -- Actor principal

    -- La La Land

    (31, 14, 1), -- Director

    (31, 15, 6), -- Actor principal

    -- Chicago

    (32, 16, 1), -- Director

    (32, 17, 6), -- Actor principal

    -- Mamma Mia!

    (33, 18, 1), -- Director

    (33, 16, 6); -- Actor principal

-- Películas de Sci-Fi

INSERT INTO elenco.pelicula\_artista\_cargo (id\_pelicula, id\_artista, id\_tipo\_cargo)

VALUES

    -- Blade Runner

    (34, 2, 1), -- Director

    (34, 1, 6), -- Actor principal

    -- Interestelar

    (35, 2, 1), -- Director

    (35, 3, 6), -- Actor principal

    -- 2001: Una Odisea del Espacio

    (36, 2, 1), -- Director

    (36, 5, 6), -- Actor principal

    -- Matrix

    (37, 6, 1), -- Director

    (37, 7, 6), -- Actor principal

    -- Avatar

    (38, 2, 1), -- Director

    (38, 9, 6); -- Actor principal

-- Películas de Fantasía

INSERT INTO elenco.pelicula\_artista\_cargo (id\_pelicula, id\_artista, id\_tipo\_cargo)

VALUES

    -- El Señor de los Anillos: La Comunidad del Anillo

    (39, 3, 1), -- Director

    (39, 1, 6), -- Actor principal

    -- Harry Potter y la Piedra Filosofal

    (40, 3, 1), -- Director

    (40, 3, 6), -- Actor principal

    -- La Historia Interminable

    (41, 4, 1), -- Director

    (41, 5, 6), -- Actor principal

    -- El Laberinto del Fauno

    (42, 6, 1), -- Director

    (42, 7, 6); -- Actor principal

-- Películas Infantiles

INSERT INTO elenco.pelicula\_artista\_cargo (id\_pelicula, id\_artista, id\_tipo\_cargo)

VALUES

    -- Toy Story

    (43, 3, 1), -- Director

    (43, 9, 6), -- Actor principal

    -- Frozen

    (44, 4, 1), -- Director

    (44, 6, 6), -- Actor principal

    -- Buscando a Nemo

    (45, 2, 1), -- Director

    (45, 3, 6), -- Actor principal

    -- El Rey León

    (46, 4, 1), -- Director

    (46, 5, 6), -- Actor principal

    -- Monsters, Inc.

    (47, 6, 1), -- Director

    (47, 7, 6); -- Actor principal

-- Películas de Drama

INSERT INTO elenco.pelicula\_artista\_cargo (id\_pelicula, id\_artista, id\_tipo\_cargo)

VALUES

    -- El Club de la Pelea

    (1, 3, 2), -- Productor

    (1, 4, 3), -- Guionista

    (1, 5, 4), -- Compositor de música original

    (1, 6, 5), -- Editor

    -- El Renacido

    (2, 7, 2), -- Productor

    (2, 8, 3), -- Guionista

    (2, 9, 4), -- Compositor de música original

    (2, 10, 5), -- Editor

    -- El Discurso del Rey

    (3, 11, 2), -- Productor

    (3, 12, 3), -- Guionista

    (3, 13, 4), -- Compositor de música original

    (3, 14, 5), -- Editor

    -- La Lista de Schindler

    (4, 15, 2), -- Productor

    (4, 16, 3), -- Guionista

    (4, 17, 4), -- Compositor de música original

    (4, 18, 5); -- Editor

-- Películas de Comedia

INSERT INTO elenco.pelicula\_artista\_cargo (id\_pelicula, id\_artista, id\_tipo\_cargo)

VALUES

    -- Loco por Mary

    (5, 9, 2), -- Productor

    (5, 2, 3), -- Guionista

    (5, 1, 4), -- Compositor de música original

    (5, 2, 5), -- Editor

    -- La Máscara

    (6, 3, 2), -- Productor

    (6, 4, 3), -- Guionista

    (6, 5, 4), -- Compositor de música original

    (6, 2, 5), -- Editor

    -- Supercool

    (7, 7, 2), -- Productor

    (7, 8, 3), -- Guionista

    (7, 9, 4), -- Compositor de música original

    (7, 3, 5), -- Editor

    -- Scary Movie

    (8, 1, 2), -- Productor

    (8, 2, 3), -- Guionista

    (8, 3, 4), -- Compositor de música original

    (8, 4, 5), -- Editor

    -- Dumb and Dumber

    (9, 5, 2), -- Productor

    (9, 6, 3), -- Guionista

    (9, 7, 4), -- Compositor de música original

    (9, 8, 5); -- Editor

-- Películas de Romance

INSERT INTO elenco.pelicula\_artista\_cargo (id\_pelicula, id\_artista, id\_tipo\_cargo)

VALUES

    -- Orgullo y Prejuicio

    (10, 9, 2), -- Productor

    (10, 4, 3), -- Guionista

    (10, 1, 4), -- Compositor de música original

    (10, 2, 5), -- Editor

    -- Titanic

    (11, 3, 2), -- Productor

    (11, 4, 3), -- Guionista

    (11, 5, 4), -- Compositor de música original

    (11, 6, 5), -- Editor

    -- Antes del Amanecer

    (12, 7, 2), -- Productor

    (12, 1, 3), -- Guionista

    (12, 2, 4), -- Compositor de música original

    (12, 3, 5), -- Editor

    -- El Diario de una Pasión

    (13, 4, 2), -- Productor

    (13, 5, 3), -- Guionista

    (13, 6, 4), -- Compositor de música original

    (13, 7, 5), -- Editor

    -- Un Paseo para Recordar

    (14, 8, 2), -- Productor

    (14, 9, 3), -- Guionista

    (14, 10, 4), -- Compositor de música original

    (14, 11, 5); -- Editor

-- Películas de Suspense

INSERT INTO elenco.pelicula\_artista\_cargo (id\_pelicula, id\_artista, id\_tipo\_cargo)

VALUES

    -- El Silencio de los Inocentes

    (15, 12, 2), -- Productor

    (15, 13, 3), -- Guionista

    (15, 14, 4), -- Compositor de música original

    (15, 15, 5), -- Editor

    -- Sé Lo Que Hicieron el Verano Pasado

    (16, 16, 2), -- Productor

    (16, 17, 3), -- Guionista

    (16, 18, 4), -- Compositor de música original

    (16, 1, 5), -- Editor

    -- Psicosis

    (17, 2, 2), -- Productor

    (17, 1, 3), -- Guionista

    (17, 2, 4), -- Compositor de música original

    (17, 3, 5), -- Editor

    -- El Sexto Sentido

    (18, 4, 2), -- Productor

    (18, 5, 3), -- Guionista

    (18, 6, 4), -- Compositor de música original

    (18, 7, 5), -- Editor

    -- Seven

    (19, 8, 2), -- Productor

    (19, 9, 3), -- Guionista

    (19, 10, 4), -- Compositor de música original

    (19, 1, 5); -- Editor

-- Películas de Terror

INSERT INTO elenco.pelicula\_artista\_cargo (id\_pelicula, id\_artista, id\_tipo\_cargo)

VALUES

    -- El Exorcista

    (20, 2, 2), -- Productor

    (20, 3, 3), -- Guionista

    (20, 4, 4), -- Compositor de música original

    (20, 5, 5), -- Editor

    -- El Resplandor

    (21, 6, 2), -- Productor

    (21, 7, 3), -- Guionista

    (21, 8, 4), -- Compositor de música original

    (21, 9, 5), -- Editor

    -- Actividad Paranormal

    (22, 4, 2), -- Productor

    (22, 1, 3), -- Guionista

    (22, 2, 4), -- Compositor de música original

    (22, 3, 5), -- Editor

    -- La Noche de Halloween

    (23, 4, 2), -- Productor

    (23, 5, 3), -- Guionista

    (23, 6, 4), -- Compositor de música original

    (23, 7, 5), -- Editor

    -- El Conjuro

    (24, 1, 2), -- Productor

    (24, 2, 3), -- Guionista

    (24, 3, 4), -- Compositor de música original

    (24, 4, 5); -- Editor

-- Insert de datos por defecto de la tabla personaje

INSERT INTO elenco.personaje (id\_artista, id\_pelicula, nombre\_personaje, personaje\_principal)

VALUES

    -- El Club de la Pelea

    (2, 1, 'Tyler Durden', TRUE),

    -- El Renacido

    (4, 2, 'Hugh Glass', TRUE),

    -- El Discurso del Rey

    (6, 3, 'Rey Jorge VI', TRUE),

    -- La Lista de Schindler

    (8, 4, 'Oskar Schindler', TRUE),

    -- Loco por Mary

    (10, 5, 'Ted Stroehmann', TRUE),

    -- La Máscara

    (12, 6, 'Stanley Ipkiss / La Máscara', TRUE),

    -- Supercool

    (14, 7, 'Evan', TRUE),

    -- Scary Movie

    (16, 8, 'Cindy Campbell', TRUE),

    -- Dumb and Dumber

    (18, 9, 'Lloyd Christmas', TRUE),

    -- Orgullo y Prejuicio

    (9, 10, 'Elizabeth Bennet', TRUE),

    -- Titanic

    (1, 11, 'Jack Dawson', TRUE),

    -- Antes del Amanecer

    (14, 12, 'Jesse', TRUE),

    -- El Diario de una Pasión

    (2, 13, 'Noah Calhoun', TRUE),

    -- Un Paseo para Recordar

    (16, 14, 'Landon Carter', TRUE),

    -- El Silencio de los Inocentes

    (18, 15, 'Dr. Hannibal Lecter', TRUE),

    -- Sé Lo Que Hicieron el Verano Pasado

    (3, 16, 'Julie James', TRUE),

    -- Psicosis

    (5, 17, 'Norman Bates', TRUE),

    -- El Sexto Sentido

    (3, 18, 'Cole Sear', TRUE),

    -- Seven

    (3, 19, 'David Mills', TRUE),

    -- El Exorcista

    (4, 20, 'Regan MacNeil', TRUE),

    -- El Resplandor

    (12, 21, 'Jack Torrance', TRUE),

    -- Actividad Paranormal

    (4, 22, 'Katie', TRUE),

    -- La Noche de Halloween

    (6, 23, 'Michael Myers', TRUE),

    -- El Conjuro

    (1, 24, 'Ed Warren', TRUE),

    -- Duro de Matar

    (3, 25, 'John McClane', TRUE),

    -- Terminator 2: El Juicio Final

    (5, 26, 'Terminator (T-800)', TRUE),

    -- Mad Max: Furia en el Camino

    (7, 27, 'Max Rockatansky', TRUE),

    -- Rápido y Furioso

    (8, 28, 'Dominic Toretto', TRUE),

    -- Cantando Bajo la Lluvia

    (10, 29, 'Don Lockwood', TRUE),

    -- El Mago de Oz

    (12, 30, 'Dorothy Gale', TRUE),

    -- La La Land

    (14, 31, 'Mia Dolan', TRUE),

    -- Chicago

    (16, 32, 'Roxie Hart', TRUE),

    -- Mamma Mia!

    (16, 33, 'Sophie Sheridan', TRUE),

    -- Blade Runner

    (1, 34, 'Rick Deckard', TRUE),

    -- Interestelar

    (2, 35, 'Cooper', TRUE),

    -- 2001: Una Odisea del Espacio

    (2, 36, 'Dr. Dave Bowman', TRUE),

    -- Matrix

    (6, 37, 'Neo', TRUE),

    -- Avatar

    (9, 38, 'Jake Sully', TRUE),

    -- El Señor de los Anillos: La Comunidad del Anillo

    (1, 39, 'Frodo Bolsón', TRUE),

    -- Harry Potter y la Piedra Filosofal

    (3, 40, 'Harry Potter', TRUE),

    -- La Historia Interminable

    (4, 41, 'Bastian Balthazar Bux', TRUE),

    -- El Laberinto del Fauno

    (7, 42, 'Ofelia', TRUE),

    -- Toy Story

    (9, 43, 'Woody', TRUE),

    -- Frozen

    (6, 44, 'Elsa', TRUE),

    -- Buscando a Nemo

    (3, 45, 'Marlin', TRUE),

    -- El Rey León

    (5, 46, 'Simba', TRUE),

    -- Monsters, Inc.

    (7, 47, 'Sulley', TRUE);

## Funciones almacenadas (CRUD)

--Insertar Genero

CREATE OR REPLACE FUNCTION insertar\_genero(

    nombre\_genero varchar(50)

) RETURNS VOID AS $$

BEGIN

    INSERT INTO catalogo\_programacion.genero (nombre\_genero)

    VALUES (nombre\_genero);

END;

$$ LANGUAGE plpgsql;

--Mostrar todos los generos

CREATE OR REPLACE FUNCTION mostrar\_genero()

RETURNS TABLE (

    id\_genero int,

    nombre\_genero varchar(50)

) AS $$

BEGIN

    RETURN QUERY

    SELECT g.id\_genero, g.nombre\_genero

    FROM catalogo\_programacion.genero g;

END;

$$ LANGUAGE plpgsql;

SELECT \*

FROM mostrar\_genero();

--Mostrar un genero

CREATE OR REPLACE FUNCTION mostrar\_genero\_por\_id(

    p\_id\_genero int

) RETURNS TABLE (

    id\_genero int,

    nombre\_genero varchar(50)

) AS $$

BEGIN

    RETURN QUERY

    SELECT g.id\_genero, g.nombre\_genero

    FROM catalogo\_programacion.genero g

    WHERE g.id\_genero = p\_id\_genero;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_genero\_por\_id(2);

--Eliminar

CREATE OR REPLACE FUNCTION eliminar\_genero(

    IN p\_id\_genero int

) RETURNS VOID AS $$

BEGIN

    DELETE

    FROM catalogo\_programacion.genero

    WHERE catalogo\_programacion.genero.id\_genero = p\_id\_genero;

END;

$$ LANGUAGE plpgsql;

--Actualizar Genero

CREATE OR REPLACE FUNCTION actualizar\_genero(

    IN p\_id\_genero int,

    IN nuevo\_nombre\_genero varchar(50)

) RETURNS VOID AS $$

BEGIN

    UPDATE catalogo\_programacion.genero

    SET nombre\_genero = nuevo\_nombre\_genero

    WHERE id\_genero = p\_id\_genero;

END;

$$ LANGUAGE plpgsql;

-------------------------------------------------------------------------------------------------------------

--Insertar clasificacion

CREATE OR REPLACE FUNCTION insertar\_clasificacion(

    nombre\_clasificacion varchar(50)

) RETURNS VOID AS $$

BEGIN

    INSERT INTO catalogo\_programacion.clasificacion (nombre\_clasificacion)

    VALUES (nombre\_clasificacion);

END;

$$ LANGUAGE plpgsql;

--Mostrar Clasificación

CREATE OR REPLACE FUNCTION mostrar\_clasificacion()

RETURNS TABLE (

    id\_clasificacion int,

    nombre\_clasificacion varchar(50)

) AS $$

BEGIN

    RETURN QUERY

    SELECT c.id\_clasificacion, c.nombre\_clasificacion

    FROM catalogo\_programacion.clasificacion c;

END;

$$ LANGUAGE plpgsql;

SELECT \*

FROM mostrar\_clasificacion();

--Mostrar una clasificacion

CREATE OR REPLACE FUNCTION mostrar\_clasificacion\_por\_id(

    p\_id\_clasificacion int

) RETURNS TABLE (

    id\_clasifiacion int,

    nombre\_clasificacion varchar(50)

) AS $$

BEGIN

    RETURN QUERY

    SELECT c.id\_clasificacion, c.nombre\_clasificacion

    FROM catalogo\_programacion.clasificacion c

    WHERE c.id\_clasificacion = p\_id\_clasificacion;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_clasificacion\_por\_id(2);

--ELIMINAR

CREATE OR REPLACE FUNCTION eliminar\_clasificacion(

    IN p\_id\_clasificacion int

) RETURNS VOID AS $$

BEGIN

    DELETE

    FROM catalogo\_programacion.clasificacion

    WHERE id\_clasificacion = p\_id\_clasificacion;

END;

$$ LANGUAGE plpgsql;

--Actualizar

CREATE OR REPLACE FUNCTION actualizar\_clasificacion(

    p\_id\_clasificacion int,

    nuevo\_nombre\_clasificacion varchar(50)

) RETURNS VOID AS $$

BEGIN

    UPDATE catalogo\_programacion.clasificacion

    SET nombre\_clasificacion = nuevo\_nombre\_clasificacion

    WHERE id\_clasificacion = p\_id\_clasificacion;

END;

$$ LANGUAGE plpgsql;

-------------------------------------------------------------------------------------------------------------------

--Insertar Pelicula

CREATE OR REPLACE FUNCTION insertar\_pelicula(

    nombre\_pelicula varchar(150),

    id\_genero int,

    id\_clasificacion int,

    anio\_relace date,

    duracion time,

    descripcion\_pelicula varchar(300)

) RETURNS VOID AS $$

BEGIN

    INSERT INTO catalogo\_programacion.pelicula (nombre\_pelicula, id\_genero, id\_clasificacion, anio\_relace, duracion, descripcion\_pelicula)

    VALUES (nombre\_pelicula, id\_genero, id\_clasificacion, anio\_relace, duracion, descripcion\_pelicula);

END;

$$ LANGUAGE plpgsql;

SELECT \*

FROM catalogo\_programacion.pelicula

--Mostrar pelicula

CREATE OR REPLACE FUNCTION mostrar\_pelicula()

RETURNS TABLE (

    id\_pelicula int,

    id\_genero int,

    id\_clasificacion int,

    nombre\_pelicula varchar(150),

    anio\_relace date,

    duracion time,

    descripcion\_pelicula varchar(300)

) AS $$

BEGIN

    RETURN QUERY

    SELECT p.id\_pelicula, p.id\_genero, p.id\_clasificacion, p.nombre\_pelicula,

           p.anio\_relace, p.duracion, p.descripcion\_pelicula

    FROM catalogo\_programacion.pelicula p;

END;

$$ LANGUAGE plpgsql;

SELECT \*

FROM mostrar\_pelicula();

--Mostrar una pelicula

CREATE OR REPLACE FUNCTION mostrar\_pelicula\_por\_id(

    p\_id\_pelicula int

) RETURNS TABLE (

    id\_pelicula int,

    id\_genero int,

    id\_clasificacion int,

    nombre\_pelicula varchar(150),

    anio\_relace date,

    duracion time,

    descripcion\_pelicula varchar(300)

) AS $$

BEGIN

    RETURN QUERY

    SELECT p.id\_pelicula, p.id\_genero, p.id\_clasificacion, p.nombre\_pelicula,

           p.anio\_relace, p.duracion, p.descripcion\_pelicula

    FROM catalogo\_programacion.pelicula p

    WHERE p.id\_pelicula = p\_id\_pelicula;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_pelicula\_por\_id(2);

--Eliminar Pelicula

CREATE OR REPLACE FUNCTION eliminar\_pelicula(

    p\_id\_pelicula int

) RETURNS VOID AS $$

BEGIN

    DELETE

    FROM catalogo\_programacion.pelicula

    WHERE id\_pelicula = p\_id\_pelicula;

END;

$$ LANGUAGE plpgsql;

--Actualizar

CREATE OR REPLACE FUNCTION actualizar\_pelicula(

    p\_id\_pelicula int,

    p\_nombre\_pelicula varchar(150),

    p\_id\_genero int,

    p\_id\_clasificacion int,

    p\_anio\_relace date,

    p\_duracion time,

    p\_descripcion\_pelicula varchar(300)

) RETURNS VOID AS $$

BEGIN

    UPDATE catalogo\_programacion.pelicula

    SET nombre\_pelicula = p\_nombre\_pelicula,

        id\_genero = p\_id\_genero,

        id\_clasificacion = p\_id\_clasificacion,

        anio\_relace = p\_anio\_relace,

        duracion = p\_duracion,

        descripcion\_pelicula = p\_descripcion\_pelicula

    WHERE id\_pelicula = p\_id\_pelicula;

END;

$$ LANGUAGE plpgsql;

--------------------------------------------------------------------------------------------------

--Insertar compania productora

CREATE OR REPLACE FUNCTION insertar\_compania\_productora(

    nombre\_compania\_productora varchar(100)

) RETURNS VOID AS $$

BEGIN

    INSERT INTO catalogo\_programacion.compania\_productora (nombre\_compania\_productora)

    VALUES (nombre\_compania\_productora);

END;

$$ LANGUAGE plpgsql;

--Mostrar companias productoras

CREATE OR REPLACE FUNCTION mostrar\_compania\_productora()

RETURNS TABLE (

    id\_compania\_productora int,

    nombre\_compania\_productora varchar(100)

) AS $$

BEGIN

    RETURN QUERY

    SELECT cp.id\_compania\_productora, cp.nombre\_compania\_productora

    FROM catalogo\_programacion.compania\_productora cp;

END;

$$ LANGUAGE plpgsql;

SELECT \*

FROM mostrar\_compania\_productora();

--Mostrar una compania

CREATE OR REPLACE FUNCTION mostrar\_compania\_por\_id(

    p\_id\_compania\_productora int

) RETURNS TABLE (

    id\_compania\_productora int,

    nombre\_compania\_productora varchar(100)

) AS $$

BEGIN

    RETURN QUERY

    SELECT cp.id\_compania\_productora, cp.nombre\_compania\_productora

    FROM catalogo\_programacion.compania\_productora cp;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_compania\_por\_id(2);

--Eliminar

CREATE OR REPLACE FUNCTION eliminar\_compania\_productora(

    p\_id\_compania\_productora int

) RETURNS VOID AS $$

BEGIN

    DELETE

    FROM catalogo\_programacion.compania\_productora

    WHERE id\_compania\_productora = p\_id\_compania\_productora;

END;

$$ LANGUAGE plpgsql;

--Actualizar

CREATE OR REPLACE FUNCTION actualizar\_compania\_productora(

    p\_id\_compania\_productora int,

    p\_nombre\_compania\_productora varchar(100)

) RETURNS VOID AS $$

BEGIN

    UPDATE catalogo\_programacion.compania\_productora

    SET nombre\_compania\_productora = p\_nombre\_compania\_productora

    WHERE id\_compania\_productora = p\_id\_compania\_productora;

END;

$$ LANGUAGE plpgsql;

--------------------------------------------------------------------------------------------------------------

--Insertar

CREATE OR REPLACE FUNCTION insertar\_pelicula\_compania\_productora(

    p\_id\_pelicula int,

    p\_id\_compania\_productora int

) RETURNS VOID AS $$

BEGIN

    INSERT INTO catalogo\_programacion.pelicula\_compania\_productora (id\_pelicula, id\_compania\_productora)

    VALUES (p\_id\_pelicula, p\_id\_compania\_productora);

END;

$$ LANGUAGE plpgsql;

--Mostrar

CREATE OR REPLACE FUNCTION mostrar\_pelicula\_compania\_productora()

RETURNS TABLE (

    id\_pelicula int,

    id\_compania\_productora int

) AS $$

BEGIN

    RETURN QUERY

    SELECT pcp.id\_pelicula, pcp.id\_compania\_productora

    FROM catalogo\_programacion.pelicula\_compania\_productora pcp;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_pelicula\_compania\_productora();

--Eliminar

CREATE OR REPLACE FUNCTION eliminar\_pelicula\_compania\_productora(

    p\_id\_pelicula int,

    p\_id\_compania\_productora int

) RETURNS VOID AS $$

BEGIN

    DELETE

    FROM catalogo\_programacion.pelicula\_compania\_productora

    WHERE id\_pelicula = p\_id\_pelicula AND id\_compania\_productora = p\_id\_compania\_productora;

END;

$$ LANGUAGE plpgsql;

--Actualizar

CREATE OR REPLACE FUNCTION actualizar\_pelicula\_compania\_productora(

    p\_id\_pelicula int,

    p\_id\_compania\_productora int,

    p\_nuevo\_id\_pelicula int,

    p\_nuevo\_id\_compania\_productora int

) RETURNS VOID AS $$

BEGIN

    UPDATE catalogo\_programacion.pelicula\_compania\_productora

    SET id\_pelicula = p\_nuevo\_id\_pelicula,

        id\_compania\_productora = p\_nuevo\_id\_compania\_productora

    WHERE id\_pelicula = p\_id\_pelicula AND id\_compania\_productora = p\_id\_compania\_productora;

END;

$$ LANGUAGE plpgsql;

-----------------------------------------------------------------------------------------------------------------

--Insertar Programacion

CREATE OR REPLACE FUNCTION insertar\_programacion(

    id\_pelicula int,

    hora time,

    fecha date

) RETURNS VOID AS $$

BEGIN

    INSERT INTO catalogo\_programacion.programacion (id\_pelicula, hora, fecha)

    VALUES (id\_pelicula, hora, fecha);

END;

$$ LANGUAGE plpgsql;

--Mostrar programacion

CREATE OR REPLACE FUNCTION mostrar\_programacion()

RETURNS TABLE (

    id\_programacion int,

    hora time,

    fecha date

) AS $$

BEGIN

    RETURN QUERY

    SELECT p.id\_programacion, p.hora, p.fecha

    FROM catalogo\_programacion.programacion p;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_programacion();

--Mostrar una programacion

CREATE OR REPLACE FUNCTION mostrar\_programacion\_por\_id(

    p\_id\_programacion int

) RETURNS TABLE (

    id\_programacion int,

    hora time,

    fecha date

) AS $$

BEGIN

    RETURN QUERY

    SELECT p.id\_programacion, p.hora, p.fecha

    FROM catalogo\_programacion.programacion p

    WHERE p.id\_programacion = p\_id\_programacion;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_programacion();

--Eliminar

CREATE OR REPLACE FUNCTION eliminar\_programacion(

    id\_programacion int

) RETURNS VOID AS $$

BEGIN

    DELETE

    FROM catalogo\_programacion.programacion

    WHERE id\_programacion = id\_programacion;

END;

$$ LANGUAGE plpgsql;

--Actualizar

CREATE OR REPLACE FUNCTION actualizar\_programacion(

    p\_id\_programacion int,

    p\_id\_pelicula int,

    p\_hora time,

    p\_fecha date

) RETURNS VOID AS $$

BEGIN

    UPDATE catalogo\_programacion.programacion

    SET id\_pelicula = p\_id\_pelicula,

        hora = p\_hora,

        fecha = p\_fecha

    WHERE id\_programacion = p\_id\_programacion;

END;

$$ LANGUAGE plpgsql;

------------------------------------------------------------------------------------------------

--Insertar Espacio Especial

CREATE OR REPLACE FUNCTION insertar\_espacio\_especial(

    nombre varchar(60),

    descripcion varchar(300),

    horario varchar(100)

) RETURNS VOID AS $$

BEGIN

    INSERT INTO catalogo\_programacion.espacio\_especial (nombre, descripcion, horario)

    VALUES (nombre, descripcion, horario);

END;

$$ LANGUAGE plpgsql;

--Mostrar Espacio Especial

CREATE OR REPLACE FUNCTION mostrar\_espacio\_especial()

RETURNS TABLE (

    id\_espacio\_especial int,

    nombre varchar(60),

    descripcion varchar(300),

    horario varchar(100)

) AS $$

BEGIN

    RETURN QUERY

    SELECT e.id\_espacio\_especial, e.nombre, e.descripcion, e.horario

    FROM catalogo\_programacion.espacio\_especial e;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_espacio\_especial();

--Mostrar por filtros

CREATE OR REPLACE FUNCTION mostrar\_espacio\_especial\_por\_id(

    p\_id\_espacio\_especial int

) RETURNS TABLE (

    id\_espacio\_especial int,

    nombre varchar(60),

    descripcion varchar(300),

    horario varchar(100)

) AS $$

BEGIN

    RETURN QUERY

    SELECT e.id\_espacio\_especial, e.nombre, e.descripcion, e.horario

    FROM catalogo\_programacion.espacio\_especial e

    WHERE e.id\_espacio\_especial = p\_id\_espacio\_especial;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_espacio\_especial\_por\_id(2);

--Eliminar espacio especial

CREATE OR REPLACE FUNCTION eliminar\_espacio\_especial(

    id\_espacio\_especial int

) RETURNS VOID AS $$

BEGIN

    DELETE

    FROM catalogo\_programacion.espacio\_especial

    WHERE id\_espacio\_especial = id\_espacio\_especial;

END;

$$ LANGUAGE plpgsql;

--Actualizar

CREATE OR REPLACE FUNCTION actualizar\_espacio\_especial(

    p\_id\_espacio\_especial varchar,

    p\_nombre varchar(60),

    p\_descripcion varchar(300),

    p\_horario varchar(100)

) RETURNS VOID AS $$

BEGIN

    UPDATE catalogo\_programacion.espacio\_especial

    SET nombre = p\_nombre,

        descripcion = p\_descripcion,

        horario = p\_horario

    WHERE id\_espacio\_especial = p\_id\_espacio\_especial;

END;

$$ LANGUAGE plpgsql;

----------------------------------------------------------------------------------------------------------

--Insertar

CREATE OR REPLACE FUNCTION insertar\_espacio\_especial\_pelicula(

    id\_espacio\_especial int,

    id\_pelicula int,

    fecha date

) RETURNS VOID AS $$

BEGIN

    INSERT INTO catalogo\_programacion.espacio\_especial\_pelicula (id\_espacio\_especial, id\_pelicula, fecha)

    VALUES (id\_espacio\_especial, id\_pelicula, fecha);

END;

$$ LANGUAGE plpgsql;

--Mostrar

CREATE OR REPLACE FUNCTION mostrar\_espacio\_especial\_pelicula()

RETURNS TABLE (

    id\_espacio\_especial int,

    id\_pelicula int,

    fecha date

) AS $$

BEGIN

    RETURN QUERY

    SELECT ee.id\_espacio\_especial, ee.pelicula, ee.fecha

    FROM catalogo\_programacion.espacio\_especial\_pelicula ee;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_espacio\_especial();

--Eliminar

CREATE OR REPLACE FUNCTION eliminar\_espacio\_especial\_pelicula(

    p\_id\_espacio\_especial int,

    p\_id\_pelicula int

) RETURNS VOID AS $$

BEGIN

    DELETE FROM catalogo\_programacion.espacio\_especial\_pelicula

    WHERE id\_espacio\_especial = p\_id\_espacio\_especial AND id\_pelicula = p\_id\_pelicula;

END;

$$ LANGUAGE plpgsql;

--Actualizar

CREATE OR REPLACE FUNCTION actualizar\_fecha\_espacio\_especial\_pelicula(

    p\_id\_espacio\_especial int,

    p\_id\_pelicula int,

    p\_fecha date

) RETURNS VOID AS $$

BEGIN

    UPDATE catalogo\_programacion.espacio\_especial\_pelicula

    SET fecha = p\_fecha

    WHERE id\_espacio\_especial = p\_id\_espacio\_especial AND id\_pelicula = p\_id\_pelicula;

END;

$$ LANGUAGE plpgsql;

-------------------------------------------------------------------------------------------------------------------

--Insertar nacionalidad

CREATE OR REPLACE FUNCTION insertar\_nacionalidad(

    nombre\_nacionalidad varchar(150),

    pais varchar(100)

) RETURNS VOID AS $$

BEGIN

    INSERT INTO elenco.nacionalidad (nombre\_nacionalidad, pais)

    VALUES (nombre\_nacionalidad, pais);

END;

$$ LANGUAGE plpgsql;

--Mostrar Nacionalidad

CREATE OR REPLACE FUNCTION mostrar\_nacionalidad()

RETURNS TABLE (

    id\_nacionalidad int,

    nombre\_nacionalidad varchar(150),

    pais varchar(100)

) AS $$

BEGIN

    RETURN QUERY

    SELECT n.id\_nacionalidad, n.nombre\_nacionalidad, n.pais

    FROM elenco.nacionalidad n;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_nacionalidad();

--Mostrar una nacionalidad

CREATE OR REPLACE FUNCTION mostrar\_nacionalidad\_por\_id(

    p\_id\_nacionalidad int

) RETURNS TABLE (

    id\_nacionalidad int,

    nombre\_nacionalidad varchar(150),

    pais varchar(100)

) AS $$

BEGIN

    RETURN QUERY

    SELECT n.id\_nacionalidad, n.nombre\_nacionalidad, n.pais

    FROM elenco.nacionalidad n

    WHERE n.id\_nacionalidad = p\_id\_nacionalidad;

END;

$$ LANGUAGE plpgsql;

SELECT  mostrar\_nacionalidad\_por\_id(2);

--Eliminar

CREATE OR REPLACE FUNCTION eliminar\_nacionalidad(

    p\_id\_nacionalidad int

) RETURNS VOID AS $$

BEGIN

    DELETE

    FROM elenco.nacionalidad

    WHERE id\_nacionalidad = p\_id\_nacionalidad;

END;

$$ LANGUAGE plpgsql;

--Actualizar

CREATE OR REPLACE FUNCTION actualizar\_nacionalidad(

    p\_id\_nacionalidad int,

    p\_nombre\_nacionalidad varchar(150),

    p\_pais varchar(100)

) RETURNS VOID AS $$

BEGIN

    UPDATE elenco.nacionalidad

    SET nombre\_nacionalidad = p\_nombre\_nacionalidad,

    pais = p\_pais

    WHERE id\_nacionalidad = p\_id\_nacionalidad;

END;

$$ LANGUAGE plpgsql;

-------------------------------------------------------------------------------------------------------------

--Insertar profesion

CREATE OR REPLACE FUNCTION insertar\_profesion(

    nombre\_profesion varchar(15)

) RETURNS VOID AS $$

BEGIN

    INSERT INTO elenco.profesion (nombre\_profesion)

    VALUES (nombre\_profesion);

END;

$$ LANGUAGE plpgsql;

--Mostrar Profesion

CREATE OR REPLACE FUNCTION mostrar\_profesion()

RETURNS TABLE (

    id\_profesion int,

    nombre\_profesion varchar(15)

) AS $$

BEGIN

    RETURN QUERY

    SELECT p.id\_profesion, p.nombre\_profesion

    FROM elenco.profesion p;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_profesion();

--Mostrar Profesion

CREATE OR REPLACE FUNCTION mostrar\_profesion\_por\_id(

    p\_id\_profesion int

) RETURNS TABLE (

    id\_profesion int,

    nombre\_profesion varchar(15)

) AS $$

BEGIN

    RETURN QUERY

    SELECT p.id\_profesion, p.nombre\_profesion

    FROM elenco.profesion p

    WHERE p.id\_profesion = p\_id\_profesion;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_profesion\_por\_id(2);

--Eliminar

CREATE OR REPLACE FUNCTION eliminar\_profesion(

    p\_id\_profesion int

) RETURNS VOID AS $$

BEGIN

    DELETE

    FROM elenco.profesion

    WHERE id\_profesion = p\_id\_profesion;

END;

$$ LANGUAGE plpgsql;

--Actualizar

CREATE OR REPLACE FUNCTION actualizar\_profesion(

    p\_id\_profesion int,

    p\_nombre\_profesion varchar(15)

) RETURNS VOID AS $$

BEGIN

    UPDATE elenco.profesion

    SET nombre\_profesion = p\_nombre\_profesion

    WHERE id\_profesion = p\_id\_profesion;

END;

$$ LANGUAGE plpgsql;

---------------------------------------------------------------------------------------------------

--Insertar artista

CREATE OR REPLACE FUNCTION insertar\_artista(

    id\_nacionalidad int,

    id\_profesion int,

    manager varchar(50),

    nombres varchar(50),

    apellidos varchar(50),

    biografia varchar(500),

    fecha\_nacimiento date

) RETURNS VOID AS $$

BEGIN

    INSERT INTO elenco.artista (id\_nacionalidad, id\_profesion, manager, nombres, apellidos, biografia, fecha\_nacimiento)

    VALUES (id\_nacionalidad, id\_profesion, manager, nombres, apellidos, biografia, fecha\_nacimiento);

END;

$$ LANGUAGE plpgsql;

--Mostrar artista

CREATE OR REPLACE FUNCTION mostrar\_artista()

RETURNS TABLE (

    id\_artista int,

    id\_nacionalidad int,

    id\_profesion int,

    manager varchar(50),

    nombres varchar(50),

    apellidos varchar(50),

    biografia varchar(500),

    fecha\_nacimiento date

) AS $$

BEGIN

    RETURN QUERY

    SELECT a.id\_artista, a.id\_nacionalidad, a.id\_profesion, a.manager, a.nombres, a.apellidos,

           a.biografia, a.fecha\_nacimiento

    FROM elenco.artista a;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_profesion();

--Mostrar una nacionalidad

CREATE OR REPLACE FUNCTION mostrar\_artista\_por\_id(

    p\_id\_artista int

) RETURNS TABLE (

    id\_artista int,

    id\_nacionalidad int,

    id\_profesion int,

    manager varchar(50),

    nombres varchar(50),

    apellidos varchar(50),

    biografia varchar(500),

    fecha\_nacimiento date

) AS $$

BEGIN

    RETURN QUERY

    SELECT a.id\_artista, a.id\_nacionalidad, a.id\_profesion, a.manager, a.nombres, a.apellidos,

           a.biografia, a.fecha\_nacimiento

    FROM elenco.artista a

    WHERE a.id\_artista = p\_id\_artista;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_artista\_por\_id(2);

--Eliminar

CREATE OR REPLACE FUNCTION eliminar\_artista(

    p\_id\_artista int

) RETURNS VOID AS $$

BEGIN

    DELETE

    FROM elenco.artista

    WHERE id\_artista = p\_id\_artista;

END;

$$ LANGUAGE plpgsql;

--Actualizar artista

CREATE OR REPLACE FUNCTION actualizar\_artista(

    p\_id\_artista int,

    p\_id\_nacionalidad int,

    p\_id\_profesion int,

    p\_manager varchar(50),

    p\_nombres varchar(50),

    p\_apellidos varchar(50),

    p\_biografia varchar(500),

    p\_fecha\_nacimiento date

) RETURNS VOID AS $$

BEGIN

    UPDATE elenco.artista

    SET id\_nacionalidad = p\_id\_nacionalidad,

        id\_profesion = p\_id\_profesion,

        manager = p\_manager,

        nombres = p\_nombres,

        apellidos = p\_apellidos,

        biografia = p\_biografia,

        fecha\_nacimiento = p\_fecha\_nacimiento

    WHERE id\_artista = p\_id\_artista;

END;

$$ LANGUAGE plpgsql;

------------------------------------------------------------------------------------------------

--Insertar personaje

CREATE OR REPLACE FUNCTION insertar\_personaje(

    id\_artista int,

    id\_pelicula int,

    nombre\_personaje varchar(100),

    personaje\_principal boolean DEFAULT FALSE

) RETURNS VOID AS $$

BEGIN

    INSERT INTO elenco.personaje (id\_artista, id\_pelicula, nombre\_personaje, personaje\_principal)

    VALUES (id\_artista, id\_pelicula, nombre\_personaje, personaje\_principal);

END;

$$ LANGUAGE plpgsql;

--Mostrar Personajes

CREATE OR REPLACE FUNCTION mostrar\_personaje()

RETURNS TABLE (

    id\_personaje int,

    id\_artista int,

    id\_pelicula int,

    nombre\_personaje varchar(100),

    personaje\_principal boolean

) AS $$

BEGIN

    RETURN QUERY

    SELECT p.id\_personaje, p.id\_artista, p.id\_pelicula, p.nombre\_personaje, p.personaje\_principal

    FROM elenco.personaje p;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_personaje();

--Mostrar un personaje

CREATE OR REPLACE FUNCTION mostrar\_personaje\_por\_id(

    p\_id\_personaje int

) RETURNS TABLE (

    id\_personaje int,

    id\_artista int,

    id\_pelicula int,

    nombre\_personaje varchar(100),

    personaje\_principal boolean

) AS $$

BEGIN

    RETURN QUERY

    SELECT p.id\_personaje, p.id\_artista, p.id\_pelicula, p.nombre\_personaje, p.personaje\_principal

    FROM elenco.personaje p

    WHERE p.id\_personaje = p\_id\_personaje;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_personaje\_por\_id(2);

--Eliminar personaje

CREATE OR REPLACE FUNCTION eliminar\_personaje(

    p\_id\_personaje int

) RETURNS VOID AS $$

BEGIN

    DELETE

    FROM elenco.personaje

    WHERE id\_personaje = p\_id\_personaje;

END;

$$ LANGUAGE plpgsql;

--Actualizar personaje

CREATE OR REPLACE FUNCTION actualizar\_personaje(

    p\_id\_personaje int,

    p\_id\_artista int,

    p\_id\_pelicula int,

    p\_nombre\_personaje varchar(100),

    p\_personaje\_principal boolean

) RETURNS VOID AS $$

BEGIN

    UPDATE elenco.personaje

    SET id\_artista = p\_id\_artista,

        id\_pelicula = p\_id\_pelicula,

        nombre\_personaje = p\_nombre\_personaje,

        personaje\_principal = p\_personaje\_principal

    WHERE id\_personaje = p\_id\_personaje;

END;

$$ LANGUAGE plpgsql;

---------------------------------------------------------------------------------------------------------------

--Insertar

CREATE OR REPLACE FUNCTION insertar\_sitie(

    id\_artista int,

    fan\_site\_url varchar(150),

    personal\_site\_url varchar(150)

) RETURNS VOID AS $$

BEGIN

    INSERT INTO elenco.sitie (id\_artista, fan\_site\_url, personal\_site\_url)

    VALUES (id\_artista, fan\_site\_url, personal\_site\_url);

END;

$$ LANGUAGE plpgsql;

--Mostrar

CREATE OR REPLACE FUNCTION mostrar\_sitie()

RETURNS TABLE (

    id\_artista int,

    fan\_site\_url varchar(150),

    personal\_site\_url varchar(150)

) AS $$

BEGIN

    RETURN QUERY

    SELECT s.id\_artista, s.fan\_site\_url, s.personal\_site\_url

    FROM elenco.sitie s;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_sitie();

--Mostrar

CREATE OR REPLACE FUNCTION mostrar\_sitie\_artista(

    p\_id\_artista int

) RETURNS TABLE (

    id\_artista int,

    fan\_site\_url varchar(150),

    personal\_site\_url varchar(150)

) AS $$

BEGIN

    RETURN QUERY

    SELECT s.id\_artista, s.fan\_site\_url, s.personal\_site\_url

    FROM elenco.sitie s

    WHERE s.id\_artista = p\_id\_artista;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_sitie\_artista(16);

--Eliminar

CREATE OR REPLACE FUNCTION eliminar\_sitie(

    p\_id\_artista int

) RETURNS VOID AS $$

BEGIN

    DELETE

    FROM elenco.sitie

    WHERE id\_artista = p\_id\_artista;

END;

$$ LANGUAGE plpgsql;

--Actualizar

CREATE OR REPLACE FUNCTION actualizar\_sitie(

    p\_id\_artista int,

    p\_fan\_site\_url varchar(150),

    p\_personal\_site\_url varchar(150)

) RETURNS VOID AS $$

BEGIN

    UPDATE elenco.sitie

    SET fan\_site\_url = p\_fan\_site\_url,

    personal\_site\_url = p\_personal\_site\_url

    WHERE id\_artista = p\_id\_artista;

END;

$$ LANGUAGE plpgsql;

----------------------------------------------------------------------------------------------------------------

--Insertar Tipo Cargo

CREATE OR REPLACE FUNCTION insertar\_tipo\_cargo(

    nombre\_tipo\_cargo varchar(50)

) RETURNS VOID AS $$

BEGIN

    INSERT INTO elenco.tipo\_cargo (nombre\_tipo\_cargo)

    VALUES (nombre\_tipo\_cargo);

END;

$$ LANGUAGE plpgsql;

--Mostrar TipoCargo

CREATE OR REPLACE FUNCTION mostrar\_tipo\_cargo()

RETURNS TABLE (

    id\_tipo\_cargo int,

    nombre\_tipo\_cargo varchar(50)

) AS $$

BEGIN

    RETURN QUERY

    SELECT tc.id\_tipo\_cargo, tc.nombre\_tipo\_cargo

    FROM elenco.tipo\_cargo tc;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_tipo\_cargo();

--Mostrar un personaje

CREATE OR REPLACE FUNCTION mostrar\_cargo\_por\_id(

    p\_id\_tipo\_cargo int

) RETURNS TABLE (

    id\_tipo\_cargo int,

    nombre\_tipo\_cargo varchar(50)

) AS $$

BEGIN

    RETURN QUERY

    SELECT tc.id\_tipo\_cargo, tc.nombre\_tipo\_cargo

    FROM elenco.tipo\_cargo tc

    WHERE tc.id\_tipo\_cargo = p\_id\_tipo\_cargo;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_cargo\_por\_id(1);

--Eliminar

CREATE OR REPLACE FUNCTION eliminar\_tipo\_cargo(

    p\_id\_tipo\_cargo int

) RETURNS VOID AS $$

BEGIN

    DELETE

    FROM elenco.tipo\_cargo

    WHERE id\_tipo\_cargo = p\_id\_tipo\_cargo;

END;

$$ LANGUAGE plpgsql;

--Actualizar

CREATE OR REPLACE FUNCTION actualizar\_cargo(

    p\_id\_tipo\_cargo int,

    p\_nombre\_tipo\_cargo varchar(50)

) RETURNS VOID AS $$

BEGIN

    UPDATE elenco.tipo\_cargo

    SET nombre\_tipo\_cargo = p\_nombre\_tipo\_cargo

    WHERE id\_tipo\_cargo = p\_id\_tipo\_cargo;

END;

$$ LANGUAGE plpgsql;

-------------------------------------------------------------------------------------------------------

--Insertar

CREATE OR REPLACE FUNCTION insertar\_pelicula\_artista\_cargo(

    id\_pelicula int,

    id\_artista int,

    id\_tipo\_cargo int

) RETURNS VOID AS $$

BEGIN

    INSERT INTO elenco.pelicula\_artista\_cargo (id\_pelicula, id\_artista, id\_tipo\_cargo)

    VALUES (id\_pelicula, id\_artista, id\_tipo\_cargo);

END;

$$ LANGUAGE plpgsql;

--Mostrar

CREATE OR REPLACE FUNCTION mostrar\_pelicula\_artista\_cargo()

RETURNS TABLE (

    id\_pelicula int,

    id\_artista int,

    id\_tipo\_cargo int

) AS $$

BEGIN

    RETURN QUERY

    SELECT pac.id\_pelicula, pac.id\_artista, pac.id\_tipo\_cargo

    FROM elenco.pelicula\_artista\_cargo pac;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_pelicula\_artista\_cargo();

--Mostrar

CREATE OR REPLACE FUNCTION mostrar\_pelicula\_artista\_cargo\_id(

    p\_id\_artista int

) RETURNS TABLE (

    id\_pelicula int,

    id\_artista int,

    id\_tipo\_cargo int

) AS $$

BEGIN

    RETURN QUERY

    SELECT pac.id\_pelicula, pac.id\_artista, pac.id\_tipo\_cargo

    FROM elenco.pelicula\_artista\_cargo pac

    WHERE pac.id\_artista = p\_id\_artista;

END;

$$ LANGUAGE plpgsql;

SELECT mostrar\_pelicula\_artista\_cargo\_id(2);

--Eliminar

CREATE OR REPLACE FUNCTION eliminar\_pelicula\_artista\_cargo(

    p\_id\_pelicula int,

    p\_id\_artista int,

    p\_id\_tipo\_cargo int

) RETURNS VOID AS $$

BEGIN

    DELETE FROM elenco.pelicula\_artista\_cargo

    WHERE id\_pelicula = p\_id\_pelicula AND id\_artista = p\_id\_artista AND id\_tipo\_cargo = p\_id\_tipo\_cargo;

END;

$$ LANGUAGE plpgsql;

--Actualizar

CREATE OR REPLACE FUNCTION actualizar\_pelicula\_artista\_cargo(

    p\_id\_pelicula int,

    p\_id\_artista int,

    p\_id\_tipo\_cargo int

) RETURNS VOID AS $$

BEGIN

    UPDATE elenco.pelicula\_artista\_cargo

    SET id\_pelicula = p\_id\_pelicula,

        id\_artista = p\_id\_artista,

        id\_tipo\_cargo = p\_id\_tipo\_cargo

    WHERE id\_pelicula = p\_id\_pelicula

        AND id\_artista = p\_id\_artista

        AND id\_tipo\_cargo = p\_id\_tipo\_cargo;

END;

$$ LANGUAGE plpgsql;

## Funciones almacenadas Adicionales

-- Sp para mostrar la cartelera de peliculas

CREATE OR REPLACE FUNCTION obtener\_cartelera()

RETURNS TABLE (

    fecha\_programacion date,

    hora\_programacion time,

    nombre\_pelicula varchar(100),

    nombre\_genero varchar(50),

    nombre\_clasificacion varchar(50),

    anio\_relace date,

    duracion time,

    descripcion\_pelicula varchar(300),

    nombres\_director varchar(100)

) AS $$

DECLARE

    id\_director int;

BEGIN

    SELECT id\_tipo\_cargo INTO id\_director

    FROM elenco.tipo\_cargo

    WHERE nombre\_tipo\_cargo = 'Director';

    RETURN QUERY

    SELECT

        cp.fecha AS fecha\_programacion,

        cp.hora AS hora\_programacion,

        cpp.nombre\_pelicula,

        cpg.nombre\_genero,

        cpc.nombre\_clasificacion,

        cpp.anio\_relace,

        cpp.duracion,

        cpp.descripcion\_pelicula,

        ea.nombres AS nombres\_director

    FROM

        catalogo\_programacion.pelicula cpp JOIN catalogo\_programacion.programacion cp

            ON cpp.id\_pelicula = cp.id\_pelicula

            LEFT JOIN catalogo\_programacion.genero cpg

            ON cpp.id\_genero = cpg.id\_genero

            LEFT JOIN catalogo\_programacion.clasificacion cpc

            ON cpp.id\_clasificacion = cpc.id\_clasificacion

            LEFT JOIN elenco.pelicula\_artista\_cargo epac

            ON cpp.id\_pelicula = epac.id\_pelicula

            LEFT JOIN elenco.artista ea

            ON epac.id\_artista = ea.id\_artista

    WHERE

        epac.id\_tipo\_cargo = id\_director;

END;

$$ LANGUAGE plpgsql;

SELECT \* FROM obtener\_cartelera();

-- Sp para mostrar la cartelera, pero para una fecha determinada

CREATE OR REPLACE FUNCTION obtener\_cartelera\_fecha(p\_fecha\_programacion date)

RETURNS TABLE (

    fecha\_programacion date,

    hora\_programacion time,

    nombre\_pelicula varchar(100),

    nombre\_genero varchar(50),

    nombre\_clasificacion varchar(50),

    anio\_relace date,

    duracion time,

    descripcion\_pelicula varchar(300),

    nombres\_director varchar(100)

) AS $$

DECLARE

    id\_director int;

BEGIN

    SELECT id\_tipo\_cargo INTO id\_director

    FROM elenco.tipo\_cargo

    WHERE nombre\_tipo\_cargo = 'Director';

    RETURN QUERY

    SELECT

        cp.fecha AS fecha\_programacion,

        cp.hora AS hora\_programacion,

        cpp.nombre\_pelicula,

        cpg.nombre\_genero,

        cpc.nombre\_clasificacion,

        cpp.anio\_relace,

        cpp.duracion,

        cpp.descripcion\_pelicula,

        ea.nombres AS nombres\_director

    FROM

        catalogo\_programacion.pelicula cpp JOIN catalogo\_programacion.programacion cp

            ON cpp.id\_pelicula = cp.id\_pelicula

            LEFT JOIN catalogo\_programacion.genero cpg

            ON cpp.id\_genero = cpg.id\_genero

            LEFT JOIN catalogo\_programacion.clasificacion cpc

            ON cpp.id\_clasificacion = cpc.id\_clasificacion

            LEFT JOIN elenco.pelicula\_artista\_cargo epac

            ON cpp.id\_pelicula = epac.id\_pelicula

            LEFT JOIN elenco.artista ea

            ON epac.id\_artista = ea.id\_artista

    WHERE

        epac.id\_tipo\_cargo = id\_director AND cp.fecha = p\_fecha\_programacion;

END;

$$ LANGUAGE plpgsql;

SELECT obtener\_cartelera\_fecha('2003-03-03');

-- Sp para mostrar la cartelera por genero por un determinado mes

CREATE OR REPLACE FUNCTION obtener\_cartelera\_mes(p\_fecha\_programacion date, p\_nombre\_genero varchar(50))

RETURNS TABLE (

    fecha\_programacion date,

    hora\_programacion time,

    nombre\_pelicula varchar(100),

    nombre\_genero varchar(50),

    nombre\_clasificacion varchar(50),

    anio\_relace date,

    duracion time,

    descripcion\_pelicula varchar(300),

    nombres\_director varchar(100)

) AS $$

DECLARE

    id\_director int;

BEGIN

    SELECT id\_tipo\_cargo INTO id\_director

    FROM elenco.tipo\_cargo

    WHERE nombre\_tipo\_cargo = 'Director';

    RETURN QUERY

    SELECT

        cp.fecha AS fecha\_programacion,

        cp.hora AS hora\_programacion,

        cpp.nombre\_pelicula,

        cpg.nombre\_genero,

        cpc.nombre\_clasificacion,

        cpp.anio\_relace,

        cpp.duracion,

        cpp.descripcion\_pelicula,

        ea.nombres AS nombres\_director

    FROM

        catalogo\_programacion.pelicula cpp JOIN catalogo\_programacion.programacion cp

            ON cpp.id\_pelicula = cp.id\_pelicula

            LEFT JOIN catalogo\_programacion.genero cpg

            ON cpp.id\_genero = cpg.id\_genero

            LEFT JOIN catalogo\_programacion.clasificacion cpc

            ON cpp.id\_clasificacion = cpc.id\_clasificacion

            LEFT JOIN elenco.pelicula\_artista\_cargo epac

            ON cpp.id\_pelicula = epac.id\_pelicula

            LEFT JOIN elenco.artista ea

            ON epac.id\_artista = ea.id\_artista

    WHERE

        epac.id\_tipo\_cargo = id\_director AND cp.fecha >= p\_fecha\_programacion

        AND cp.fecha < p\_fecha\_programacion + INTERVAL '1 month'

        AND cpg.nombre\_genero = p\_nombre\_genero;

END;

$$ LANGUAGE plpgsql;

SELECT obtener\_cartelera\_mes('2003-03-03','Comedia');

-- Sp para mostrar la cartelera por clasificación por un determinado mes

CREATE OR REPLACE FUNCTION obtener\_cartelera\_mes\_clasificacion(p\_fecha\_programacion date, p\_nombre\_clasificacion varchar(50))

RETURNS TABLE (

    fecha\_programacion date,

    hora\_programacion time,

    nombre\_pelicula varchar(100),

    nombre\_genero varchar(50),

    nombre\_clasificacion varchar(50),

    anio\_relace date,

    duracion time,

    descripcion\_pelicula varchar(300),

    nombres\_director varchar(100)

) AS $$

DECLARE

    id\_director int;

BEGIN

    SELECT id\_tipo\_cargo INTO id\_director

    FROM elenco.tipo\_cargo

    WHERE nombre\_tipo\_cargo = 'Director';

    RETURN QUERY

    SELECT

        cp.fecha AS fecha\_programacion,

        cp.hora AS hora\_programacion,

        cpp.nombre\_pelicula,

        cpg.nombre\_genero,

        cpc.nombre\_clasificacion,

        cpp.anio\_relace,

        cpp.duracion,

        cpp.descripcion\_pelicula,

        ea.nombres AS nombres\_director

    FROM

        catalogo\_programacion.pelicula cpp JOIN catalogo\_programacion.programacion cp

            ON cpp.id\_pelicula = cp.id\_pelicula

            LEFT JOIN catalogo\_programacion.genero cpg

            ON cpp.id\_genero = cpg.id\_genero

            LEFT JOIN catalogo\_programacion.clasificacion cpc

            ON cpp.id\_clasificacion = cpc.id\_clasificacion

            LEFT JOIN elenco.pelicula\_artista\_cargo epac

            ON cpp.id\_pelicula = epac.id\_pelicula

            LEFT JOIN elenco.artista ea

            ON epac.id\_artista = ea.id\_artista

    WHERE

        epac.id\_tipo\_cargo = id\_director AND cp.fecha >= p\_fecha\_programacion

        AND cp.fecha < p\_fecha\_programacion + INTERVAL '1 month'

        AND cpc.nombre\_clasificacion = p\_nombre\_clasificacion;

END;

$$ LANGUAGE plpgsql;

SELECT obtener\_cartelera\_mes\_clasificacion('2003-03-03','PG-13');

-- Sp para mostrar la cartelera por director

CREATE OR REPLACE FUNCTION obtener\_cartelera\_director(p\_nombres\_director varchar(100))

RETURNS TABLE (

    fecha\_programacion date,

    hora\_programacion time,

    nombre\_pelicula varchar(100),

    nombre\_genero varchar(50),

    nombre\_clasificacion varchar(50),

    anio\_relace date,

    duracion time,

    descripcion\_pelicula varchar(300),

    nombres\_director varchar(100)

) AS $$

DECLARE

    id\_director int;

BEGIN

    SELECT id\_tipo\_cargo INTO id\_director

    FROM elenco.tipo\_cargo

    WHERE nombre\_tipo\_cargo = 'Director';

    RETURN QUERY

    SELECT

        cp.fecha AS fecha\_programacion,

        cp.hora AS hora\_programacion,

        cpp.nombre\_pelicula,

        cpg.nombre\_genero,

        cpc.nombre\_clasificacion,

        cpp.anio\_relace,

        cpp.duracion,

        cpp.descripcion\_pelicula,

        ea.nombres AS nombres\_director

    FROM

        catalogo\_programacion.pelicula cpp JOIN catalogo\_programacion.programacion cp

            ON cpp.id\_pelicula = cp.id\_pelicula

            LEFT JOIN catalogo\_programacion.genero cpg

            ON cpp.id\_genero = cpg.id\_genero

            LEFT JOIN catalogo\_programacion.clasificacion cpc

            ON cpp.id\_clasificacion = cpc.id\_clasificacion

            LEFT JOIN elenco.pelicula\_artista\_cargo epac

            ON cpp.id\_pelicula = epac.id\_pelicula

            LEFT JOIN elenco.artista ea

            ON epac.id\_artista = ea.id\_artista

    WHERE

        epac.id\_tipo\_cargo = id\_director AND ea.nombres = p\_nombres\_director;

END;

$$ LANGUAGE plpgsql;

SELECT obtener\_cartelera\_director('Christopher');

-- Sp para mostrar la cartelera de espacios especiales

CREATE OR REPLACE FUNCTION obtener\_cartelera\_espacios\_especiales()

RETURNS TABLE (

    fecha\_espacio\_especial\_pelicula date,

    nombre\_pelicula varchar(100),

    nombres\_director varchar(100),

    descripcion\_pelicula varchar(300),

    nombre\_espacio\_especial varchar(100)

) AS $$

DECLARE

    id\_director int;

BEGIN

    SELECT id\_tipo\_cargo INTO id\_director

    FROM elenco.tipo\_cargo

    WHERE nombre\_tipo\_cargo = 'Director';

    RETURN QUERY

    SELECT

        cpep.fecha AS fecha\_espacio\_especial\_pelicula,

        cpp.nombre\_pelicula,

        ea.nombres AS nombres\_director,

        cpp.descripcion\_pelicula,

        cpee.nombre

    FROM

        catalogo\_programacion.espacio\_especial cpee JOIN catalogo\_programacion.espacio\_especial\_pelicula cpep

            ON cpee.id\_espacio\_especial = cpep.id\_espacio\_especial

            JOIN catalogo\_programacion.pelicula cpp

            ON cpep.id\_pelicula = cpp.id\_pelicula

            LEFT JOIN elenco.pelicula\_artista\_cargo epac

            ON cpp.id\_pelicula = epac.id\_pelicula

            LEFT JOIN elenco.artista ea

            ON epac.id\_artista = ea.id\_artista

    WHERE

        epac.id\_tipo\_cargo = id\_director;

END;

$$ LANGUAGE plpgsql;

SELECT obtener\_cartelera\_espacios\_especiales();

-- Sp para mostrar la cartelera de espacios especiales por fecha

CREATE OR REPLACE FUNCTION obtener\_cartelera\_espacios\_especiales\_fecha(fecha\_espacio\_especial date)

RETURNS TABLE (

    fecha\_espacio\_especial\_pelicula date,

    nombre\_pelicula varchar(100),

    nombres\_director varchar(100),

    descripcion\_pelicula varchar(300),

    nombre\_espacio\_especial varchar(100)

) AS $$

DECLARE

    id\_director int;

BEGIN

    SELECT id\_tipo\_cargo INTO id\_director

    FROM elenco.tipo\_cargo

    WHERE nombre\_tipo\_cargo = 'Director';

    RETURN QUERY

    SELECT

        cpep.fecha AS fecha\_espacio\_especial\_pelicula,

        cpp.nombre\_pelicula,

        ea.nombres AS nombres\_director,

        cpp.descripcion\_pelicula,

        cpee.nombre

    FROM

        catalogo\_programacion.espacio\_especial cpee JOIN catalogo\_programacion.espacio\_especial\_pelicula cpep

            ON cpee.id\_espacio\_especial = cpep.id\_espacio\_especial

            JOIN catalogo\_programacion.pelicula cpp

            ON cpep.id\_pelicula = cpp.id\_pelicula

            LEFT JOIN elenco.pelicula\_artista\_cargo epac

            ON cpp.id\_pelicula = epac.id\_pelicula

            LEFT JOIN elenco.artista ea ON epac.id\_artista = ea.id\_artista

    WHERE

        epac.id\_tipo\_cargo = id\_director AND cpep.fecha = fecha\_espacio\_especial;

END;

$$ LANGUAGE plpgsql;

SELECT obtener\_cartelera\_espacios\_especiales\_fecha('2024-04-19');

-- Sp para mostrar la cartelera de espacios especiales por el espacio y mes

CREATE OR REPLACE FUNCTION obtener\_cartelera\_espacios\_especiales\_espacio\_mes(fecha\_mes date, nombre\_espacio\_especial\_mostrar varchar(100))

RETURNS TABLE (

    fecha\_espacio\_especial\_pelicula date,

    nombre\_pelicula varchar(100),

    nombres\_director varchar(100),

    descripcion\_pelicula varchar(300),

    nombre\_espacio\_especial varchar(100)

) AS $$

DECLARE

    id\_director int;

BEGIN

    SELECT id\_tipo\_cargo INTO id\_director

    FROM elenco.tipo\_cargo

    WHERE nombre\_tipo\_cargo = 'Director';

    RETURN QUERY

    SELECT

        cpep.fecha AS fecha\_espacio\_especial\_pelicula,

        cpp.nombre\_pelicula,

        ea.nombres AS nombres\_director,

        cpp.descripcion\_pelicula,

        cpee.nombre

    FROM

        catalogo\_programacion.espacio\_especial cpee JOIN catalogo\_programacion.espacio\_especial\_pelicula cpep

            ON cpee.id\_espacio\_especial = cpep.id\_espacio\_especial

            JOIN catalogo\_programacion.pelicula cpp

            ON cpep.id\_pelicula = cpp.id\_pelicula

            LEFT JOIN elenco.pelicula\_artista\_cargo epac

            ON cpp.id\_pelicula = epac.id\_pelicula

            LEFT JOIN elenco.artista ea

            ON epac.id\_artista = ea.id\_artista

    WHERE

        epac.id\_tipo\_cargo = id\_director AND cpep.fecha >= fecha\_mes

        AND cpep.fecha < fecha\_mes + INTERVAL '1 month'

        AND cpee.nombre = nombre\_espacio\_especial\_mostrar;

END;

$$ LANGUAGE plpgsql;

SELECT obtener\_cartelera\_espacios\_especiales\_espacio\_mes('2024-04-19','Adrenalina al máximo');

-- Sp para mostrar la ficha bibliográfica de un artista

CREATE OR REPLACE FUNCTION obtener\_ficha\_bibliografica\_artista(id\_artista\_mostrar int)

RETURNS TABLE (

    nombres varchar(50),

    apellidos varchar(50),

    nombre\_nacionalidad varchar(150),

    pais varchar(100),

    nombre\_profesion varchar(15),

    manager varchar(50),

    biografia varchar(500),

    fan\_site\_url varchar(150),

    personal\_site\_url varchar(150),

    fecha\_nacimiento date,

    director\_en text,

    guionista\_en text

) AS $$

DECLARE

    id\_director int;

    id\_guionista int;

BEGIN

    SELECT id\_tipo\_cargo INTO id\_director

    FROM elenco.tipo\_cargo

    WHERE nombre\_tipo\_cargo = 'Director';

    SELECT id\_tipo\_cargo INTO id\_guionista

    FROM elenco.tipo\_cargo

    WHERE nombre\_tipo\_cargo = 'Guionista';

    RETURN QUERY

    SELECT

        ea.nombres,

        ea.apellidos,

        en.nombre\_nacionalidad,

        en.pais,

        ep.nombre\_profesion,

        ea.manager,

        ea.biografia,

        es.fan\_site\_url,

        es.personal\_site\_url,

        ea.fecha\_nacimiento,

        (

            SELECT STRING\_AGG(cpp.nombre\_pelicula, ', ')

            FROM elenco.artista ea\_director

            INNER JOIN elenco.pelicula\_artista\_cargo epac\_director ON ea\_director.id\_artista = epac\_director.id\_artista

            INNER JOIN catalogo\_programacion.pelicula cpp ON epac\_director.id\_pelicula = cpp.id\_pelicula

            WHERE epac\_director.id\_tipo\_cargo = id\_director AND ea\_director.id\_artista = id\_artista\_mostrar

        ) AS director\_en,

        (

            SELECT STRING\_AGG(cpp.nombre\_pelicula, ', ')

            FROM elenco.artista ea\_director

            INNER JOIN elenco.pelicula\_artista\_cargo epac\_director ON ea\_director.id\_artista = epac\_director.id\_artista

            INNER JOIN catalogo\_programacion.pelicula cpp ON epac\_director.id\_pelicula = cpp.id\_pelicula

            WHERE epac\_director.id\_tipo\_cargo = id\_guionista AND ea\_director.id\_artista = id\_artista\_mostrar

        ) AS guionista\_en

    FROM

        elenco.artista ea JOIN elenco.nacionalidad en ON ea.id\_nacionalidad = en.id\_nacionalidad

            JOIN elenco.profesion ep

            ON ea.id\_profesion = ep.id\_profesion

            LEFT JOIN elenco.sitie es

            ON ea.id\_artista = es.id\_artista

    WHERE

        ea.id\_artista = id\_artista\_mostrar;

END;

$$ LANGUAGE plpgsql;

SELECT \* FROM obtener\_ficha\_bibliografica\_artista(1);

-- Sp para mostrar los artistas de una determinada nacionalidad

CREATE OR REPLACE FUNCTION obtener\_artistas\_nacionalidad(nombre\_nacionalidad\_artista varchar(150))

RETURNS TABLE (

    nombres varchar(50),

    apellidos varchar(50),

    nombre\_nacionalidad varchar(150),

    pais varchar(100),

    nombre\_profesion varchar(15),

    manager varchar(50),

    biografia varchar(500),

    fan\_site\_url varchar(150),

    personal\_site\_url varchar(150),

    fecha\_nacimiento date

) AS $$

BEGIN

    RETURN QUERY

    SELECT

        ea.nombres,

        ea.apellidos,

        en.nombre\_nacionalidad,

        en.pais,

        ep.nombre\_profesion,

        ea.manager,

        ea.biografia,

        es.fan\_site\_url,

        es.personal\_site\_url,

        ea.fecha\_nacimiento

    FROM

        elenco.artista ea JOIN elenco.nacionalidad en

            ON ea.id\_nacionalidad = en.id\_nacionalidad

            JOIN elenco.profesion ep

            ON ea.id\_profesion = ep.id\_profesion

            LEFT JOIN elenco.sitie es

            ON ea.id\_artista = es.id\_artista

    WHERE

        en.nombre\_nacionalidad = nombre\_nacionalidad\_artista;

END;

$$ LANGUAGE plpgsql;

SELECT obtener\_artistas\_nacionalidad('Albanesa');

-- Sp que muestre la lista de las películas que ha sido actor el artista

CREATE OR REPLACE FUNCTION obtener\_peliculas\_artista(id\_artista\_mostrar int)

RETURNS TABLE (

    nombre\_pelicula varchar(100),

    nombre\_genero varchar(50),

    nombre\_clasificacion varchar(50),

    anio\_relace date,

    duracion time,

    descripcion\_pelicula varchar(300),

    nombre\_personaje varchar(100),

    personaje\_principal boolean

) AS $$

BEGIN

    RETURN QUERY

    SELECT

        cpp.nombre\_pelicula,

        cpg.nombre\_genero,

        cpc.nombre\_clasificacion,

        cpp.anio\_relace,

        cpp.duracion,

        cpp.descripcion\_pelicula,

        ep.nombre\_personaje,

        ep.personaje\_principal

    FROM

        elenco.personaje ep JOIN catalogo\_programacion.pelicula cpp

            ON ep.id\_pelicula = cpp.id\_pelicula

            LEFT JOIN catalogo\_programacion.genero cpg

            ON cpp.id\_genero = cpg.id\_genero

            LEFT JOIN catalogo\_programacion.clasificacion cpc

            ON cpp.id\_clasificacion = cpc.id\_clasificacion

    WHERE

        ep.id\_artista = id\_artista\_mostrar;

END;

$$ LANGUAGE plpgsql;

SELECT obtener\_peliculas\_artista(2);

-- Sp para mostrar la ficha de la película

CREATE OR REPLACE FUNCTION obtener\_ficha\_pelicula(id\_pelicula\_mostrar int)

RETURNS TABLE (

    nombre\_pelicula varchar(100),

    nombre\_genero varchar(50),

    nombre\_clasificacion varchar(50),

    anio\_relace date,

    duracion time,

    descripcion\_pelicula varchar(300),

    nombres\_directores text,

    nombres\_guionistas text

) AS $$

DECLARE

    id\_director int;

    id\_guionista int;

BEGIN

    SELECT id\_tipo\_cargo INTO id\_director

    FROM elenco.tipo\_cargo

    WHERE nombre\_tipo\_cargo = 'Director';

    SELECT id\_tipo\_cargo INTO id\_guionista

    FROM elenco.tipo\_cargo

    WHERE nombre\_tipo\_cargo = 'Guionista';

    RETURN QUERY

    SELECT

        cpp.nombre\_pelicula,

        cpg.nombre\_genero,

        cpc.nombre\_clasificacion,

        cpp.anio\_relace,

        cpp.duracion,

        cpp.descripcion\_pelicula,

        (

            SELECT STRING\_AGG(CONCAT(ea\_director.nombres, ' ', ea\_director.apellidos), ', ')

            FROM elenco.artista ea\_director

            INNER JOIN elenco.pelicula\_artista\_cargo epac\_director ON ea\_director.id\_artista = epac\_director.id\_artista

            WHERE epac\_director.id\_pelicula = cpp.id\_pelicula AND epac\_director.id\_tipo\_cargo = id\_director

        ) AS nombres\_directores,

        (

            SELECT STRING\_AGG(CONCAT(ea\_guionista.nombres, ' ', ea\_guionista.apellidos), ', ')

            FROM elenco.artista ea\_guionista

            INNER JOIN elenco.pelicula\_artista\_cargo epac\_guionista ON ea\_guionista.id\_artista = epac\_guionista.id\_artista

            WHERE epac\_guionista.id\_pelicula = cpp.id\_pelicula AND epac\_guionista.id\_tipo\_cargo = id\_guionista

        ) AS nombres\_guionistas

    FROM

        catalogo\_programacion.pelicula cpp LEFT JOIN catalogo\_programacion.genero cpg

            ON cpp.id\_genero = cpg.id\_genero

            LEFT JOIN catalogo\_programacion.clasificacion cpc ON cpp.id\_clasificacion = cpc.id\_clasificacion

    WHERE

        cpp.id\_pelicula = id\_pelicula\_mostrar;

END;

$$ LANGUAGE plpgsql;

SELECT obtener\_ficha\_pelicula(6);

-- Sp para listar las películas por genero

CREATE OR REPLACE FUNCTION obtener\_peliculas\_genero(nombre\_genero\_moatrar varchar(50))

RETURNS TABLE (

    nombre\_pelicula varchar(100),

    nombre\_genero varchar(50),

    nombre\_clasificacion varchar(50),

    anio\_relace date,

    duracion time,

    descripcion\_pelicula varchar(300),

    nombres\_directores text,

    nombres\_guionistas text

) AS $$

DECLARE

    id\_director int;

    id\_guionista int;

BEGIN

    SELECT id\_tipo\_cargo INTO id\_director

    FROM elenco.tipo\_cargo

    WHERE nombre\_tipo\_cargo = 'Director';

    SELECT id\_tipo\_cargo INTO id\_guionista

    FROM elenco.tipo\_cargo

    WHERE nombre\_tipo\_cargo = 'Guionista';

    RETURN QUERY

    SELECT

        cpp.nombre\_pelicula,

        cpg.nombre\_genero,

        cpc.nombre\_clasificacion,

        cpp.anio\_relace,

        cpp.duracion,

        cpp.descripcion\_pelicula,

        (

            SELECT STRING\_AGG(CONCAT(ea\_director.nombres, ' ', ea\_director.apellidos), ', ')

            FROM elenco.artista ea\_director

            INNER JOIN elenco.pelicula\_artista\_cargo epac\_director ON ea\_director.id\_artista = epac\_director.id\_artista

            WHERE epac\_director.id\_pelicula = cpp.id\_pelicula AND epac\_director.id\_tipo\_cargo = id\_director

        ) AS nombres\_directores,

        (

            SELECT STRING\_AGG(CONCAT(ea\_guionista.nombres, ' ', ea\_guionista.apellidos), ', ')

            FROM elenco.artista ea\_guionista

            INNER JOIN elenco.pelicula\_artista\_cargo epac\_guionista ON ea\_guionista.id\_artista = epac\_guionista.id\_artista

            WHERE epac\_guionista.id\_pelicula = cpp.id\_pelicula AND epac\_guionista.id\_tipo\_cargo = id\_guionista

        ) AS nombres\_guionistas

    FROM

        catalogo\_programacion.pelicula cpp LEFT JOIN catalogo\_programacion.genero cpg

            ON cpp.id\_genero = cpg.id\_genero

            LEFT JOIN catalogo\_programacion.clasificacion cpc

            ON cpp.id\_clasificacion = cpc.id\_clasificacion

    WHERE

        cpg.nombre\_genero = nombre\_genero\_moatrar;

END;

$$ LANGUAGE plpgsql;

SELECT obtener\_peliculas\_genero('Comedia');

-- Sp para listar las películas por clasificación

CREATE OR REPLACE FUNCTION obtener\_peliculas\_clasificacion(nombre\_clasificacion\_mostrar varchar(50))

RETURNS TABLE (

    nombre\_pelicula varchar(100),

    nombre\_genero varchar(50),

    nombre\_clasificacion varchar(50),

    anio\_relace date,

    duracion time,

    descripcion\_pelicula varchar(300),

    nombres\_directores text,

    nombres\_guionistas text

) AS $$

DECLARE

    id\_director int;

    id\_guionista int;

BEGIN

    SELECT id\_tipo\_cargo INTO id\_director

    FROM elenco.tipo\_cargo

    WHERE nombre\_tipo\_cargo = 'Director';

    SELECT id\_tipo\_cargo INTO id\_guionista

    FROM elenco.tipo\_cargo

    WHERE nombre\_tipo\_cargo = 'Guionista';

    RETURN QUERY

    SELECT

        cpp.nombre\_pelicula,

        cpg.nombre\_genero,

        cpc.nombre\_clasificacion,

        cpp.anio\_relace,

        cpp.duracion,

        cpp.descripcion\_pelicula,

        (

            SELECT STRING\_AGG(CONCAT(ea\_director.nombres, ' ', ea\_director.apellidos), ', ')

            FROM elenco.artista ea\_director

            INNER JOIN elenco.pelicula\_artista\_cargo epac\_director ON ea\_director.id\_artista = epac\_director.id\_artista

            WHERE epac\_director.id\_pelicula = cpp.id\_pelicula AND epac\_director.id\_tipo\_cargo = 1

        ) AS nombres\_directores,

        (

            SELECT STRING\_AGG(CONCAT(ea\_guionista.nombres, ' ', ea\_guionista.apellidos), ', ')

            FROM elenco.artista ea\_guionista

            INNER JOIN elenco.pelicula\_artista\_cargo epac\_guionista ON ea\_guionista.id\_artista = epac\_guionista.id\_artista

            WHERE epac\_guionista.id\_pelicula = cpp.id\_pelicula AND epac\_guionista.id\_tipo\_cargo = 1

        ) AS nombres\_guionistas

    FROM

        catalogo\_programacion.pelicula cpp LEFT JOIN catalogo\_programacion.genero cpg

            ON cpp.id\_genero = cpg.id\_genero

            LEFT JOIN catalogo\_programacion.clasificacion cpc

            ON cpp.id\_clasificacion = cpc.id\_clasificacion

    WHERE

        cpc.nombre\_clasificacion = nombre\_clasificacion\_mostrar;

END;

$$ LANGUAGE plpgsql;

SELECT obtener\_peliculas\_clasificacion('R');

-- Sp para mostrar la cantidad de películas por genero

CREATE OR REPLACE FUNCTION obtener\_cantidad\_peliculas\_gener()

RETURNS TABLE (

    nombre\_genero varchar(50),

    cantidad\_peliculas bigint

) AS $$

BEGIN

    RETURN QUERY

    SELECT

        cpg.nombre\_genero,

        COUNT(cpp.id\_pelicula) AS cantidad\_peliculas

    FROM

        catalogo\_programacion.pelicula cpp JOIN catalogo\_programacion.genero cpg

        ON cpp.id\_genero = cpg.id\_genero

    GROUP BY

        cpg.nombre\_genero;

END;

$$ LANGUAGE plpgsql;

SELECT obtener\_cantidad\_peliculas\_gener();

-- Sp para mostrar la cantidad de películas por clasificación

CREATE OR REPLACE FUNCTION obtener\_cantidad\_peliculas\_clasificacion()

RETURNS TABLE (

    nombre\_clasificacion varchar(50),

    cantidad\_peliculas bigint

) AS $$

BEGIN

    RETURN QUERY

    SELECT

        cpc.nombre\_clasificacion,

        COUNT(cpp.id\_pelicula) AS cantidad\_peliculas

    FROM

        catalogo\_programacion.pelicula cpp JOIN catalogo\_programacion.clasificacion cpc

        ON cpp.id\_clasificacion = cpc.id\_clasificacion

    GROUP BY

        cpc.nombre\_clasificacion;

END;

$$ LANGUAGE plpgsql;

SELECT obtener\_cantidad\_peliculas\_clasificacion();

-- Sp para mostrar la cantidad de películas por artista

CREATE OR REPLACE FUNCTION obtener\_cantidad\_peliculas\_artista()

RETURNS TABLE (

    nombres varchar(50),

    apellidos varchar(50),

    cantidad\_peliculas bigint

) AS $$

BEGIN

    RETURN QUERY

    SELECT

        ea.nombres,

        ea.apellidos,

        COUNT(epac.id\_pelicula) AS cantidad\_peliculas

    FROM

        elenco.pelicula\_artista\_cargo epac RIGHT JOIN elenco.artista ea

        ON epac.id\_artista = ea.id\_artista

    GROUP BY

        ea.nombres, ea.apellidos;

END;

$$ LANGUAGE plpgsql;

SELECT obtener\_cantidad\_peliculas\_artista();

-- Sp para mostrar la cantidad de películas por espacio especial

CREATE OR REPLACE FUNCTION obtener\_cantidad\_peliculas\_espacio\_especia()

RETURNS TABLE (

    nombre\_espacio\_especial varchar(100),

    cantidad\_peliculas bigint

) AS $$

BEGIN

    RETURN QUERY

    SELECT

        cpee.nombre,

        COUNT(cpep.id\_pelicula) AS cantidad\_peliculas

    FROM

        catalogo\_programacion.espacio\_especial cpee JOIN catalogo\_programacion.espacio\_especial\_pelicula cpep

        ON cpee.id\_espacio\_especial = cpep.id\_espacio\_especial

    GROUP BY

        cpee.nombre;

END;

$$ LANGUAGE plpgsql;

SELECT obtener\_cantidad\_peliculas\_espacio\_especia() ;

-- Sp para mostrar la cantidad de películas por casa productora

CREATE OR REPLACE FUNCTION obtener\_cantidad\_peliculas\_casa\_productor()

RETURNS TABLE (

    nombre\_compania\_productora varchar(100),

    cantidad\_peliculas bigint

) AS $$

BEGIN

    RETURN QUERY

    SELECT

        ccp.nombre\_compania\_productora,

        COUNT(cpp.id\_pelicula) AS cantidad\_peliculas

    FROM

        catalogo\_programacion.pelicula cpp LEFT JOIN catalogo\_programacion.pelicula\_compania\_productora cppcp

            ON cpp.id\_pelicula = cppcp.id\_pelicula

            LEFT JOIN catalogo\_programacion.compania\_productora ccp

            ON cppcp.id\_compania\_productora = ccp.id\_compania\_productora

    GROUP BY

        ccp.nombre\_compania\_productora;

END;

$$ LANGUAGE plpgsql;

SELECT obtener\_cantidad\_peliculas\_casa\_productor();

-- Sp para mostrar los datos de las películas en un rango de duración

CREATE OR REPLACE FUNCTION obtener\_peliculas\_rango\_duracion(duracion\_min time, duracion\_max time)

RETURNS TABLE (

    nombre\_pelicula varchar(100),

    nombre\_genero varchar(50),

    nombre\_clasificacion varchar(50),

    anio\_relace date,

    duracion time,

    descripcion\_pelicula varchar(300)

) AS $$

BEGIN

    RETURN QUERY

    SELECT

        cpp.nombre\_pelicula,

        cpg.nombre\_genero,

        cpc.nombre\_clasificacion,

        cpp.anio\_relace,

        cpp.duracion,

        cpp.descripcion\_pelicula

    FROM

        catalogo\_programacion.pelicula cpp LEFT JOIN catalogo\_programacion.genero cpg

            ON cpp.id\_genero = cpg.id\_genero

            LEFT JOIN catalogo\_programacion.clasificacion cpc

            ON cpp.id\_clasificacion = cpc.id\_clasificacion

    WHERE

        cpp.duracion >= duracion\_min AND cpp.duracion <= duracion\_max;

END;

$$ LANGUAGE plpgsql;

SELECT obtener\_peliculas\_rango\_duracion('02:19:00','03:15:00');

-- Sp para obtener la clasificación con más películas, moda, bimodal y hasta 3 modas

CREATE OR REPLACE FUNCTION obtener\_clasificacion\_moda(cantidad\_modas int)

RETURNS TABLE (

    nombre\_clasificacion varchar(50),

    cantidad\_peliculas bigint

) AS $$

BEGIN

    RETURN QUERY

    SELECT

        cpc.nombre\_clasificacion,

        COUNT(cpp.id\_pelicula) AS cantidad\_peliculas

    FROM

        catalogo\_programacion.pelicula cpp JOIN catalogo\_programacion.clasificacion cpc

            ON cpp.id\_clasificacion = cpc.id\_clasificacion

    GROUP BY

        cpc.nombre\_clasificacion

    ORDER BY

        COUNT(cpp.id\_pelicula) DESC

    LIMIT cantidad\_modas;

END;

$$ LANGUAGE plpgsql;

SELECT obtener\_clasificacion\_moda(2);

--Sp para mostrar los datos de un artista calculando la edad (crear una función) nombre del artista, fecha de nacimiento, edad.

CREATE OR REPLACE FUNCTION obtener\_datos\_artista(p\_id\_artista int)

RETURNS TABLE (

    p\_nombres varchar(50),

    p\_apellidos varchar(50),

    p\_fecha\_nacimiento date,

    p\_edad numeric

) AS $$

BEGIN

    RETURN QUERY

    SELECT

        ea.nombres,

        ea.apellidos,

        ea.fecha\_nacimiento,

        EXTRACT(YEAR FROM AGE(NOW(), ea.fecha\_nacimiento)) AS edad

    FROM

        elenco.artista ea

    WHERE

        ea.id\_artista = p\_id\_artista;

END;

$$ LANGUAGE plpgsql;

SELECT obtener\_datos\_artista(2);

## Implementación de las tablas de auditoria(Audilog) con sus respectivos disparadores(Trigger)

-- tabla para normalizar los tipos de cambios

CREATE TABLE tipo\_cambio (

    id\_tipo\_cambio SERIAL PRIMARY KEY,

    nombre\_tipo\_cambio VARCHAR(10) UNIQUE

);

-- Insertar los tipos de cambio por defecto

INSERT INTO tipo\_cambio (nombre\_tipo\_cambio) VALUES ('INSERT'), ('UPDATE'), ('DELETE');

------------------------------------------------------------------------------------------------------------------

-- auditoria a la tabla peliculas

CREATE TABLE auditoria\_peliculas (

    id\_registro SERIAL PRIMARY KEY,

    id\_pelicula INT,

    fecha TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

    id\_tipo\_cambio INT,

    valores\_anteriores JSONB,

    valores\_nuevos JSONB,

    FOREIGN KEY (id\_tipo\_cambio) REFERENCES tipo\_cambio (id\_tipo\_cambio)

);

-- trigger asociado a su auditoria

CREATE OR REPLACE FUNCTION auditar\_peliculas()

RETURNS TRIGGER AS $$

BEGIN

    IF TG\_OP = 'INSERT' THEN

        INSERT INTO auditoria\_peliculas (id\_pelicula, id\_tipo\_cambio, valores\_nuevos)

        VALUES (NEW.id\_pelicula, (SELECT id\_tipo\_cambio FROM tipo\_cambio WHERE nombre\_tipo\_cambio = 'INSERT'), to\_jsonb(NEW));

    ELSIF TG\_OP = 'UPDATE' THEN

        INSERT INTO auditoria\_peliculas (id\_pelicula, id\_tipo\_cambio, valores\_anteriores, valores\_nuevos)

        VALUES (NEW.id\_pelicula, (SELECT id\_tipo\_cambio FROM tipo\_cambio WHERE nombre\_tipo\_cambio = 'UPDATE'), to\_jsonb(OLD), to\_jsonb(NEW));

    ELSIF TG\_OP = 'DELETE' THEN

        INSERT INTO auditoria\_peliculas (id\_pelicula, id\_tipo\_cambio, valores\_anteriores)

        VALUES (OLD.id\_pelicula, (SELECT id\_tipo\_cambio FROM tipo\_cambio WHERE nombre\_tipo\_cambio = 'DELETE'), to\_jsonb(OLD));

    END IF;

    RETURN NULL;

END;

$$ LANGUAGE plpgsql;

-- Asociar el disparador a la tabla de películas

CREATE TRIGGER peliculas\_trigger AFTER INSERT OR UPDATE OR DELETE

ON catalogo\_programacion.pelicula

FOR EACH ROW EXECUTE FUNCTION auditar\_peliculas();

------------------------------------------------------------------------------------------------------------------

-- auditoria a la tabla artista

CREATE TABLE auditoria\_artista (

    id\_registro SERIAL PRIMARY KEY,

    id\_artista INT,

    fecha TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

    id\_tipo\_cambio INT,

    valores\_anteriores JSONB,

    valores\_nuevos JSONB,

    FOREIGN KEY (id\_tipo\_cambio) REFERENCES tipo\_cambio (id\_tipo\_cambio)

);

-- trigger asociado a su auditoria

CREATE OR REPLACE FUNCTION auditar\_artista()

RETURNS TRIGGER AS $$

BEGIN

    IF TG\_OP = 'INSERT' THEN

        INSERT INTO auditoria\_artista (id\_artista, id\_tipo\_cambio, valores\_nuevos)

        VALUES (NEW.id\_artista, (SELECT id\_tipo\_cambio FROM tipo\_cambio WHERE nombre\_tipo\_cambio = 'INSERT'), to\_jsonb(NEW));

    ELSIF TG\_OP = 'UPDATE' THEN

        INSERT INTO auditoria\_artista (id\_artista, id\_tipo\_cambio, valores\_anteriores, valores\_nuevos)

        VALUES (NEW.id\_artista, (SELECT id\_tipo\_cambio FROM tipo\_cambio WHERE nombre\_tipo\_cambio = 'UPDATE'), to\_jsonb(OLD), to\_jsonb(NEW));

    ELSIF TG\_OP = 'DELETE' THEN

        INSERT INTO auditoria\_artista (id\_artista, id\_tipo\_cambio, valores\_anteriores)

        VALUES (OLD.id\_artista, (SELECT id\_tipo\_cambio FROM tipo\_cambio WHERE nombre\_tipo\_cambio = 'DELETE'), to\_jsonb(OLD));

    END IF;

    RETURN NULL;

END;

$$ LANGUAGE plpgsql;

-- Asociar el disparador a la tabla de artistas

CREATE TRIGGER artista\_trigger AFTER INSERT OR UPDATE OR DELETE

ON elenco.artista

FOR EACH ROW EXECUTE FUNCTION auditar\_artista();

------------------------------------------------------------------------------------------------------------------

-- auditoria a la tabla personaje

CREATE TABLE auditoria\_personaje (

    id\_registro SERIAL PRIMARY KEY,

    id\_personaje INT,

    fecha TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

    id\_tipo\_cambio INT,

    valores\_anteriores JSONB,

    valores\_nuevos JSONB,

    FOREIGN KEY (id\_tipo\_cambio) REFERENCES tipo\_cambio (id\_tipo\_cambio)

);

-- trigger asociado a su auditoria

CREATE OR REPLACE FUNCTION auditar\_personaje()

RETURNS TRIGGER AS $$

BEGIN

    IF TG\_OP = 'INSERT' THEN

        INSERT INTO auditoria\_personaje (id\_personaje, id\_tipo\_cambio, valores\_nuevos)

        VALUES (NEW.id\_personaje, (SELECT id\_tipo\_cambio FROM tipo\_cambio WHERE nombre\_tipo\_cambio = 'INSERT'), to\_jsonb(NEW));

    ELSIF TG\_OP = 'UPDATE' THEN

        INSERT INTO auditoria\_personaje (id\_personaje, id\_tipo\_cambio, valores\_anteriores, valores\_nuevos)

        VALUES (NEW.id\_personaje, (SELECT id\_tipo\_cambio FROM tipo\_cambio WHERE nombre\_tipo\_cambio = 'UPDATE'), to\_jsonb(OLD), to\_jsonb(NEW));

    ELSIF TG\_OP = 'DELETE' THEN

        INSERT INTO auditoria\_personaje (id\_personaje, id\_tipo\_cambio, valores\_anteriores)

        VALUES (OLD.id\_personaje, (SELECT id\_tipo\_cambio FROM tipo\_cambio WHERE nombre\_tipo\_cambio = 'DELETE'), to\_jsonb(OLD));

    END IF;

    RETURN NULL;

END;

$$ LANGUAGE plpgsql;

-- Asociar el disparador a la tabla de personajes

CREATE TRIGGER personaje\_trigger AFTER INSERT OR UPDATE OR DELETE

ON elenco.personaje

FOR EACH ROW EXECUTE FUNCTION auditar\_personaje();

------------------------------------------------------------------------------------------------------------------

-- auditoria a la tabla programacion

CREATE TABLE auditoria\_programacion (

    id\_registro SERIAL PRIMARY KEY,

    id\_programacion INT,

    fecha TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

    id\_tipo\_cambio INT,

    valores\_anteriores JSONB,

    valores\_nuevos JSONB,

    FOREIGN KEY (id\_tipo\_cambio) REFERENCES tipo\_cambio (id\_tipo\_cambio)

);

-- trigger asociado a su auditoria

CREATE OR REPLACE FUNCTION auditar\_programacion()

RETURNS TRIGGER AS $$

BEGIN

    IF TG\_OP = 'INSERT' THEN

        INSERT INTO auditoria\_programacion (id\_programacion, id\_tipo\_cambio, valores\_nuevos)

        VALUES (NEW.id\_programacion, (SELECT id\_tipo\_cambio FROM tipo\_cambio WHERE nombre\_tipo\_cambio = 'INSERT'), to\_jsonb(NEW));

    ELSIF TG\_OP = 'UPDATE' THEN

        INSERT INTO auditoria\_programacion (id\_programacion, id\_tipo\_cambio, valores\_anteriores, valores\_nuevos)

        VALUES (NEW.id\_programacion, (SELECT id\_tipo\_cambio FROM tipo\_cambio WHERE nombre\_tipo\_cambio = 'UPDATE'), to\_jsonb(OLD), to\_jsonb(NEW));

    ELSIF TG\_OP = 'DELETE' THEN

        INSERT INTO auditoria\_programacion (id\_programacion, id\_tipo\_cambio, valores\_anteriores)

        VALUES (OLD.id\_programacion, (SELECT id\_tipo\_cambio FROM tipo\_cambio WHERE nombre\_tipo\_cambio = 'DELETE'), to\_jsonb(OLD));

    END IF;

    RETURN NULL;

END;

$$ LANGUAGE plpgsql;

-- Asociar el disparador a la tabla de programacion

CREATE TRIGGER programacion\_trigger AFTER INSERT OR UPDATE OR DELETE

ON catalogo\_programacion.programacion

FOR EACH ROW EXECUTE FUNCTION auditar\_programacion();

------------------------------------------------------------------------------------------------------------------

-- auditoria a la tabla espacio\_especial

CREATE TABLE auditoria\_espacio\_especial (

    id\_registro SERIAL PRIMARY KEY,

    id\_espacio\_especial INT,

    fecha TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

    id\_tipo\_cambio INT,

    valores\_anteriores JSONB,

    valores\_nuevos JSONB,

    FOREIGN KEY (id\_tipo\_cambio) REFERENCES tipo\_cambio (id\_tipo\_cambio)

);

-- trigger asociado a su auditoria

CREATE OR REPLACE FUNCTION auditar\_espacio\_especial()

RETURNS TRIGGER AS $$

BEGIN

    IF TG\_OP = 'INSERT' THEN

        INSERT INTO auditoria\_espacio\_especial (id\_espacio\_especial, id\_tipo\_cambio, valores\_nuevos)

        VALUES (NEW.id\_espacio\_especial, (SELECT id\_tipo\_cambio FROM tipo\_cambio WHERE nombre\_tipo\_cambio = 'INSERT'), to\_jsonb(NEW));

    ELSIF TG\_OP = 'UPDATE' THEN

        INSERT INTO auditoria\_espacio\_especial (id\_espacio\_especial, id\_tipo\_cambio, valores\_anteriores, valores\_nuevos)

        VALUES (NEW.id\_espacio\_especial, (SELECT id\_tipo\_cambio FROM tipo\_cambio WHERE nombre\_tipo\_cambio = 'UPDATE'), to\_jsonb(OLD), to\_jsonb(NEW));

    ELSIF TG\_OP = 'DELETE' THEN

        INSERT INTO auditoria\_espacio\_especial (id\_espacio\_especial, id\_tipo\_cambio, valores\_anteriores)

        VALUES (OLD.id\_espacio\_especial, (SELECT id\_tipo\_cambio FROM tipo\_cambio WHERE nombre\_tipo\_cambio = 'DELETE'), to\_jsonb(OLD));

    END IF;

    RETURN NULL;

END;

$$ LANGUAGE plpgsql;

-- Asociar el disparador a la tabla de espacio\_especial

CREATE TRIGGER espacio\_especial\_trigger AFTER INSERT OR UPDATE OR DELETE

ON catalogo\_programacion.espacio\_especial

FOR EACH ROW EXECUTE FUNCTION auditar\_espacio\_especial();

------------------------------------------------------------------------------------------------------------------

-- auditoria a la tabla compania\_productora

CREATE TABLE auditoria\_compania\_productora (

    id\_registro SERIAL PRIMARY KEY,

    id\_compania\_productora INT,

    fecha TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

    id\_tipo\_cambio INT,

    valores\_anteriores JSONB,

    valores\_nuevos JSONB,

    FOREIGN KEY (id\_tipo\_cambio) REFERENCES tipo\_cambio (id\_tipo\_cambio)

);

-- trigger asociado a su auditoria

CREATE OR REPLACE FUNCTION auditar\_compania\_productora()

RETURNS TRIGGER AS $$

BEGIN

    IF TG\_OP = 'INSERT' THEN

        INSERT INTO auditoria\_compania\_productora (id\_compania\_productora, id\_tipo\_cambio, valores\_nuevos)

        VALUES (NEW.id\_compania\_productora, (SELECT id\_tipo\_cambio FROM tipo\_cambio WHERE nombre\_tipo\_cambio = 'INSERT'), to\_jsonb(NEW));

    ELSIF TG\_OP = 'UPDATE' THEN

        INSERT INTO auditoria\_compania\_productora (id\_compania\_productora, id\_tipo\_cambio, valores\_anteriores, valores\_nuevos)

        VALUES (NEW.id\_compania\_productora, (SELECT id\_tipo\_cambio FROM tipo\_cambio WHERE nombre\_tipo\_cambio = 'UPDATE'), to\_jsonb(OLD), to\_jsonb(NEW));

    ELSIF TG\_OP = 'DELETE' THEN

        INSERT INTO auditoria\_compania\_productora (id\_compania\_productora, id\_tipo\_cambio, valores\_anteriores)

        VALUES (OLD.id\_compania\_productora, (SELECT id\_tipo\_cambio FROM tipo\_cambio WHERE nombre\_tipo\_cambio = 'DELETE'), to\_jsonb(OLD));

    END IF;

    RETURN NULL;

END;

$$ LANGUAGE plpgsql;

-- Asociar el disparador a la tabla de compania\_productora

CREATE TRIGGER compania\_productora\_trigger AFTER INSERT OR UPDATE OR DELETE

ON catalogo\_programacion.compania\_productora

FOR EACH ROW EXECUTE FUNCTION auditar\_compania\_productora();

## Disparadores(Triggers) especiales

-- para validar que no se ingrese un manager si el nuevo artista no es un actor o director

CREATE OR REPLACE FUNCTION validar\_manager()

RETURNS TRIGGER AS $$

BEGIN

    IF NEW.id\_profesion IN (SELECT id\_profesion FROM elenco.profesion WHERE nombre\_profesion IN ('actor', 'director')) THEN

        RETURN NEW;

    ELSE

        RAISE EXCEPTION 'El artista solo puede tener un manager si es actor o director';

    END IF;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER validar\_manager\_before\_insert

BEFORE INSERT ON elenco.artista

FOR EACH ROW

EXECUTE FUNCTION validar\_manager();

-- ara validar que no se agrege un registro en sitie a un artista que no sea actor

CREATE OR REPLACE FUNCTION validar\_sitie()

RETURNS TRIGGER AS $$

BEGIN

    IF EXISTS (

        SELECT 1 FROM elenco.artista

        WHERE id\_artista = NEW.id\_artista

          AND id\_profesion = (SELECT id\_profesion FROM elenco.profesion WHERE nombre\_profesion = 'actor')

    ) THEN

        RETURN NEW;

    ELSE

        RAISE EXCEPTION 'Solo se pueden crear registros en sitie para artistas cuya profesión sea actor';

    END IF;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER validar\_sitie\_before\_insert

BEFORE INSERT ON elenco.sitie

FOR EACH ROW

EXECUTE FUNCTION validar\_sitie();

-- para validar que no se agrege un personaje a una pelicula de un artista que no

--haya desempeñado el tipo de cargo de actor en la tabla pelicula\_artista\_cargo

CREATE OR REPLACE FUNCTION validar\_personaje()

RETURNS TRIGGER AS $$

BEGIN

    IF EXISTS (

        SELECT 1 FROM elenco.pelicula\_artista\_cargo

        WHERE id\_pelicula = NEW.id\_pelicula

          AND id\_artista = NEW.id\_artista

          AND id\_tipo\_cargo = (SELECT id\_tipo\_cargo FROM elenco.tipo\_cargo WHERE nombre\_tipo\_cargo = 'Actor')

    ) THEN

        RETURN NEW;

    ELSE

        RAISE EXCEPTION 'Solo se pueden crear personajes para artistas que hayan desempeñado el tipo de cargo de actor en la pelicula';

    END IF;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER validar\_personaje\_before\_insert

BEFORE INSERT ON elenco.personaje

FOR EACH ROW

EXECUTE FUNCTION validar\_personaje();

# Diagrama Entidad Relación

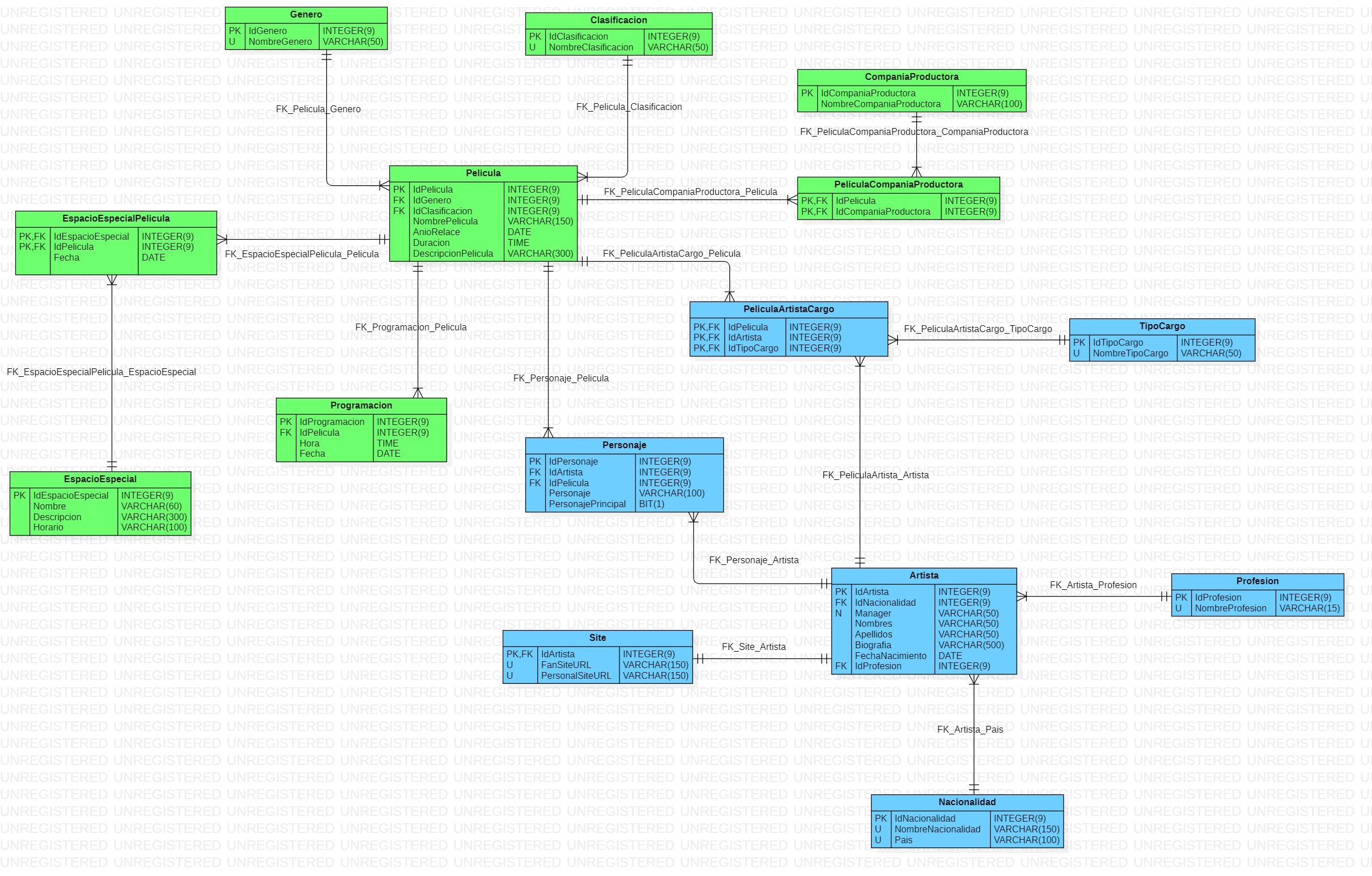


Ilustración 23. Diagrama Entidad Relacion

# Diagrama Relacional

Ilustración 24. Diagrama Relacional

# Referencias

Herrera, A. (2007, 07 05). *Nombre del Elefante*. Obtenido de https://www.postgresql.org/message-id/20070705154603.GE5500%40alvh.no-ip.org

PostgreSQL, E. G. (2024, 2 8). *PostgreSQL*. Obtenido de https://www.postgresql.org/about/

PostgreSQL, E. G. (2024, 02 08). *PostgreSQL*. Obtenido de https://www.postgresql.org/docs/8.1/datatype.html

PostgreSQL, E. G. (2024, 2 8). *postgresql documentacion*. Obtenido de https://www.postgresql.org/docs/current/history.html#HISTORY-BERKELEY

Team, T. p. (2024, 04 01). *postgresql.* Obtenido de https://ftp.postgresql.org/pub/pgadmin/pgadmin4/v8.5/docs/pgadmin4-8.5.pdf