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



# 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](#_Toc164367761)

[C1 DESARROLLO DE CASO 1](#_Toc164367762)

[1 CONTENIDO 3](#_Toc164367763)

[2 TABLA DE ILUSTRACIONES 5](#_Toc164367764)

[3 INTRODUCCIÓN 6](#_Toc164367765)

[4 OBJETIVOS 7](#_Toc164367766)

[4.1 General 7](#_Toc164367767)

[4.2 Específicos 7](#_Toc164367768)

[5 PostGreSQL 8](#_Toc164367769)

[5.1 ¿Qué es PostgreSQL? 8](#_Toc164367770)

[5.1.1 Características: 8](#_Toc164367771)

[5.2 Breve historia de PostgreSQL 11](#_Toc164367772)

[5.2.1 El proyecto POSTGRES de Berkeley 11](#_Toc164367773)

[5.2.2 Postgres95 12](#_Toc164367774)

[5.2.3 PostgreSQL 14](#_Toc164367775)

[5.3 Documentación 15](#_Toc164367776)

[5.3.1 Creación de una base de datos 15](#_Toc164367777)

[5.3.2 Creación de una nueva tabla 16](#_Toc164367778)

[5.3.3 Rellenar una tabla con filas 16](#_Toc164367779)

[5.3.4 Consultar una tabla 17](#_Toc164367780)

[5.3.5 Uniones entre tablas 18](#_Toc164367781)

[5.3.6 Funciones agregadas 21](#_Toc164367782)

[5.3.7 Actualizaciones 24](#_Toc164367783)

[5.3.8 Eliminaciones 24](#_Toc164367784)

[5.3.9 Vistas 25](#_Toc164367785)

[5.3.10 Claves foráneas 25](#_Toc164367786)

[5.4 Ranking 27](#_Toc164367787)

[5.5 pgAdmin 4 28](#_Toc164367788)

[6 Modelado de Caso FP 30](#_Toc164367789)

[6.1 Instrucciones 30](#_Toc164367790)

[6.2 Pasos para modelado de la base de datos 37](#_Toc164367791)

[6.2.1 Nombrar la base de datos 37](#_Toc164367792)

[6.2.2 Identificar las Entidades 37](#_Toc164367793)

[6.2.3 Identificar los atributos de las entidades 38](#_Toc164367794)

[7 Diagrama Entidad Relación 40](#_Toc164367795)

[8 Referencias 41](#_Toc164367796)

# TABLA DE ILUSTRACIONES

[Ilustración 1.Ranking de motores de bases de datos 27](#_Toc164368038)

[Ilustración 2.Diagrama Entidad Relacion 40](#_Toc164368039)

# 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, 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, 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, 2024).

### 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, 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

### 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

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.

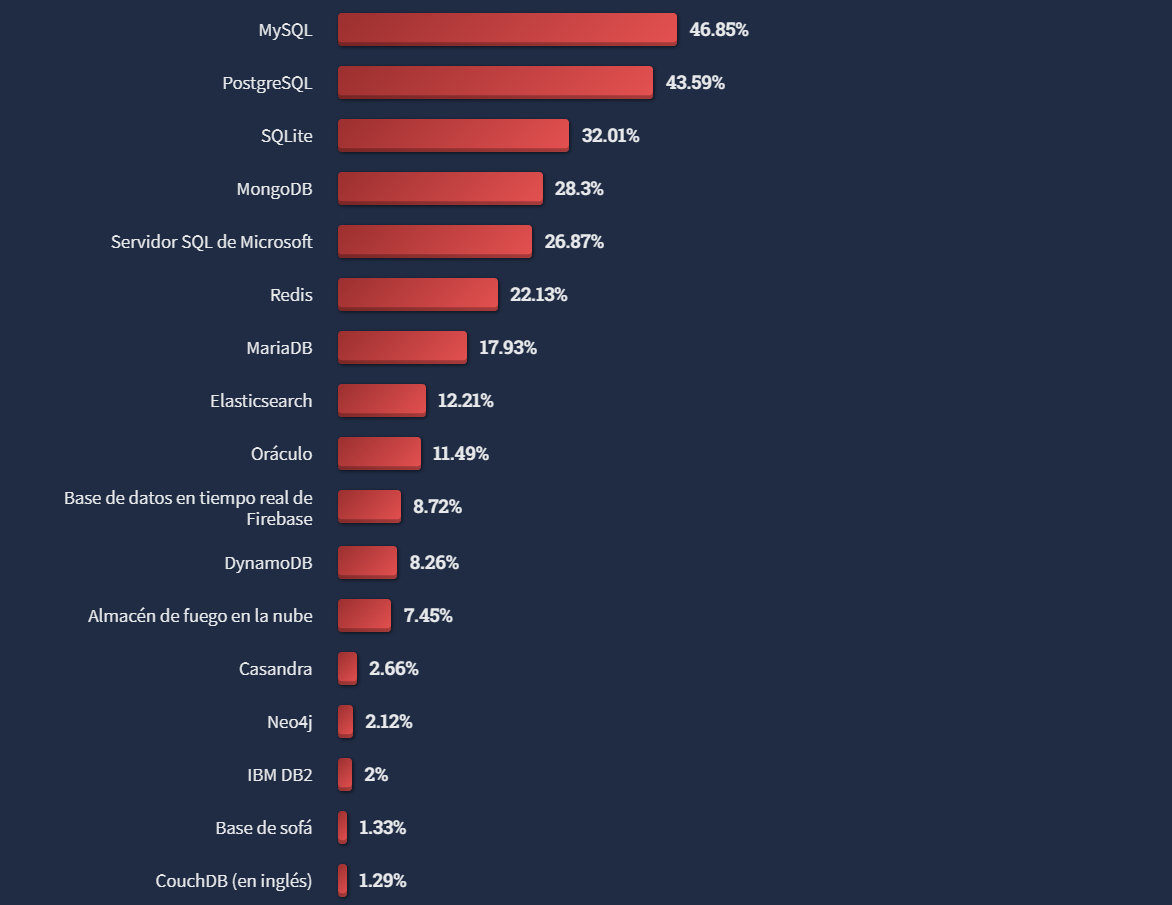


Ilustración 1.Ranking de motores de bases de datos

## 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**

#### Screma elenco

* **Profesion**
* **Nacionalidad**
* **Artista**
* **Site**
* **TipoCargo**
* **PeliculaArtistaCargo**
* **Personaje**

### Identificar los atributos de las entidades

Tabla 1 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 |
| Site | IdArtista, FanSiteURL, PersonalSiteURL |
| TipoCargo | IdTipoCargo, NombreTipoCargo |
| PeliculaArtistaCargo | IdPelicula, IdArtista, IdTipoCargo |
| Personaje | IdPersonaje, IdArtista, IdPelicula, Personaje, PersonajePrincipal |

## Diccionario de datos

### Tabla CompaniaProductora

## 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 2 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 2.

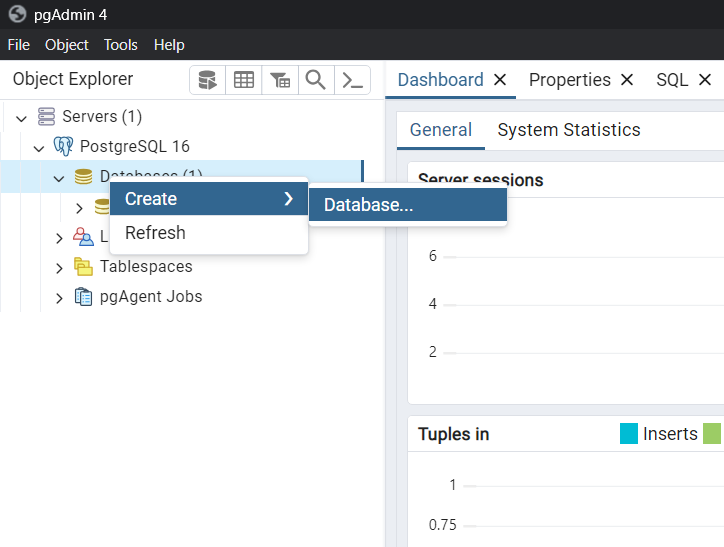


Ilustración 2. Paso 1 para crear base de datos

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, tan y como se puede apreciar en la Ilustración 4.

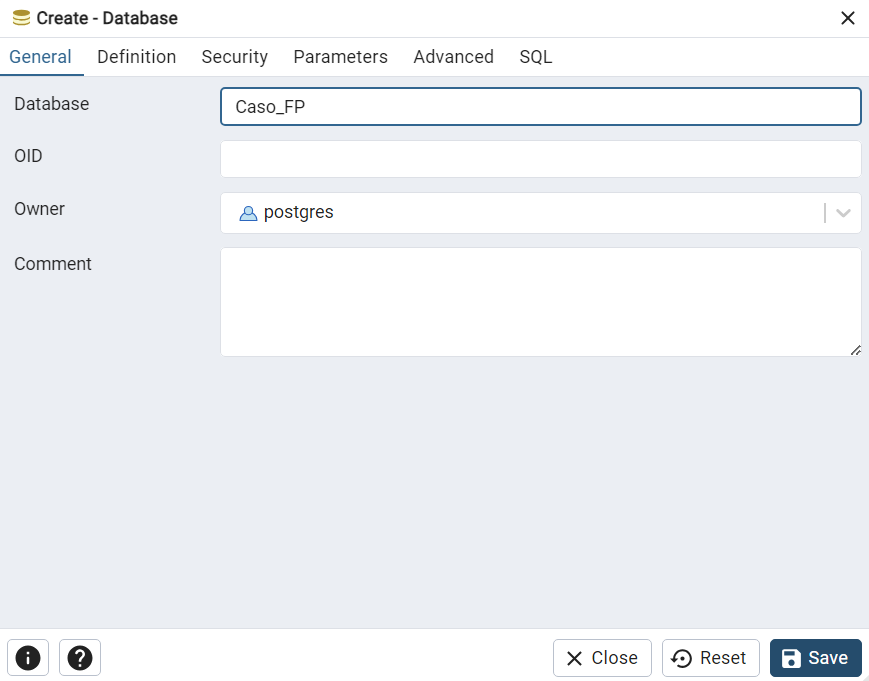


Ilustración 3. Paso 2 para crear base de datos

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 4, ahora la opción que nos interesa es la que dice Query Tool, le daremos click izquierdo sobre esta opción.

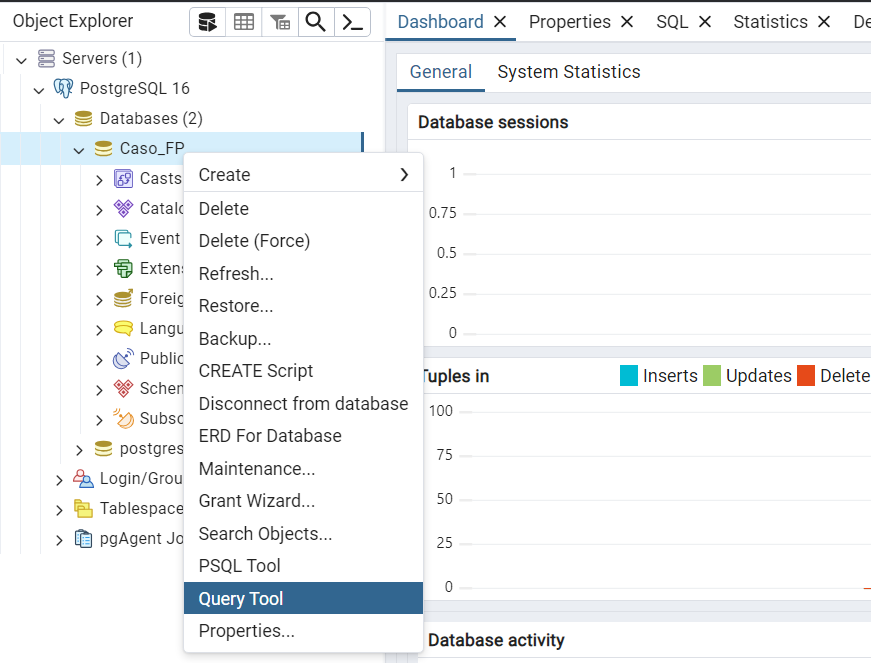


Ilustración 4. Paso 3 para crear base de datos

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 5.

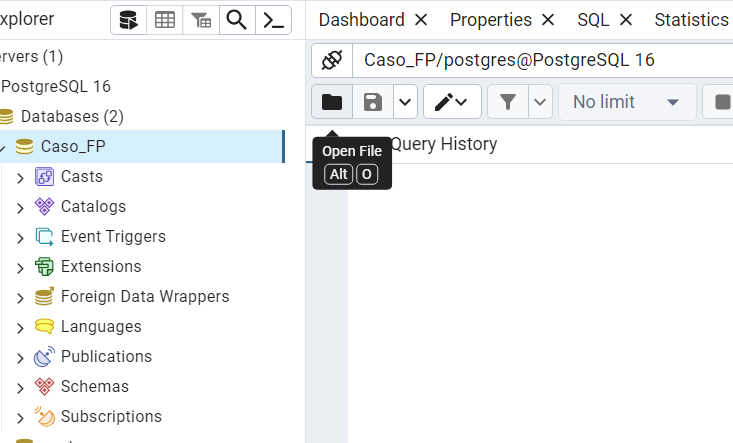


Ilustración 5. Paso 4 para crear base de datos

Se nos abrirá la ventana de la Ilustración 6 en la que tendremos que buscar el Script entre nuestros archivos seleccionarlo y darle click izquierdo al botón de **Abrir** de esa misma ventana.

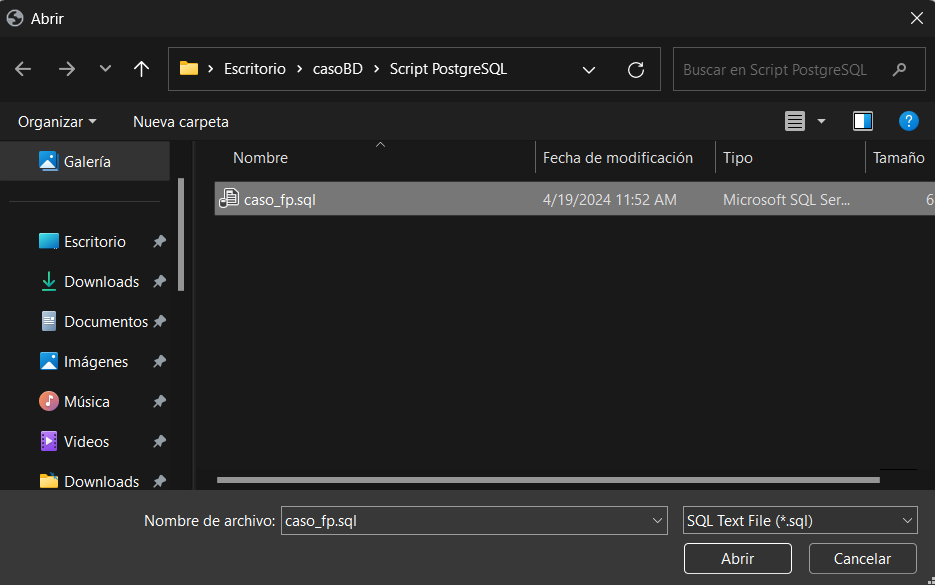


Ilustración 6. Paso 5 para crear base de datos

Una ves hecho esto se nos abrirá el Script con nuestras consultas como puede ver en la Ilustración 7 y en la que tendremos que dar click izquierdo sobre el botón Execute Script señalado en la Ilustración 7.

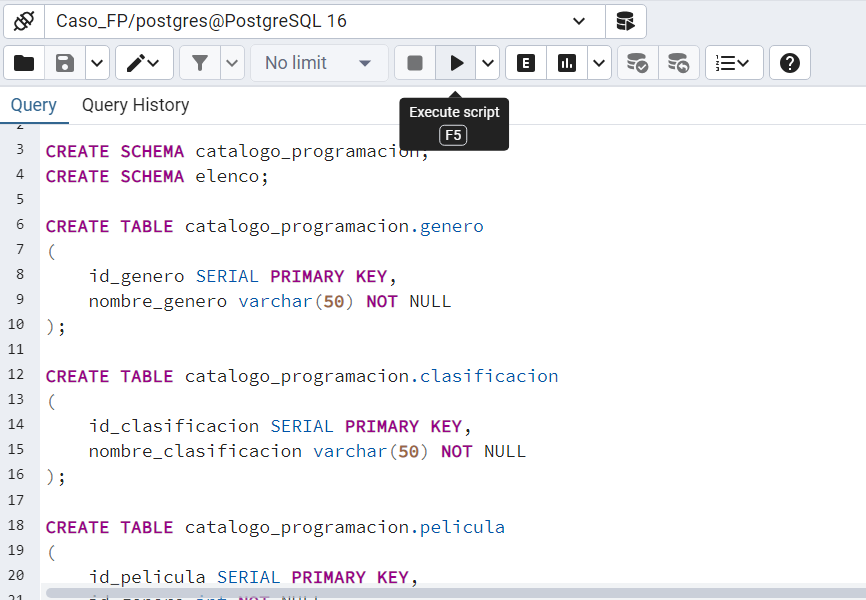


Ilustración 7. Paso 6 para crear base de datos

## Script DDL

CREATE SCHEMA catalogo\_programacion;

CREATE SCHEMA elenco;

CREATE TABLE catalogo\_programacion.genero

(

id\_genero SERIAL PRIMARY KEY,

nombre\_genero varchar(50) NOT NULL

);

CREATE TABLE catalogo\_programacion.clasificacion

(

id\_clasificacion SERIAL PRIMARY KEY,

nombre\_clasificacion varchar(50) NOT NULL

);

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,

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)

);

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,

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) NOT NULL,

personal\_site\_url varchar(150) 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)

);

# Diagrama Entidad Relación

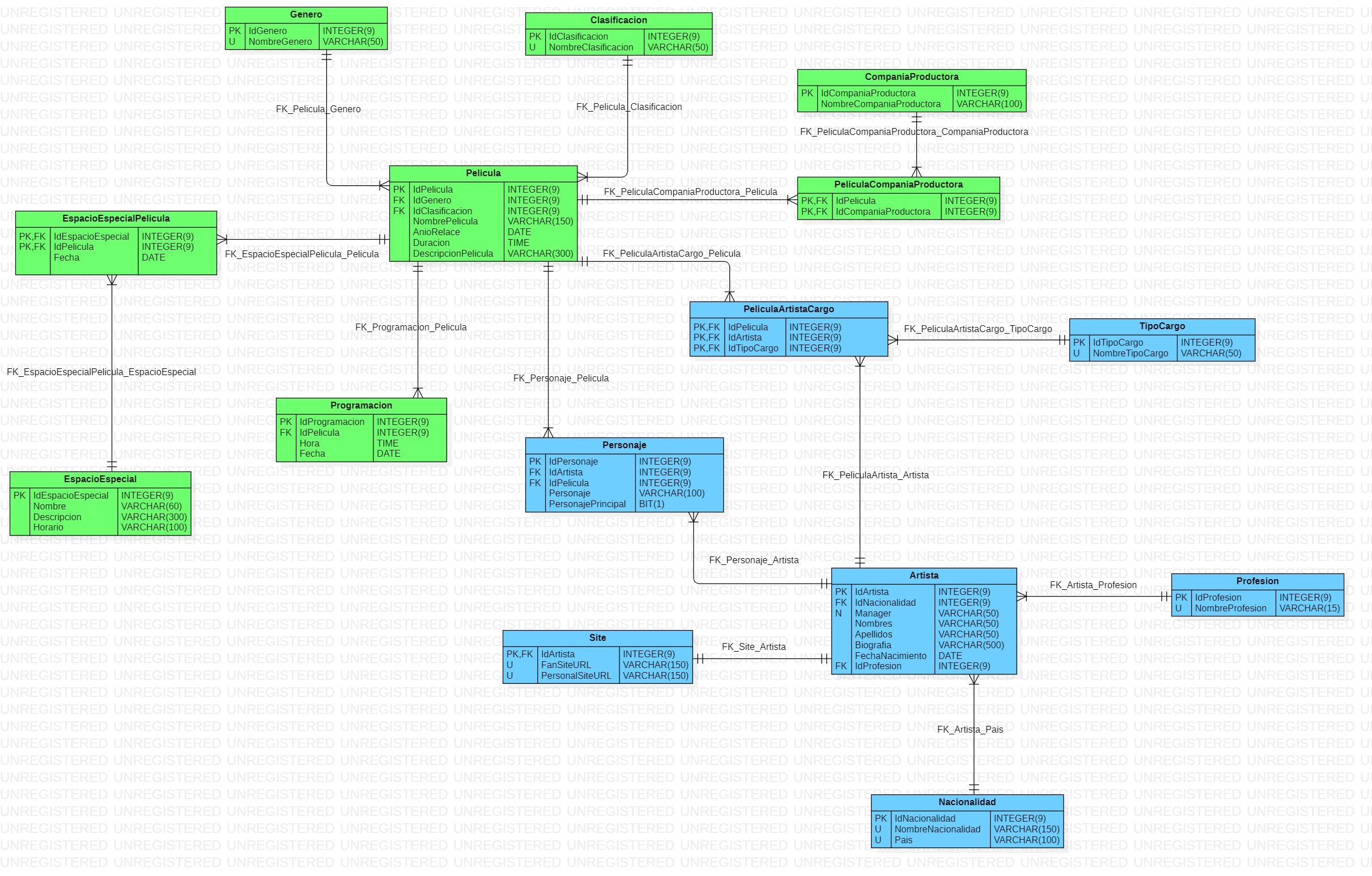


Ilustración 8. Diagrama Entidad Relacion

# Referencias

PostgreSQL, E. G. (8 de 2 de 2024). *PostgreSQL*. Obtenido de https://www.postgresql.org/about/

PostgreSQL, E. G. (8 de 2 de 2024). *postgresql documentacion*. Obtenido de https://www.postgresql.org/docs/current/history.html#HISTORY-BERKELEY

Team, T. p. (01 de 04 de 2024). *postgresql.* Obtenido de https://ftp.postgresql.org/pub/pgadmin/pgadmin4/v8.5/docs/pgadmin4-8.5.pdf