



UNIVERSIDAD VERACRUZANA
FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA
BOCA DEL RÍO, VERACRUZ



PROGRAMA EDUCATIVO

INGENIERÍA MECATRÓNICA

EXPERIENCIA EDUCATIVA

TÓPICOS AVANZADOS DE INFORMÁTICA

DOCENTE

M. YULIANA BERUMEN DÍAZ

PROYECTO
THE HUNGER GAMES

ALUMNO

INGRID MELISSA MORA HERNÁNDEZ

MATRÍCULA

S19003064

FECHA

14 DE JUNIO DE 2023

INTRODUCCIÓN

Durante la experiencia educativa de Tópicos Avanzados de Informática, adquirimos conocimientos fundamentales orientados al ámbito de la Ingeniería de Software y la Gestión de Bases de Datos. A lo largo del curso, exploramos diversos temas, como lo fueron el Modelado Orientado a Objetos, la estructura de los Diagramas Entidad-Relación y Diagramas de Clase, los cuales nos permitieron adquirir una visión profunda y entendimiento de los elementos que les conforman, tales como las entidades, atributos, relaciones y la cardinalidad.

Además, nos adentramos en la sintaxis básica del lenguaje SQL, el cual es un lenguaje enfocado en la creación y manipulación de bases de datos y utilizamos el sistema de gestión PostgreSQL, para implementación de éstas.

Como parte integral de nuestro aprendizaje, a manera de cierre “con broche de oro”, se nos encomendó la realización de un proyecto con el objetivo de poner en práctica y aplicar todo lo aprendido durante el semestre. Cada uno de nosotros recibió un planteamiento de problema único, el cual requería el diseño y la implementación de una base de datos utilizando PostgreSQL como sistema de gestión. En adición a esto, se solicitó la creación de una interfaz que permitiera la manipulación eficiente de la base de datos en cuestión. Para este propósito, empleamos Java, con apoyo de alguno de los IDEs recomendados como NetBeans, Eclipse o IntelliJ IDEA.

El objetivo principal de este proyecto es demostrar nuestra habilidad para el desarrollo de soluciones efectivas y eficientes en el ámbito de la gestión de bases de datos y la interfaz de usuario. El presente documento busca presentar un reporte detallado de los procesos y análisis que se llevaron a cabo para el correcto desarrollo y cumplimiento de lo anteriormente mencionado. A lo largo de éste, presentaré el enfoque adoptado, los desafíos encontrados y las soluciones implementadas por mí para la creación de la base de datos y la interfaz correspondientes. Este proyecto final lo vi como una oportunidad para consolidar los conocimientos teóricos adquiridos y aplicarlos en un escenario práctico, así como retomar brevemente mis orígenes como programadora retirada. Así que de antemano quiero externar mi gratitud a usted profesora por toda la experiencia de este semestre, se la estima y aprecia infinitamente ♡.

DESARROLLO

El problema que se me fue asignado fue el siguiente:

Juegos del Hambre

“Debido a la organización de los próximos juegos del hambre, el presidente de Panem, Coriolanus Snow, decidió que se debe crear un sistema de información a fin de realizar la gestión de las pruebas de entrenamiento que los tributos realizan antes de entrar a la arena. Del análisis realizado por Seneca Crane, el vigilante en jefe, se obtuvo la siguiente información:

- ✓ *El entrenamiento se compone de una serie de pruebas, en cada una de las cuales intervienen los tributos.*
- ✓ *Las pruebas son evaluadas por los vigilantes del Capitolio, siendo 1 la puntuación más baja y 12 la más alta.*
- ✓ *De cada tributo se desea guardar su CURP, nombre, sexo, edad, habilidad, puntuación obtenida en el espectáculo brindado a los vigilantes del Capitolio y el distrito al que pertenece.*
- ✓ *Cada tributo tiene un mentor, del mentor se desea guardar su CURP, nombre, sexo, edad, juego en el que resultó ganador y distrito al que pertenece.*
- ✓ *De cada vigilante se desea guardar su CURP, nombre, sexo, edad, puesto y/o especialidad.*
- ✓ *Cada Distrito se identifica por un nombre, de ellos se desea guardar en qué se especializa, puestos de trabajo, números de Juegos del Hambre ganados, tributos, cantidad de habitantes, ubicación, clima, porcentaje de hombres y mujeres y nombre de su líder.*
- ✓ *Un tributo puede pertenecer solamente a un distrito. A un Distrito pueden pertenecer muchos tributos.*
- ✓ *Los distritos son controlados por el Capitolio.*
- ✓ *Del Capitolio se desea guardar el nombre del presidente, número de habitantes, ubicación, clima, porcentaje de hombres y de mujeres, y lugares de interés.*
- ✓ *Dentro del sistema cada prueba se debe identificar a partir de un código y un nombre, se desea saber también su tipo y grado de dificultad (para esto se utilizará una escala de colores: amarillo, naranja, rojo y negro). Así también se debe registrar el nombre del participante vencedor y el tiempo empleado por este.*
- ✓ *La realización de las pruebas se desarrollará a lo largo de varias jornadas, los tributos podrán competir en varias pruebas. Para cada participante en una prueba se deben registrar la fecha en la que participa, el tiempo que le tomó llevarlo a cabo y su puntuación.”*

DESARROLLO

Lo primero que se realizó fue el diagrama Entidad-Relación, el cual en realidad formó parte de una de las primeras tareas realizadas durante el curso. Este, sin yo saberlo en su momento, me sería de utilidad para este proyecto.

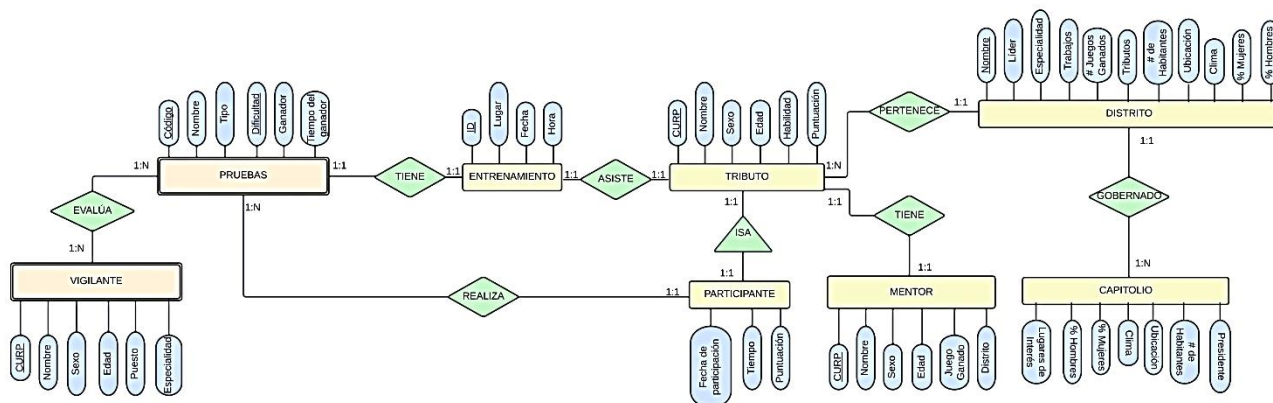


Ilustración 1. Diagrama Entidad-Relación

Como siguiente avance, se realizó la conversión de dicho Diagrama Entidad-Relación a su forma de Diagrama de Clases.

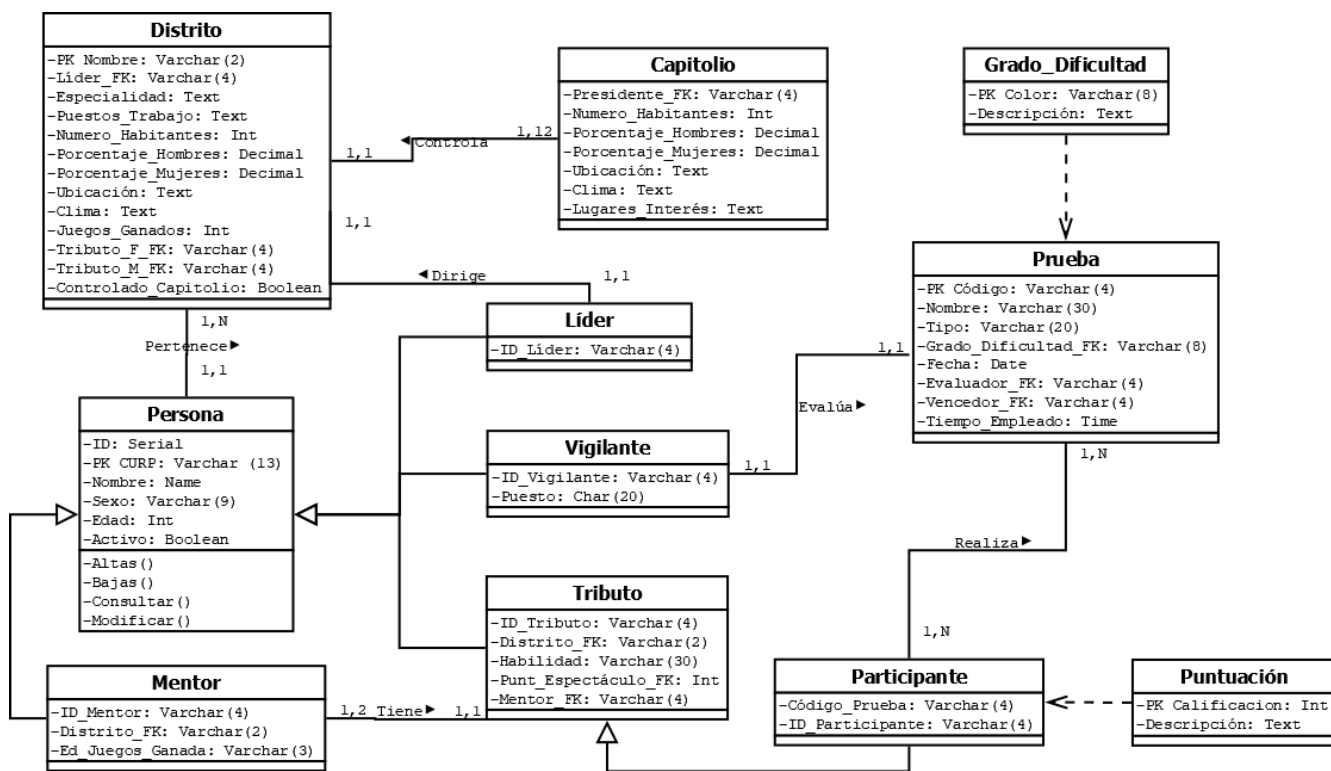
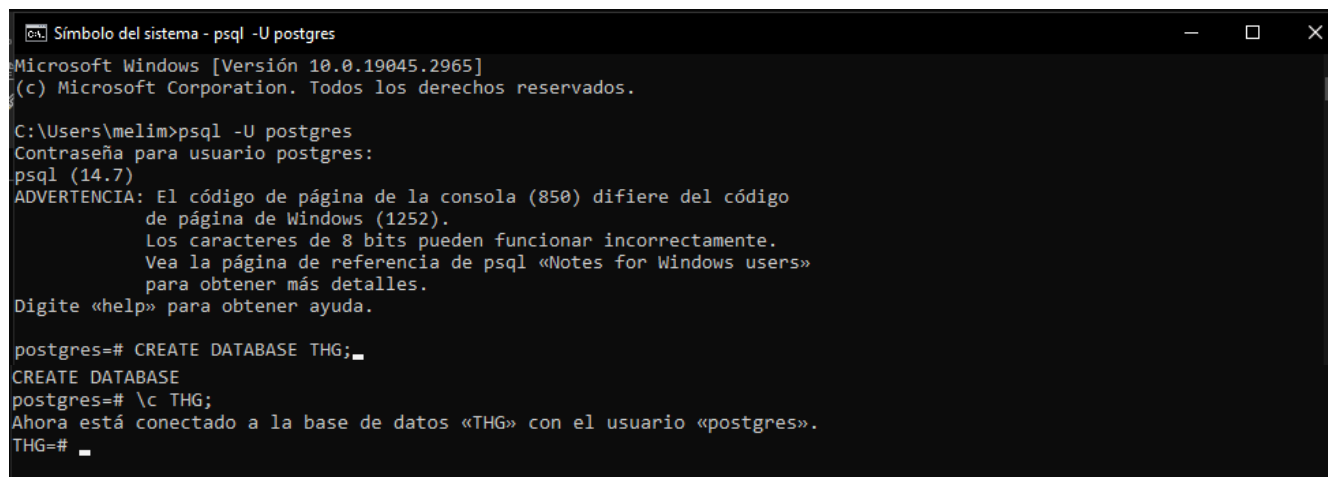


Ilustración 2. Diagrama de Clases

Y es en base a este último es que se comenzó a realizar la base de datos de PostgreSQL. Opté por realizar el proceso de la creación de la base de datos directamente en la ventana de comandos por motivos de familiaridad y comodidad.

DESARROLLO: POSTGRESQL



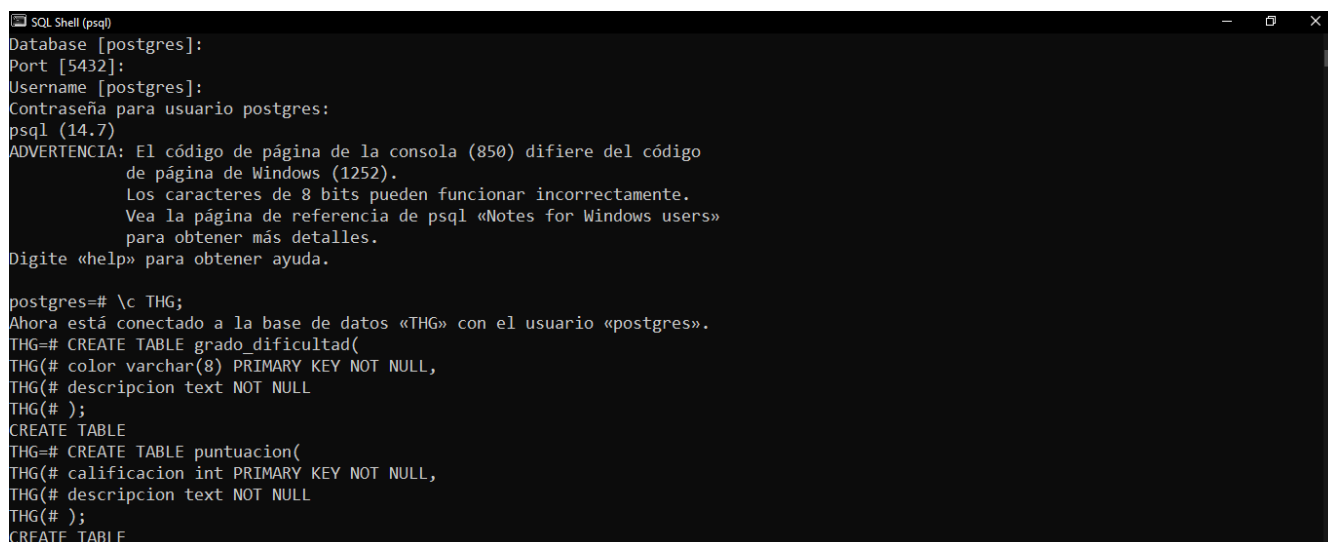
```
Símbolo del sistema - psql -U postgres
Microsoft Windows [Versión 10.0.19045.2965]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\melim>psql -U postgres
Contraseña para usuario postgres:
psql (14.7)
ADVERTENCIA: El código de página de la consola (850) difiere del código
de página de Windows (1252).
Los caracteres de 8 bits pueden funcionar incorrectamente.
Vea la página de referencia de psql «Notes for Windows users»
para obtener más detalles.
Digite «help» para obtener ayuda.

postgres=# CREATE DATABASE THG;
CREATE DATABASE
postgres=# \c THG;
Ahora está conectado a la base de datos «THG» con el usuario «postgres».
THG=#
```

Ilustración 3. Creación de la BD en CMD

Posteriormente creé cada una de mis tablas con sus respectivos atributos directo en PostgreSQL:



```
SQL Shell (psql)
Database [postgres]:
Port [5432]:
Username [postgres]:
Contraseña para usuario postgres:
psql (14.7)
ADVERTENCIA: El código de página de la consola (850) difiere del código
de página de Windows (1252).
Los caracteres de 8 bits pueden funcionar incorrectamente.
Vea la página de referencia de psql «Notes for Windows users»
para obtener más detalles.
Digite «help» para obtener ayuda.

postgres=# \c THG;
Ahora está conectado a la base de datos «THG» con el usuario «postgres».
THG=# CREATE TABLE grado_dificultad(
THG(# color varchar(8) PRIMARY KEY NOT NULL,
THG(# descripcion text NOT NULL
THG(# );
CREATE TABLE
THG=# CREATE TABLE puntuacion(
THG(# calificacion int PRIMARY KEY NOT NULL,
THG(# descripcion text NOT NULL
THG(# );
CREATE TABLE
```

Ilustración 4. Creación de las tablas Grado_Dificultad y Puntuación

El motivo por el que decidí comenzar por crear estas dos tablas es que, si nos fijamos en el diagrama de clases (*Ilustración 2*), notamos que estas no poseen ningún tipo de dependencia o relación con otras tablas, por lo que su creación e inserción de datos nos representarían ningún efecto colateral negativo para la BD.

DESARROLLO: POSTGRESQL

```
SQL Shell (psql)
THG=# INSERT INTO grado_dificultad VALUES ('Amarillo','Pruebas de peligro menor. No ponen en riesgo mortal al participante.');
```

INSERT 0 1

```
THG=# INSERT INTO grado_dificultad VALUES ('Naranja','Pruebas de peligro bajo. Ponen en ligero riesgo al participante.');
```

INSERT 0 1

```
THG=# INSERT INTO grado_dificultad VALUES ('Rojo','Pruebas de peligro intermedio. Ponen en riesgo considerable al participante.');
```

INSERT 0 1

```
THG=# INSERT INTO grado_dificultad VALUES ('Negro','Pruebas de peligro alto. Ponen en riesgo mortal al participante.');
```

INSERT 0 1

```
THG=# INSERT INTO puntuacion VALUES (1,'Muy malo');
```

INSERT 0 1

```
THG=# INSERT INTO puntuacion VALUES (2,'Malo');
```

INSERT 0 1

```
THG=# INSERT INTO puntuacion VALUES (3,'Muy Deficiente');
```

INSERT 0 1

```
THG=# INSERT INTO puntuacion VALUES (4,'Deficiente');
```

INSERT 0 1

```
THG=# INSERT INTO puntuacion VALUES (5,'Insatisfactorio');
```

INSERT 0 1

```
THG=# INSERT INTO puntuacion VALUES (6,'Aceptable');
```

INSERT 0 1

```
THG=# INSERT INTO puntuacion VALUES (7,'Satisfactorio');
```

INSERT 0 1

```
THG=# INSERT INTO puntuacion VALUES (8,'Bueno');
```

INSERT 0 1

```
THG=# INSERT INTO puntuacion VALUES (9,'Muy bueno');
```

INSERT 0 1

```
THG=# INSERT INTO puntuacion VALUES (10,'Sobresaliente');
```

INSERT 0 1

```
THG=# INSERT INTO puntuacion VALUES (11,'Excelente');
```

INSERT 0 1

```
THG=# INSERT INTO puntuacion VALUES (12,'Excepcional');
```

Ilustración 5. Inserción de datos a las tablas Grado_Dificultad y Puntuación

Creé un *type* llamado **name**, para emplearlo como un tipo de dato “ideal” para las columnas que requieran de un campo de tipo Nombre. También creé la tabla **Persona** que, por el tipo de enfoque que di a mi BD, funciona como una tabla “*padre*” que heredará sus columnas a tablas como **Líder**, **Mentor**, **Tributo** y **Vigilante** con ayuda del comando *INHERITS*.

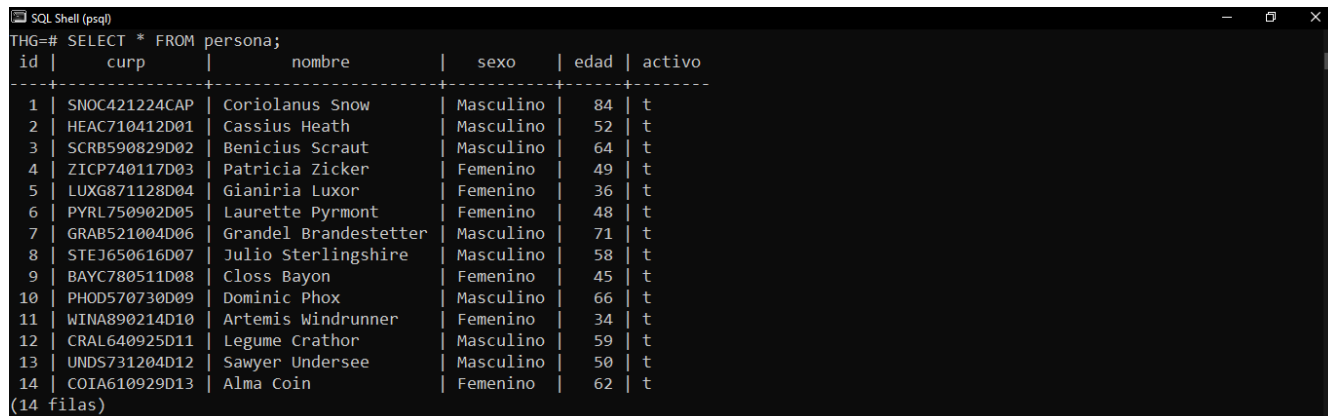
```
SQL Shell (psql)
THG=# CREATE TYPE name AS(
THG(# nombre text,
THG(# apellido text
THG(# );
CREATE TYPE
THG=# CREATE TABLE persona(
THG(# id serial PRIMARY KEY NOT NULL,
THG(# curp varchar(13) NOT NULL UNIQUE,
THG(# nombre name NOT NULL,
THG(# sexo varchar(9) NOT NULL,
THG(# edad int NOT NULL,
THG(# activo boolean NOT NULL
THG(# );
CREATE TABLE
THG=# CREATE TABLE lider(
THG(# id_lider varchar(4) NOT NULL UNIQUE
THG(#
THG(# ) INHERITS (persona);
CREATE TABLE
THG=# CREATE TABLE capitolio(
THG(# presidente_fk varchar(4) NOT NULL,
THG(# numero_habitantes int NOT NULL,
THG(# porcentaje_hombres decimal NOT NULL,
THG(# porcentaje_mujeres decimal NOT NULL,
THG(# ubicacion text NOT NULL,
THG(# clima text NOT NULL,
THG(# lugares_interes text NOT NULL,
THG(#
THG(# FOREIGN KEY (presidente_fk) REFERENCES lider(id_lider)
THG(# );
CREATE TABLE
THG=#
```

Ilustración 6. Creación del type Name y las tablas Persona y Líder y Capitolio

DESARROLLO: POSTGRESQL

La tabla **Persona** es a mi parecer, la tabla más importante de toda la BD, y es que es aquella con la que más adelante estableceré conexión en Netbeans para su interacción mediante la creación de nuestra Interfaz Gráfica de Java. A esta tabla también se le asignó una columna llama ID de tipo *serial* y otra columna llama Activo de tipo *boolean* con el propósito de que estos atributos nos serán de utilidad más adelante.

Algo a destacar es que, todos los *inserts* que se realicen en cualquiera de las tablas hijas se verán reflejados también en la tabla **Persona**, sólo que sin mostrar los atributos exclusivos de las tablas hijas. Por ejemplo, si consultamos los *inserts* de la tabla **Líder** desde la tabla **Persona**:



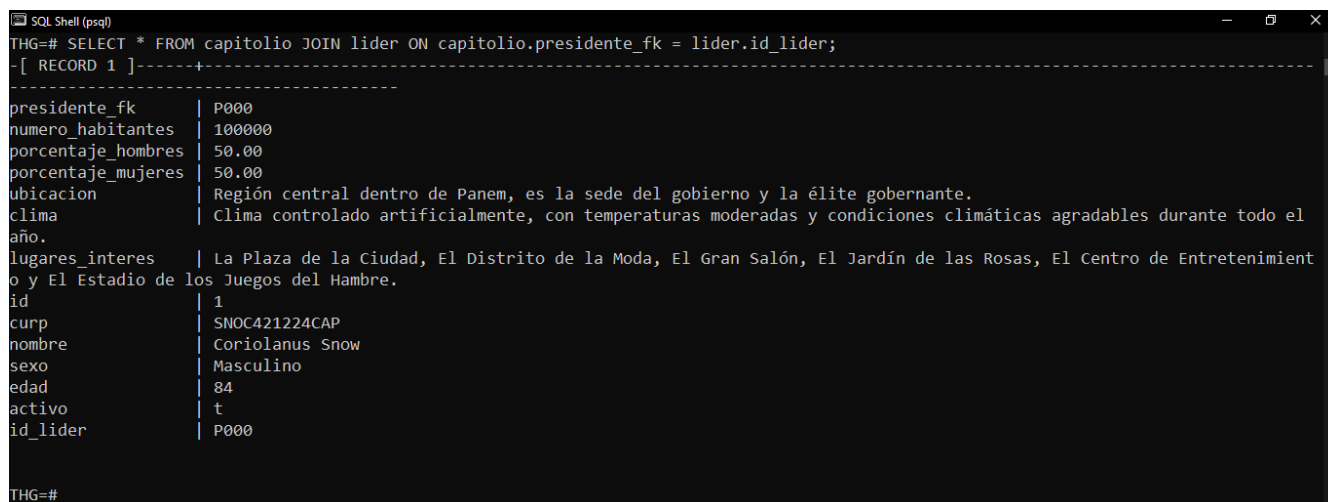
```
THG=# SELECT * FROM persona;
```

id	curp	nombre	sexo	edad	activo
1	SNOC421224CAP	Coriolanus Snow	Masculino	84	t
2	HEAC710412D01	Cassius Heath	Masculino	52	t
3	SCRB590829D02	Benicius Scraut	Masculino	64	t
4	ZICP740117D03	Patricia Zicker	Femenino	49	t
5	LUXG871128D04	Gianiria Luxor	Femenino	36	t
6	PYRL750902D05	Laurette Pyrmont	Femenino	48	t
7	GRAB521004D06	Grandel Brandestetter	Masculino	71	t
8	STEJ650616D07	Julio Sterlingshire	Masculino	58	t
9	BAYC780511D08	Closs Bayon	Femenino	45	t
10	PHOD570730D09	Dominic Phox	Masculino	66	t
11	WINA890214D10	Artemis Windrunner	Femenino	34	t
12	CRAL640925D11	Legume Crathor	Masculino	59	t
13	UNDS731204D12	Sawyer Undersee	Masculino	50	t
14	COIA610929D13	Alma Coin	Femenino	62	t

(14 filas)

Ilustración 7. Consulta de registros en la tabla Persona con el comando `SELECT * FROM`

Posteriormente creé la tabla **Capitolio**, en la cuál observamos nuestra primer *foreign key* que establece relación con nuestra tabla **Líder**, y es que el primer registro de nuestra tabla Líder le corresponde a Coriolanus Snow, quien a su vez es el presidente del Capitolio. A manera de corroborar si la relación fue establecida de manera correcta, hacemos uso del comando *JOIN* y validamos:



```
THG=# SELECT * FROM capitolio JOIN lider ON capitolio.presidente_fk = lider.id_lider;
```

- [RECORD 1] -	
presidente_fk	P000
numero_habitantes	100000
porcentaje_hombres	50.00
porcentaje_mujeres	50.00
ubicacion	Región central dentro de Panem, es la sede del gobierno y la élite gobernante.
clima	Clima controlado artificialmente, con temperaturas moderadas y condiciones climáticas agradables durante todo el año.
lugares_interes	La Plaza de la Ciudad, El Distrito de la Moda, El Gran Salón, El Jardín de las Rosas, El Centro de Entretenimiento y El Estadio de los Juegos del Hambre.
id	1
curp	SNOC421224CAP
nombre	Coriolanus Snow
sexo	Masculino
edad	84
activo	t
id_lider	P000

THG=#

Ilustración 8. Uso del comando *JOIN* para validar la relación entre las tablas Líder y Capitolio

Creé las tablas **Distrito**, **Mentor** y **Tributo** y agregué sus respectivos *inserts*, me parece importante recalcar que el orden en que realicé la creación e inserciones de dichas tablas fue hecho siguiendo la línea de relación y es que, para que la tabla **Distrito** fuera creada, primeramente debía existir **Líder**, puesto que de ella se obtenía la relación definida en su

DESARROLLO: POSTGRESQL

foreign key, de igual manera ocurría con la tabla **Mentor** que requería de **Distrito** y así sucesivamente, con **Tributo** que requería de **Distrito** y **Mentor**.

```
SQL Shell (psql)
THG=# CREATE TABLE distrito(
THG(# nombre varchar(2) PRIMARY KEY NOT NULL,
THG(# lider_fk varchar(4) NOT NULL,
THG(# especialidad text NOT NULL,
THG(# puestos_trabajo text NOT NULL,
THG(# numero_habitantes int NOT NULL,
THG(# porcentaje_hombres real NOT NULL,
THG(# porcentaje_mujeres real NOT NULL,
THG(# ubicacion text NOT NULL,
THG(# clima text NOT NULL,
THG(# juegos_ganados int NOT NULL,
THG(# controlado_por_capitolio bool,
THG(#
THG(# FOREIGN KEY (lider_fk) REFERENCES lider(id_lider)
THG(# );
CREATE TABLE
THG=# CREATE TABLE mentor(
THG(# id_mentor varchar(4) NOT NULL UNIQUE,
THG(# distrito_fk varchar(2) NOT NULL,
THG(# ed_juegos_ganada varchar(3) NOT NULL,
THG(#
THG(# FOREIGN KEY (distrito_fk) REFERENCES distrito(nombre)
THG(#
THG(# ) INHERITS (persona);
CREATE TABLE
THG=#
THG=# CREATE TABLE tributo(
THG(# id_tributo varchar(4) NOT NULL UNIQUE,
THG(# distrito_fk varchar(2) NOT NULL,
THG(# habilidad varchar(50) NOT NULL,
THG(# punt_espectaculo_fk int NOT NULL,
THG(# mentor_fk varchar(4) NOT NULL,
THG(#
THG(# FOREIGN KEY (distrito_fk) REFERENCES distrito(nombre),
THG(# FOREIGN KEY (punt_espectaculo_fk) REFERENCES puntuacion(calificacion),
THG(# FOREIGN KEY (mentor_fk) REFERENCES mentor(id_mentor)
THG(#
THG(# ) INHERITS (persona);
CREATE TABLE
THG=#
```

Ilustración 9. Creación de las tablas Distrito, Mentor y Tributo

Ahora bien, uno de los atributos que se nos pedían para la tabla **Distrito** era que existiera un registro de los tributos que le pertenecían, sin embargo, en primera instancia esta relación no podía ser creada ya que en ese momento aún no existía una tabla **Tributo** a la que referenciar, así que lo siguiente que hice fue hacer uso del comando *ALTER TABLE* para añadir dichas columnas a mi tabla **Distrito** y del comando *UPDATE* para asignarles valores a esos nuevos campos.

```
SQL Shell (psql)
THG=# ALTER TABLE distrito ADD tributo_f_fk varchar(4);
ALTER TABLE
THG=# ALTER TABLE distrito ADD FOREIGN KEY (tributo_f_fk) REFERENCES tributo(id_tributo);
ALTER TABLE
THG=#
THG=# ALTER TABLE distrito ADD tributo_m_fk varchar(4);
ALTER TABLE
THG=# ALTER TABLE distrito ADD FOREIGN KEY (tributo_m_fk) REFERENCES tributo(id_tributo);
ALTER TABLE
THG=# UPDATE distrito SET tributo_f_fk = 'T011' WHERE nombre = '1';
UPDATE 1
THG=# UPDATE distrito SET tributo_m_fk = 'T012' WHERE nombre = '1';
UPDATE 1
THG=# UPDATE distrito SET tributo_f_fk = 'T021' WHERE nombre = '2';
UPDATE 1
THG=# UPDATE distrito SET tributo_m_fk = 'T022' WHERE nombre = '2';
UPDATE 1
THG=# UPDATE distrito SET tributo_f_fk = 'T031' WHERE nombre = '3';
UPDATE 1
THG=# UPDATE distrito SET tributo_m_fk = 'T032' WHERE nombre = '3';
UPDATE 1
THG=# UPDATE distrito SET tributo_f_fk = 'T041' WHERE nombre = '4';
UPDATE 1
THG=# UPDATE distrito SET tributo_m_fk = 'T042' WHERE nombre = '4';
UPDATE 1
THG=# UPDATE distrito SET tributo_f_fk = 'T051' WHERE nombre = '5';
UPDATE 1
THG=# UPDATE distrito SET tributo_m_fk = 'T052' WHERE nombre = '5';
UPDATE 1
```

Ilustración 10. ALTER TABLE y UPDATE de la tabla Distrito

Añadí el *CONSTRAINT UNIQUE* a la columna ID de la tabla **Persona** porque me percaté que no había hecho esto anteriormente. Esto con el objeto de evitar ID repetidos que nos pudieran generar problemas más adelante a la hora de interactuar con la Interfaz. También creé las tablas **Vigilante**, **Prueba** y **Participante**, siendo esta última una “tabla relación” que

DESARROLLO: POSTGRESQL

conecta a la tabla **Tributo** con la tabla **Prueba**, esto se hace ya que, al ser una relación de muchos a muchos, se requiere de una tabla para esto.

Y de por último y de manera similar a cómo ocurría con las columnas Tributo_F y Tributo_M de la tabla **Distrito**, en la tabla **Prueba** requeríamos que existiera un registro del vencedor de dicha prueba, sin embargo, esta relación no podía ser creada si aún no existía una tabla **Participante** a la cual referenciar, así que nuevamente hice fue hacer uso de los comandos **ALTER TABLE** y **UPDATE** para realizar las modificaciones necesarias a la tabla **Prueba**.

```
SQL Shell (psql)

THG=# CREATE TABLE vigilante(
THG(# id_vigilante varchar(4) NOT NULL UNIQUE,
THG(# puesto varchar(20) NOT NULL
THG(#
THG(# ) INHERITS (persona);
CREATE TABLE
THG=# CREATE TABLE prueba(
THG(# codigo varchar(4) PRIMARY KEY NOT NULL,
THG(# nombre varchar(30) NOT NULL,
THG(# tipo varchar(20) NOT NULL,
THG(# grado_dificultad_fk varchar(8) NOT NULL,
THG(# fecha date NOT NULL,
THG(# evaluador_fk varchar(4) NOT NULL,
THG(# tiempo_empleado time NOT NULL,
THG(#
THG(# FOREIGN KEY (grado_dificultad_fk) REFERENCES grado_dificultad(color),
THG(# FOREIGN KEY (evaluador_fk) REFERENCES vigilante(id_vigilante)
THG(# );
CREATE TABLE
THG=# CREATE TABLE participante(
THG(# codigo_prueba varchar(4) NOT NULL,
THG(# id_participante varchar(4) NOT NULL,
THG(#
THG(# FOREIGN KEY (codigo_prueba) REFERENCES prueba(codigo),
THG(# FOREIGN KEY (id_participante) REFERENCES tributo(id_tributo)
THG(# );
CREATE TABLE
THG=# ALTER TABLE prueba ADD vencedor_fk varchar(4);
ALTER TABLE
THG=# ALTER TABLE prueba ADD FOREIGN KEY (vencedor_fk) REFERENCES tributo(id_tributo);
ALTER TABLE
THG=#
```

Ilustración 11. Creación de las tablas Vigilante, Prueba y Participante

Con esto finalizaría con la creación de mi Base de Datos, ya sólo para comprobar inspeccioné que todas las tablas fueran creadas correctamente:

```
THG=# \dt
Listado de relaciones
Esquema | Nombre | Tipo | Dueño
-----|-----|-----|-----
public | capitolio | tabla | postgres
public | distrito | tabla | postgres
public | grado_dificultad | tabla | postgres
public | lider | tabla | postgres
public | mentor | tabla | postgres
public | participante | tabla | postgres
public | persona | tabla | postgres
public | prueba | tabla | postgres
public | puntuacion | tabla | postgres
public | tributo | tabla | postgres
public | vigilante | tabla | postgres
(11 filas)
```

Ilustración 12. Desglose de todas las tablas existentes en la base de datos THG

DESARROLLO: NETBEANS

Para comenzar con la creación de mi Interfaz en Java, lo primordial era crear un *script* que fuera capaz de hacer la conexión entre Netbeans y mi BD de PostgreSQL. Para esto se creó un *Java Class* al que llamé **MiBD**:

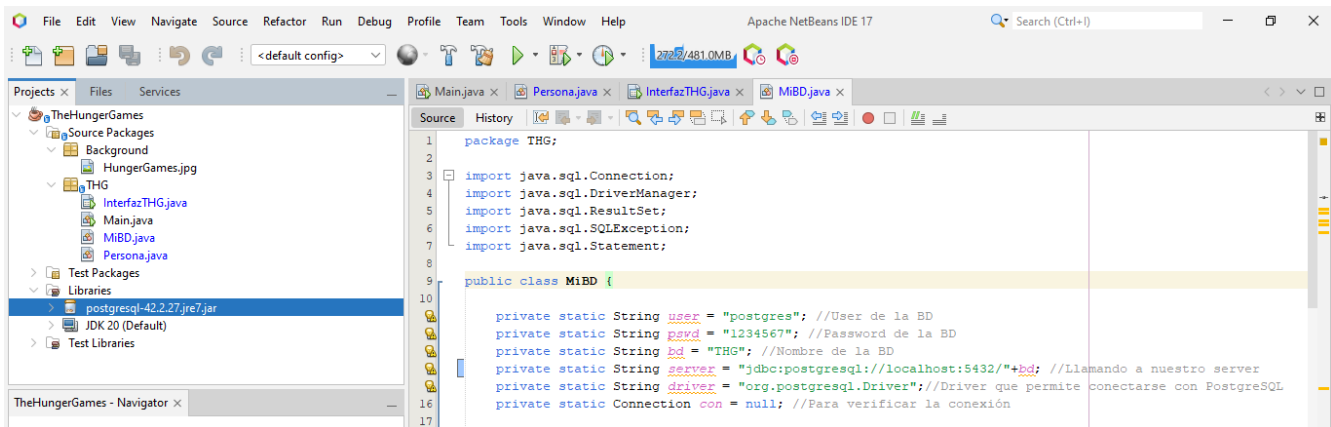


Ilustración 13. Java Class MiBD para la conexión a la base de datos

Primeramente, se realiza el correspondiente *import* de todas las librerías y métodos a citar en el código, también se crearon variables para cada dato necesario para la conexión, como lo son el usuario, contraseña, nombre de la base de datos (en este caso, THG) y el servidor de PostgreSQL; para el caso del driver, por default Netbeans cuenta con el JDK 20, pero para este caso se añadió el *JAR* de *postgresql-42.2.27.jre7*, el cual no necesariamente es el más actualizado, pero sí posee ya cierta estabilidad debido a su antigüedad.

Para el caso del objeto *con*, este se inicializa como un *null* de manera que cuando en **public MiBD()** le mandemos a llamar y asignemos con el *getConnection* los valores de user, pswd y server, detecte que si ahora esta conexión a dejado de ser “nula”, para así mostrar un mensaje de “La conexión a la BD se realizó al 100%” y en caso contrario, muestre un mensaje de error.

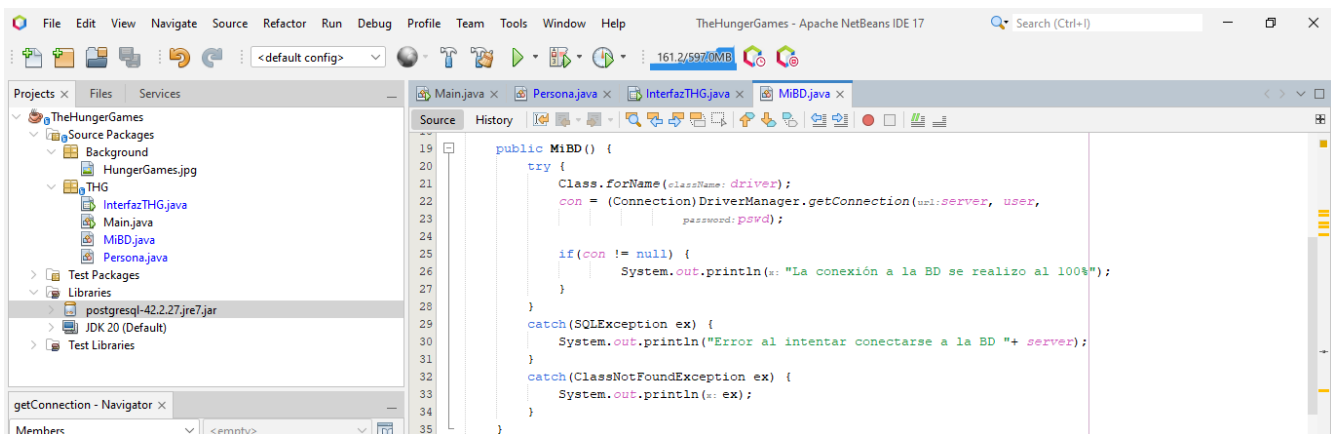


Ilustración 14. Comprobación de la conexión en MiBD

Después de establecida la conexión, lo siguiente que hice fue crear un *Java Bean* para mi tabla **Persona**, de manera que todos los componentes de mi tabla se “encapsularan” en un único objeto que pudiese ser llamado en otras clases. Lo curioso de esto es que lo anterior lo hice, sin yo saber, que seguía a un fundamento de la Programación Orientada a Objetos.

DESARROLLO: NETBEANS

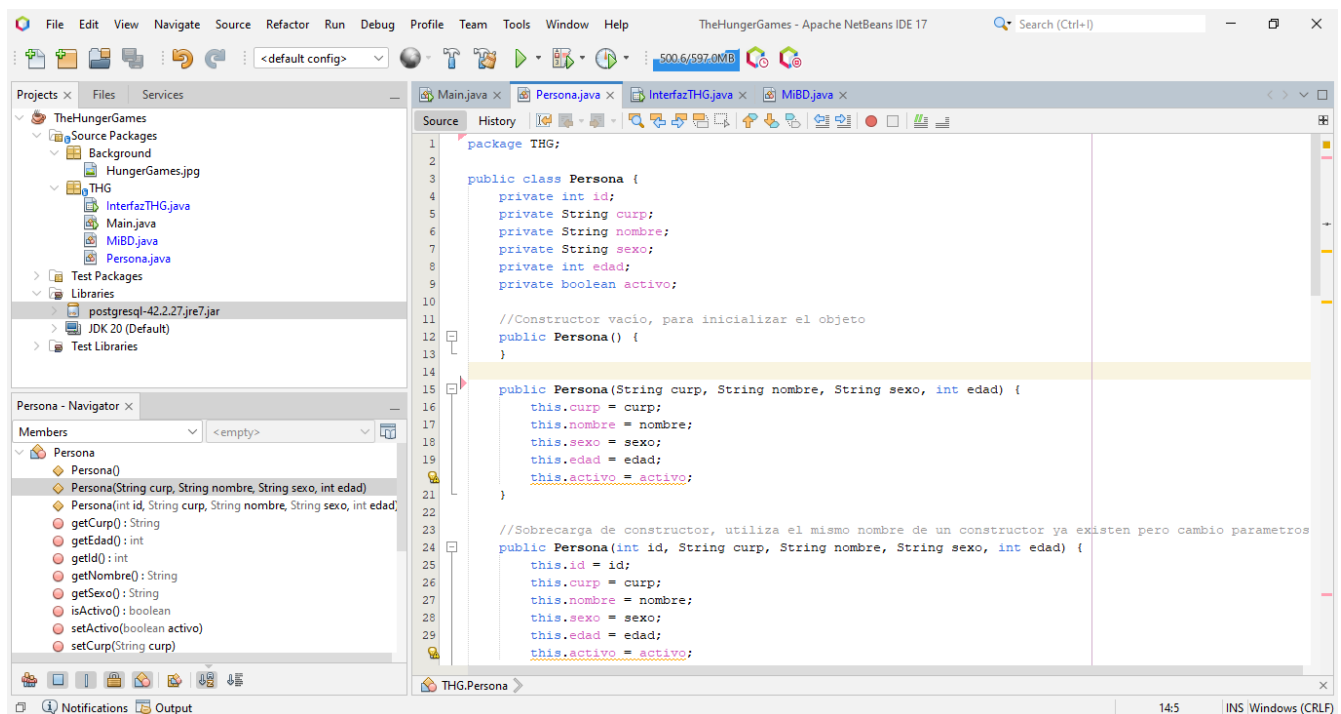


Ilustración 15. Creación del Java Bean Persona

Entonces, realicé mis respectivos *imports*, declaré mis variables privadas de manera que sólo se pudiera acceder a ellas a través de métodos, creé un *constructor vacío* que las inicializara y después agregué dos constructores que contenían a esas variables, creando así una *sobrecarga de constructor*.

Creé un *getter* y *setter* para cada una de estas variables y finalicé con un *String* que me imprimiese toda esta información en forma de cadena.

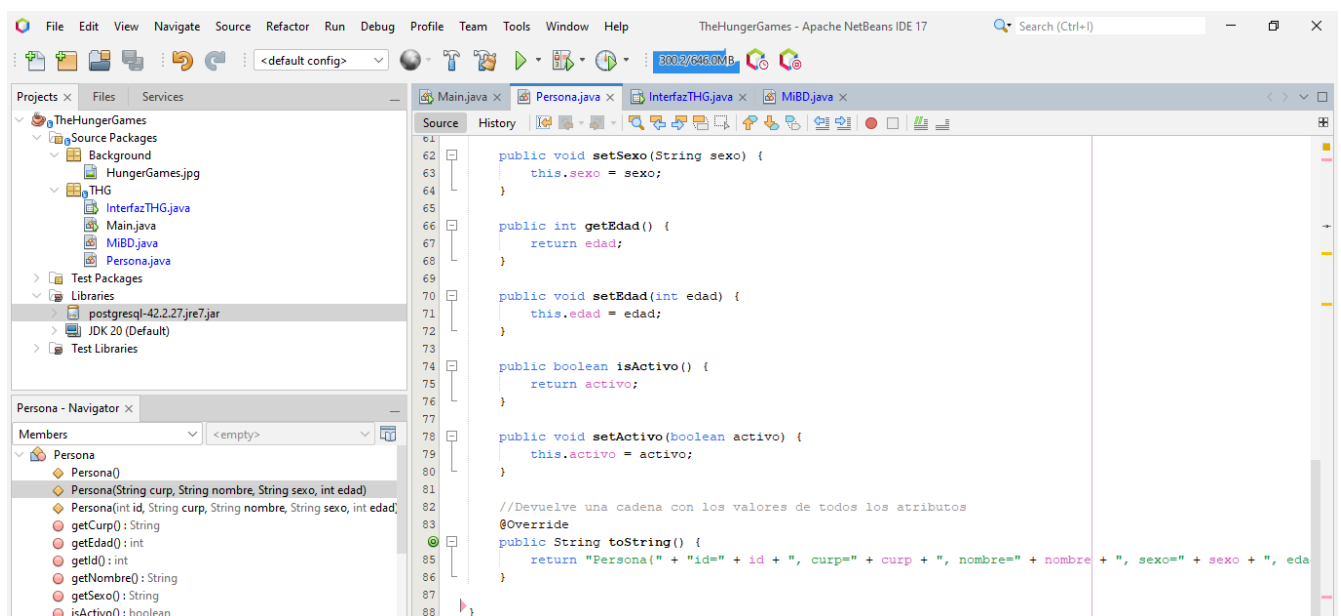


Ilustración 16. Getters y Setters para cada variable en Persona

Mi clase **Main** lo que realiza es únicamente una instancia llamada conexión de mi clase **MiBD** de la conexión, emplea el método *ResultSet* y luego de establecer la conexión, manda

DESARROLLO: NETBEANS

con el *getQuery* el comando de **SELECT * FROM persona**, después dentro de un *while* se crea una instancia llamada persona de mi clase **Persona** y se obtiene la información contenida en cada una de las columnas de mi tabla Persona, para finalmente imprimirlas en un *String*.

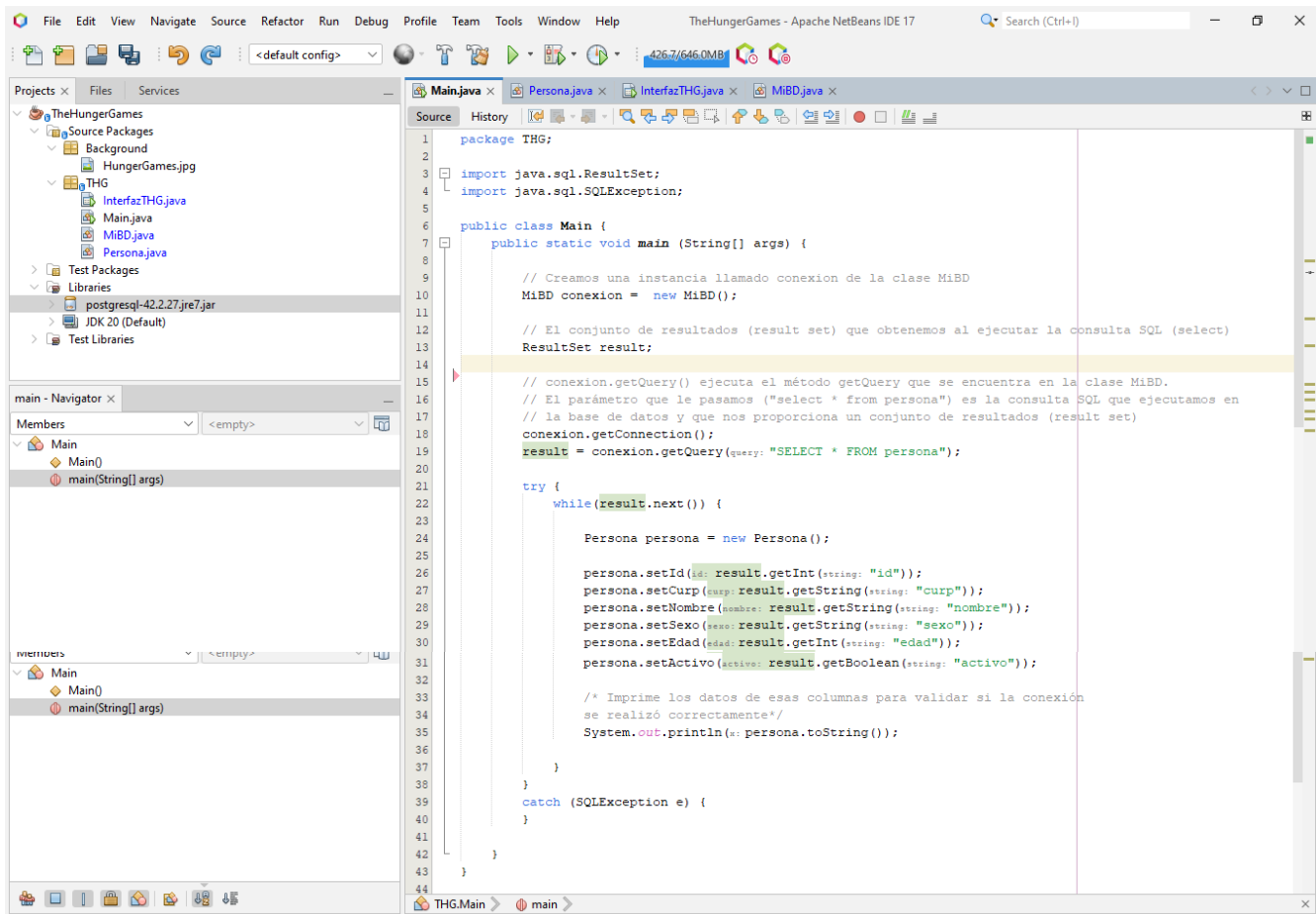


Ilustración 17. Clase Main

Finalmente, en un *Java Class* que nombré como **InterfazTHG**, realicé todo lo referente a la Interfaz que interactuaría con el usuario. Para esto, en primera instancia y luego de varios diseños preliminares y mucha prueba y error, se optó por un diseño como el siguiente:

ID	CURP	Nombre	Sexo	Edad
----	------	--------	------	------

Ilustración 18. Diseño Preliminar de la InterfazTHG

DESARROLLO: NETBEANS

En cuanto a código tenemos los infalibles *imports*, una clase llamada **InterfazTHG** que extiende al *JFrame*, declara objetos para las clases *MiBD*, *Connection*, *PreparedStatement* y *ResultSet* que posteriormente son inicializados en el constructor.

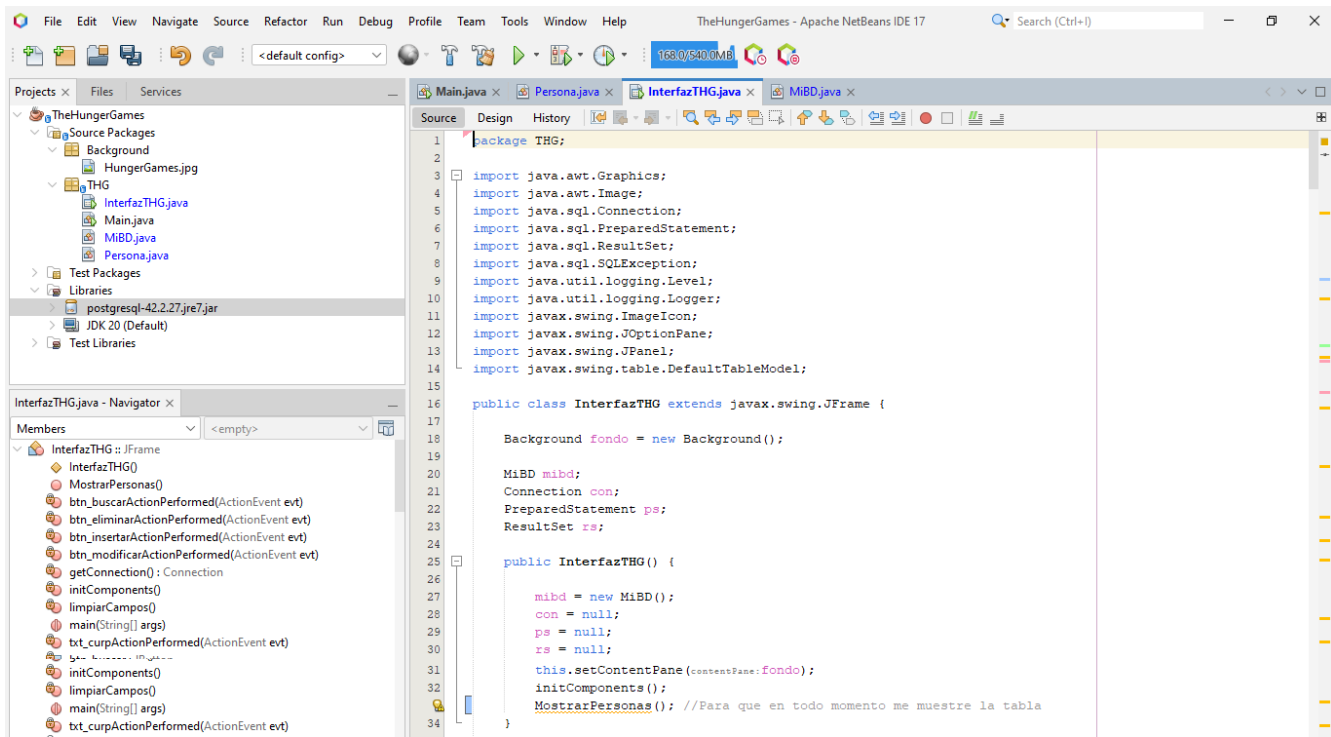


Ilustración 19. Clase InterfazTHG

En *MostrarPersonas* lo que hacemos es definir el comportamiento que tendrá la tabla que vimos previamente en el diseño de la interfaz, le damos la instrucción de que nos muestre a todas las personas cuyo estado sea activo y que estén ordenados según su ID. Efectúa la consulta, con el *if* lo que evitamos es que se dupliquen las columnas en cada iteración y con el objeto *data* obtenemos la información de cada columna de *Persona* y la acomodamos en una fila.

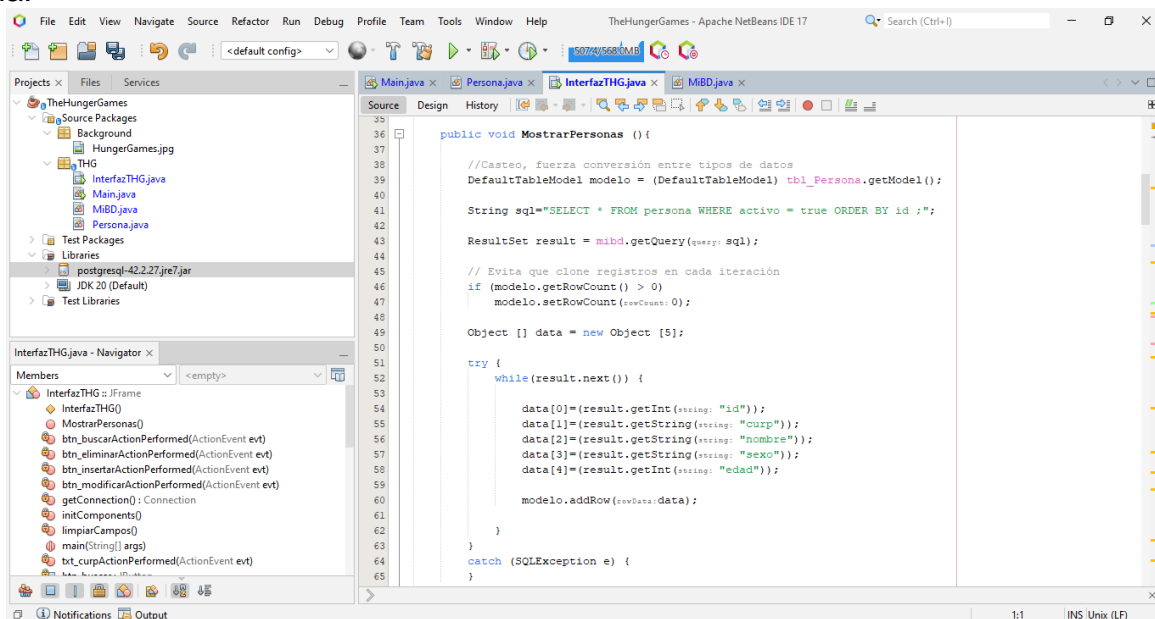


Ilustración 20. *MostrarPersonas* contiene toda la información de la tabla mostrada en la Interfaz

DESARROLLO: NETBEANS

En la línea “comprimida” que dice Generated Code tenemos a nuestros initComponents, los cuales son creados de manera automática cada que creamos algún elemento en el diseño de nuestra interfaz y no pueden ser manipulados manualmente.

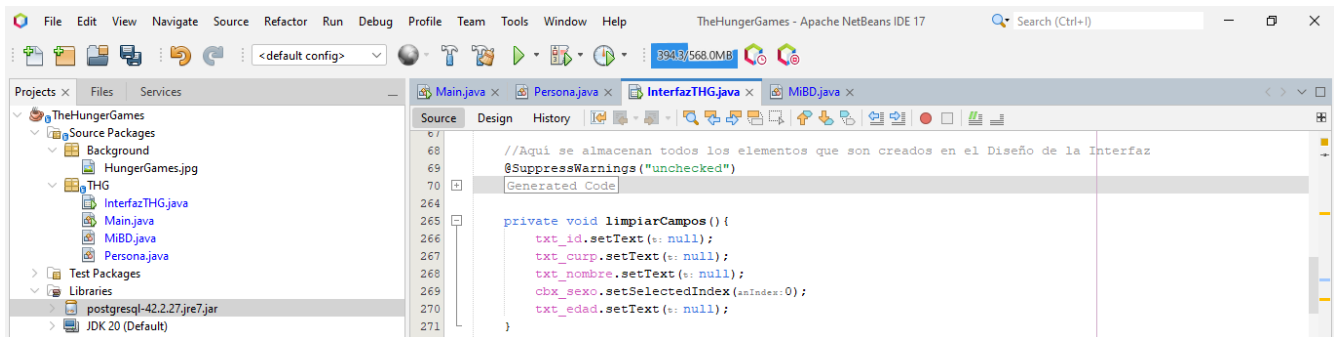


Ilustración 21. initComponents y acciones de limpiarCampos

En limpiarCampos(), lo que hacemos es definir el valor de cada uno de mis TextFields a null, de esta manera borrando el contenido de estos cuando sea requerido.

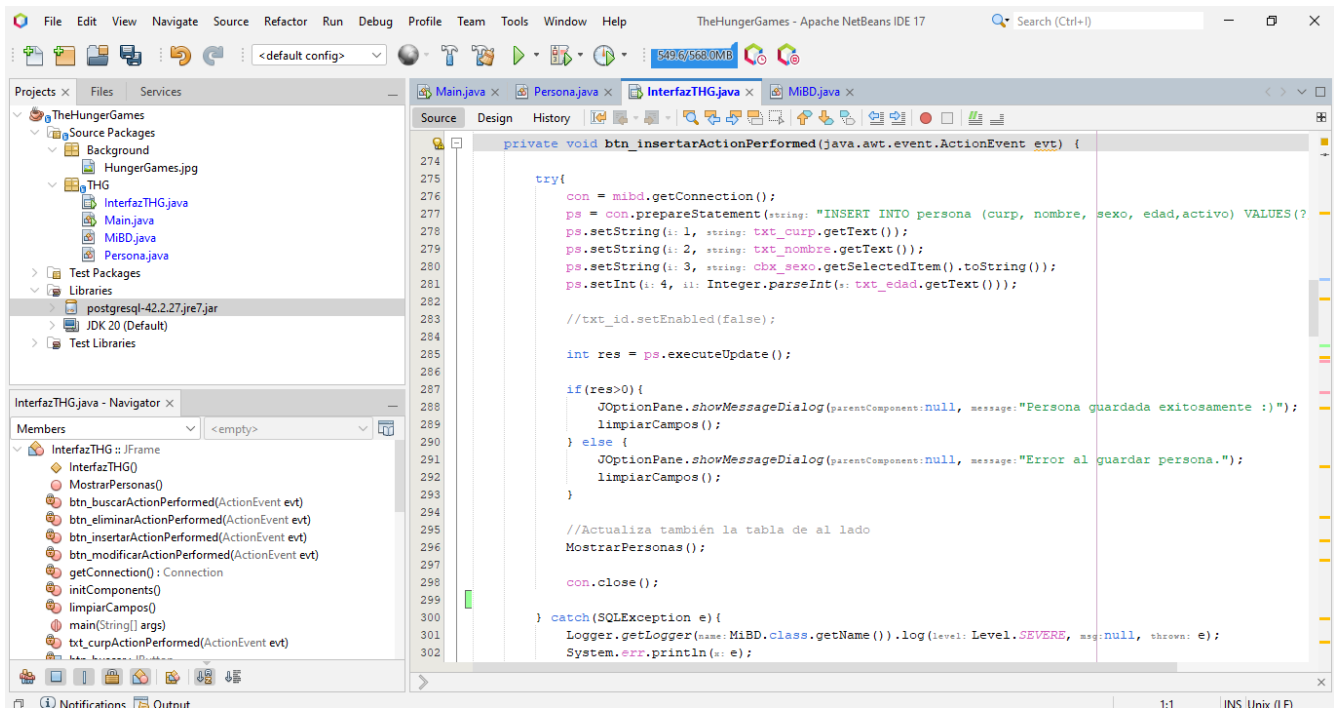


Ilustración 22. Acciones del botón Insertar de la Interfaz

Mi primer botón de acción que es el de *Insertar*, lo que hace es, establecer la conexión a **MiBD**, posteriormente con el *prepareStatement* manda el comando SQL de INSERT INTO persona los valores que son extraídos de los *TextFields* y que son llenados por el usuario. En res los almacenamos y si detecta que *res>0*, osea que sí posee información puntual, muestra un mensaje de confirmación y en caso contrario, un mensaje de error. En ambos casos, se manda a llamar a **limpiarCampos()** para que una vez realizada la acción nos libere los *TextFields*. También llamamos a **MostrarPersonas()** para que dicho cambio se vea reflejado en la tabla.

DESARROLLO: NETBEANS

Para los casos de los botones de *Buscar* y *Modificar*, se siguió el mismo procedimiento, simplemente cambiando los comandos SQL correspondientes para cada caso. Siendo *Buscar* un **SELECT * FROM persona** y *Modificar* un **UPDATE persona SET ____**. Ambos también requieren en primera instancia de que el usuario les proporcione el ID de la persona que se desea buscar y/o modificar.

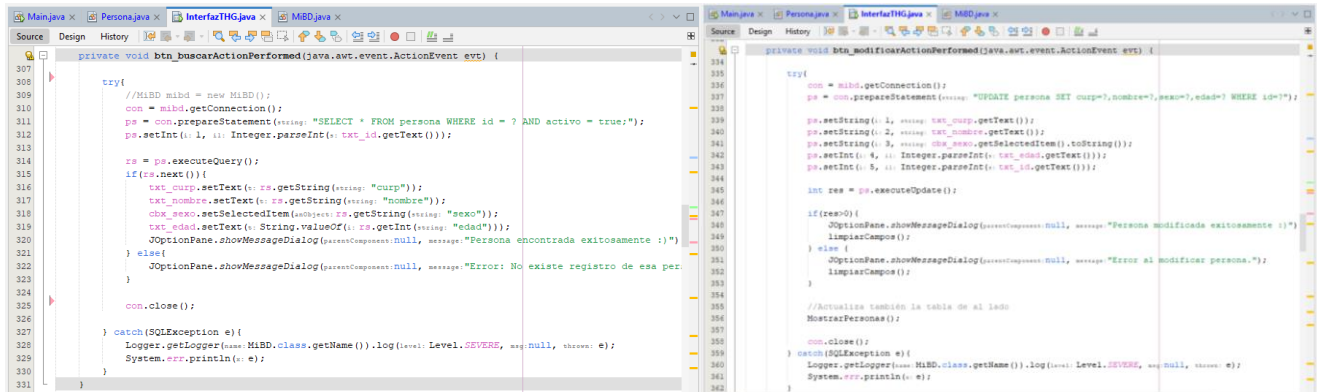


Ilustración 23. Botones de Buscar y Modificar en la Interfaz

Para el caso particular de mi botón de Eliminar, dado el análisis que realicé a mis tablas, opté por no implementar un eliminado del registro como tal para no generar problemas en mi BD dada la cantidad de herencias y dependencias que mi tabla Persona posee. Así que, es aquí donde mi atributo Activo entra en juego, y es que mi botón de Eliminar lo que hace es un **UPDATE persona SET activo = false WHERE id=___**, de manera que sólo cambiamos el estatus de este registro y, como tanto mi botón buscar como mi tabla, implementan una condición de sólo mostrar a los registros cuyo estatus sea activo, entonces al cambiar el estatus de éste, deja al usuario incapaz de visualizar a ese registro.

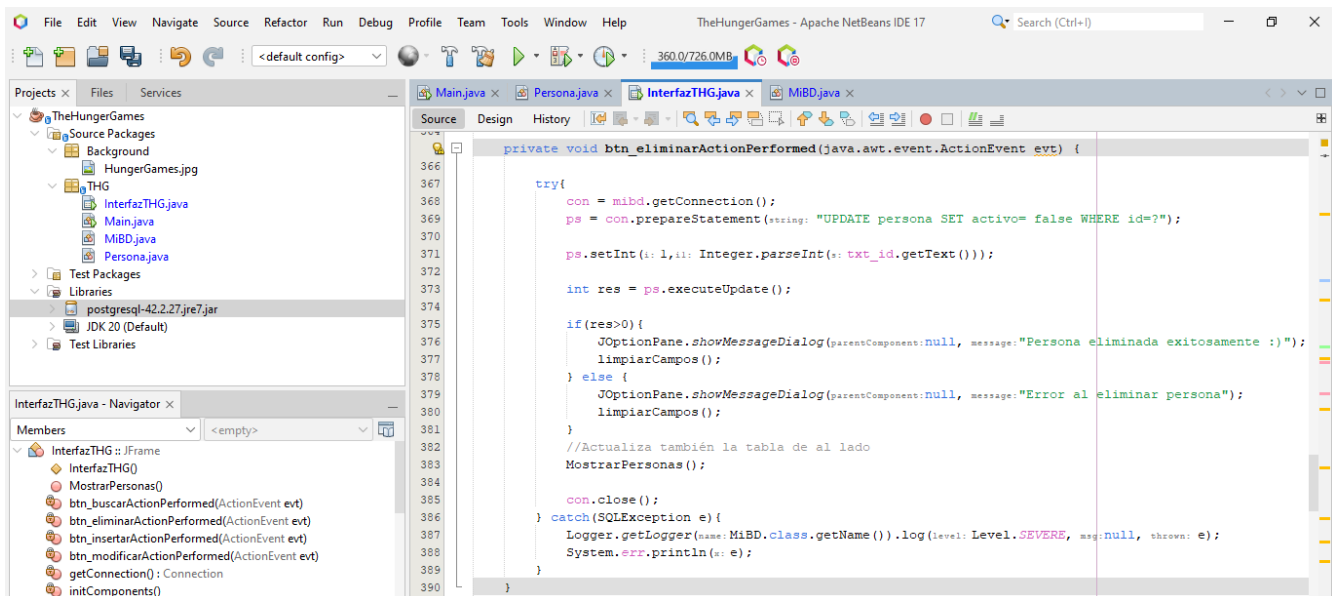


Ilustración 24. Acciones del botón Eliminar en la Interfaz

Finalmente, creé una clase llamada **Background** que se encargaba de asignar a mi fondo de la Interfaz, una imagen que yo diseñé. Para esto requerí de crear un paquete al que llamé Background y al cual le agregué la imagen en cuestión que es obtenida mediante el método *ImageIcon*.

DESARROLLO: NETBEANS

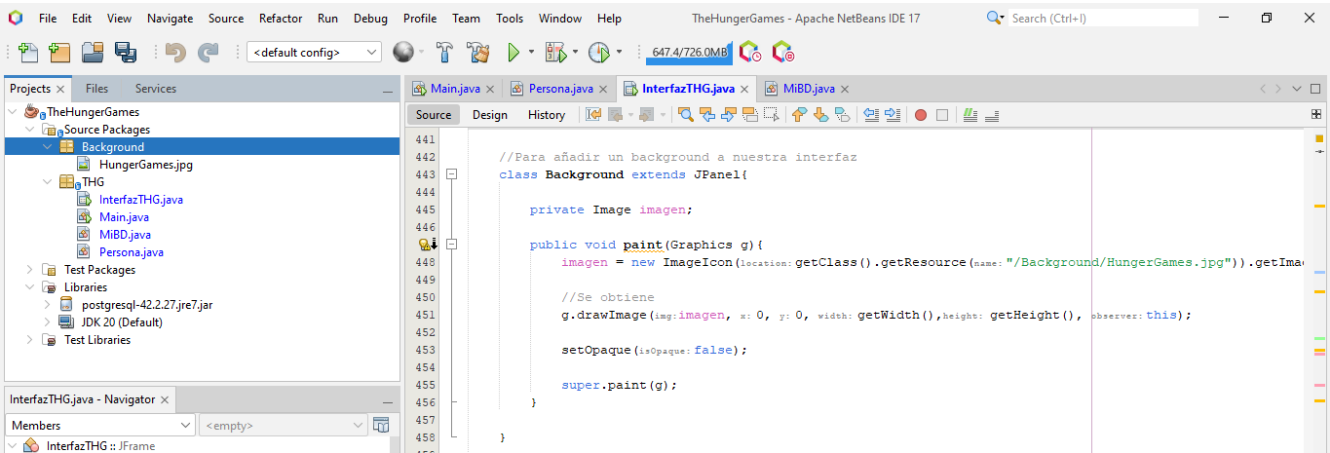


Ilustración 25. Clase que define la imagen de fondo mostrada en la Interfaz

La vista final de mi Interfaz es la siguiente:

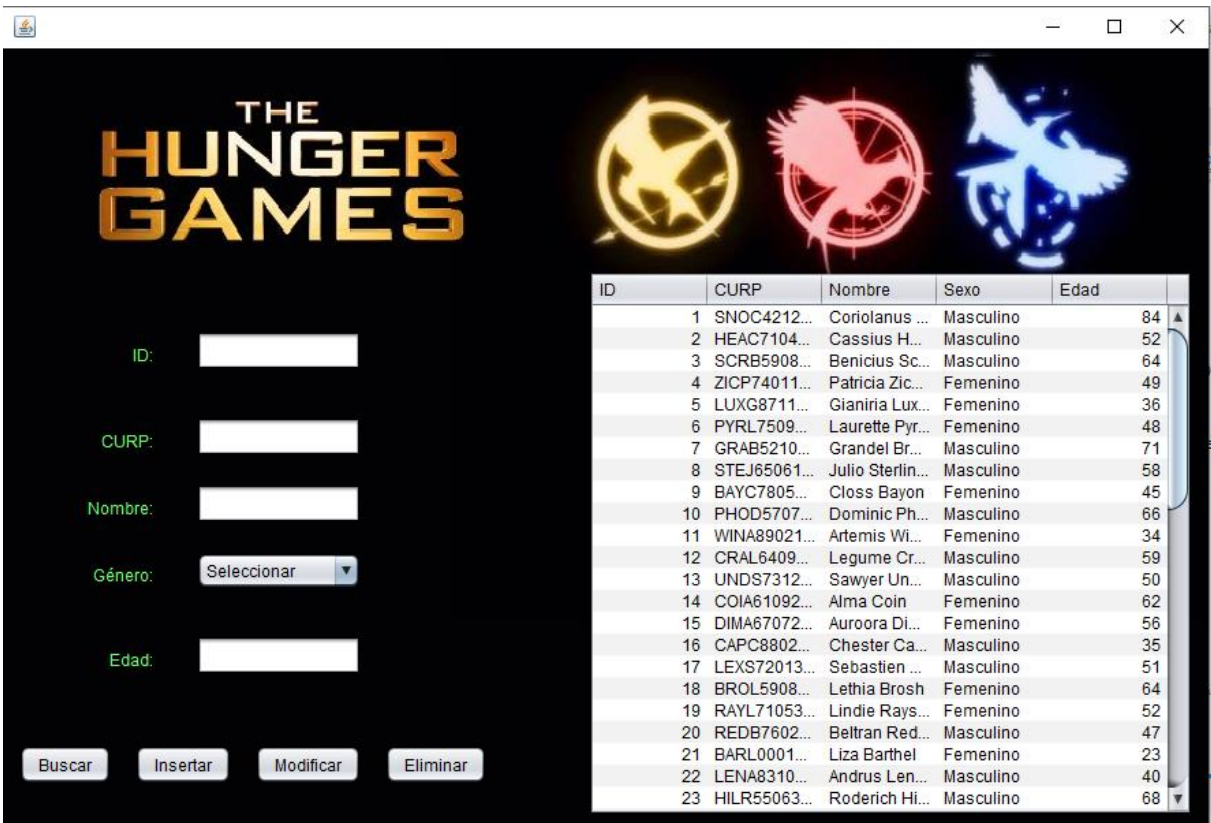


Ilustración 26. Diseño final de la Interfaz

ANEXOS

SQL Shell (psql)

```
THG=# SELECT * FROM tributo;
```

id	curp	nombre	sexo	edad	id_tributo	distrito_fk	habilidad	punt_espectaculo_fk	mentor_fk	activo
27	VOMG060214D01	Glimmer VonLuxe	Femenino	17	T011	1	Espada	10	M001	t
28	HAWM060325D01	Marvel Hawthorne	Masculino	17	T012	1	Daga	9	M001	t
29	SINC081009D02	Clove Sinclair	Femenino	15	T021	2	Hacha	10	M002	t
30	STEC050519D02	Cato Sterling	Masculino	18	T022	2	Guadaña	10	M002	t
31	MONL100308D03	Lucy Montrose	Femenino	13	T031	3	Lanza	6	M003	t
32	AURI090119D03	Ian Aurelius	Masculino	14	T032	3	Minas	7	M003	t
33	MACI070929D04	Tara Macken	Femenino	16	T041	4	Espada con diente	6	M004	t
34	ROS811105D04	Breck Rosenthal	Masculino	12	T042	4	Puñal	5	M004	t
35	BLAF080930D05	Foxface Blackwood	Femenino	15	T051	5	Cuchillo	8	M005	t
36	MARA080910D05	Alexander Mark	Masculino	15	T052	5	Hoz	7	M005	t
37	PETK081201D06	Kara Petersen	Femenino	15	T061	6	Arco y Flecha	7	M006	t
38	MOIA071031D06	Ashton Moio	Masculino	16	T062	6	Combate cuerpo a cuerpo	7	M006	t
39	HANL060627D07	Leigha Hancock	Femenino	17	T071	7	Lanzar cuchillos	7	M007	t
40	LEES060207D07	Sam Lee	Masculino	17	T072	7	Machete	9	M007	t
41	LINM080802D08	Mackenzie Lintz	Femenino	15	T081	8	Lanza	6	M008	t
42	TANS091121D08	Samuel Tan	Masculino	14	T082	8	Cuchillo	7	M008	t
43	THUA090503D09	Annie Thurman	Femenino	14	T091	9	Espada	6	M009	t
44	FREI091001D09	Imanol Freeman	Masculino	14	T092	9	Hacha	5	M009	t
45	HOOD071014D10	Dakota Hood	Femenino	16	T101	10	Daga	5	M010	t
46	MARJ050729D10	Jeremy Marinas	Masculino	18	T102	10	Fuerza Física	7	M010	t
47	HOLR111204D11	Rue Holmes	Femenino	12	T111	11	Honda y Piedra	7	M011	t
48	OKET050612D11	Tresh Oken	Masculino	18	T112	11	Fuerza Bruta	7	M011	t
49	EVEK070508D12	Katniss Everdeen	Femenino	16	T121	12	Arco y Flecha	11	M012	t
50	MELP070830D12	Peeta Mellark	Masculino	16	T122	12	Camuflaje	8	M012	t

(24 filas)

Ilustración 27. Registros de la tabla Tributos de la BD

SQL Shell (psql)

```
THG=# SELECT * FROM lider;
```

id	curp	nombre	sexo	edad	id_lider	activo
1	SNOG421224CAP	Coriolanus Snow	Masculino	84	P000	t
2	HEAC7104112D01	Cassius Heath	Masculino	52	L001	t
3	SCR8590829D02	Benicius Scraut	Masculino	64	L002	t
4	ZICP740117D03	Patricia Zicker	Femenino	49	L003	t
5	LUXG871128D04	Gianiria Luxor	Femenino	36	L004	t
6	PYRL750902D05	Laurette Pyrmont	Femenino	48	L005	t
7	GRAB521004D06	Grandel Brandestetter	Masculino	71	L006	t
8	STEJ650616D07	Julio Sterlingshire	Masculino	58	L007	t
9	BAYC780511D08	Closs Bayon	Femenino	45	L008	t
10	PHOD570730D09	Dominic Phox	Masculino	66	L009	t
11	WINA890214D10	Artemis Windrunner	Femenino	34	L010	t
12	CRAL640925D11	Legume Crathor	Masculino	59	L011	t
13	UNDS731204D12	Sawyer Undersee	Masculino	50	L012	t
14	COIA610929D13	Alma Coin	Femenino	62	L013	t

(14 filas)

```
THG=# SELECT * FROM vigilante;
```

id	curp	nombre	sexo	edad	id_vigilante	puesto	activo
51	LYMA700319V01	Alejandro Lynthal	Masculino	50	V001	Seguridad	t
52	KASR001030V02	Renie Kasting	Masculino	23	V002	Militar	t
53	TROB890119V03	Bobbie Trorey	Masculino	34	V003	Guardia	t
54	CRAS781203V04	Seneca Crane	Masculino	45	V004	Seguridad	t
55	HEAP740507V05	Plutarch Heavensbee	Masculino	49	V005	Militar	t
56	CHEH950428V06	Hervey Cheine	Masculino	28	V006	Guardia	t

(6 filas)

Ilustración 28. Registros de las tablas Líder y Vigilantes de la BD

SQL Shell (psql)

```
THG=# SELECT * FROM mentor;
```

id	curp	nombre	sexo	edad	id_mentor	distrito_fk	ed_juegos_ganada	activo
15	DIMA670729D01	Auroopa Dimond	Femenino	56	M001	1	67°	t
16	CAPC880217D02	Chester Capin	Masculino	35	M002	2	73°	t
17	LEXS720131D03	Sebastien Lexa	Masculino	51	M003	3	54°	t
18	BROL590827D04	Lethia Brosh	Femenino	64	M004	4	33°	t
19	RAYL710531D05	Lindie Rayson	Femenino	52	M005	5	59°	t
20	REDB760228D06	Beltran Reder	Masculino	47	M006	6	57°	t
21	BARL000110D07	Liza Barthel	Femenino	23	M007	7	72°	t
22	LENA831024D08	Andrus Lenchenko	Masculino	40	M008	8	58°	t
23	HTLR550630D09	Roderich Hilary	Masculino	68	M009	9	42°	t
24	DUMH670419D10	Hardis Dumbilton	Masculino	56	M010	10	68°	t
25	PRES800329D11	Sky Pretious	Femenino	43	M011	11	45°	t
26	ABEH831201D12	Haymicht Abernathy	Masculino	40	M012	12	50°	t

(12 filas)

```
THG=# SELECT * FROM grado_dificultad;
```

color	descripcion
Amarillo	Pruebas de peligro menor. No ponen en riesgo mortal al participante.
Naranja	Pruebas de peligro bajo. Ponen en ligero riesgo al participante.
Rojo	Pruebas de peligro intermedio. Ponen en riesgo considerable al participante.
Negro	Pruebas de peligro alto. Ponen en riesgo mortal al participante.

(4 filas)

Ilustración 29. Registros de las tablas Mentor y Grado_Dificultad de la BD

ANEXOS

SQL Shell (psql)

```
THG=# SELECT * FROM prueba;
```

codigo	nombre	tipo	grado_dificultad_fk	fecha	evaluador_fk	tiempo_employado	vencedor_fk
PR01	Manejo de armas	Combate	Negro	2023-04-03	V001	02:32:51	T012
PR02	Escalada	Supervivencia	Amarillo	2023-04-09	V002	00:57:49	T041
PR03	Camuflaje	Supervivencia	Amarillo	2023-04-15	V003	04:08:17	T062
PR04	Caza	Supervivencia	Rojo	2023-04-27	V004	06:20:24	T081
PR05	Tiro con arco	Combate	Naranja	2023-05-01	V005	03:16:01	T101
PR06	Combate cuerpo a cuerpo	Combate	Naranja	2023-05-30	V006	01:18:04	T122

(6 filas)

```
THG=# SELECT * FROM puntuacion;
```

calificacion	descripcion
1	Muy malo
2	Malo
3	Muy Deficiente
4	Deficiente
5	Insatisfactorio
6	Aceptable
7	Satisfactorio
8	Bueno
9	Muy bueno
10	Sobresaliente
11	Excelente
12	Excepcional

(12 filas)

Ilustración 30. Registros de las tablas Prueba y Puntuación de la BD

SQL Shell (psql)

```
THG=# SELECT * FROM participante;
```

codigo_prueba	id_participante
PR01	T011
PR01	T012
PR01	T021
PR01	T022
PR02	T031
PR02	T032
PR02	T041
PR02	T042
PR03	T051
PR03	T052
PR03	T061
PR03	T062
PR04	T071
PR04	T072
PR04	T081
PR04	T082
PR05	T091
PR05	T092
PR05	T101
PR05	T102
PR06	T111
PR06	T112
PR06	T121
PR06	T122

(24 filas)

Ilustración 31. Registros de la tabla Participante de la BD

Seleccionar SQL Shell (psql)

```
THG=# SELECT * FROM capitolio;
```

[RECORD 1]	
presidente_fk	P000
numero_habitantes	100000
porcentaje_hombres	50.00
porcentaje_mujeres	50.00
ubicacion	Región central dentro de Panem, es la sede del gobierno y la élite gobernante.
clima	Clima controlado artificialmente, con temperaturas moderadas y condiciones climáticas agradables durante todo el año.
lugares_interes	La Plaza de la Ciudad, El Distrito de la Moda, El Gran Salón, El Jardín de las Rosas, El Centro de Entretenimiento y El Estadio de los Juegos del Hambre.

Ilustración 32. Registros de la tabla Capitolio de la BD

ANEXOS

SQL Shell (psql)	
THG=# SELECT * FROM distrito;	
-[RECORD 1]-----	
nombre	13
lider_fk	L013
especialidad	Industria de energía nuclear y de armas.
puestos_trabajo	Ingenieros nucleares, Científicos de investigación, Agricultores hidropónicos y Líderes de resistencia.
numero_habitantes	41000
porcentaje_hombres	70.98
porcentaje_mujeres	29.02
ubicacion	Región subterránea secreta en ubicación desconocida del continente de Panem.
clima	Clima controlado artificialmente, clima templado constante.
juegos_ganados	0
controlado_por_capitolio	f
tributo_f_fk	
tributo_m_fk	
-[RECORD 2]-----	
nombre	1
lider_fk	L001
especialidad	Industria de lujo y productos de alta gama.
puestos_trabajo	Joyería fina, Diseño de moda, Fabricación de productos de lujo y Artesanía de calidad.
numero_habitantes	89000
porcentaje_hombres	46.32
porcentaje_mujeres	53.68
ubicacion	Región noroeste de Panem.
clima	Clima templado con estaciones bien definidas. Inviernos suaves y veranos cálidos.
juegos_ganados	11
controlado_por_capitolio	t
tributo_f_fk	T011
tributo_m_fk	T012
-[RECORD 3]-----	
nombre	6
lider_fk	L006
especialidad	Industria de Transporte y Logística.
puestos_trabajo	Conductores de trenes y camiones, Pilotos de aviones, Técnicos de mantenimiento de vehículos y Planificadores de logística.
numero_habitantes	81000
porcentaje_hombres	60.29
porcentaje_mujeres	39.71
ubicacion	Región montañosa de Panem central.
clima	Clima continental templado con inviernos fríos y veranos suaves.
juegos_ganados	4
controlado_por_capitolio	t
tributo_f_fk	T061
tributo_m_fk	T062
-[RECORD 4]-----	
nombre	2
lider_fk	L002
especialidad	Industria producción militar y construcción.
puestos_trabajo	Entrenadores militares, Fabricación de armas y equipos de defensa, Ingeniería de fortificaciones y Construcción de infraestructuras defensiva
s.	
numero_habitantes	73000
porcentaje_hombres	66.28
porcentaje_mujeres	33.72
ubicacion	Región montañosa del sur de Panem.
clima	Clima montañoso con inviernos fríos y veranos frescos.
juegos_ganados	12
controlado_por_capitolio	t
tributo_f_fk	T021
tributo_m_fk	T022
-[RECORD 5]-----	
nombre	11
lider_fk	L011
especialidad	Industria agrícola y productora de alimentos.
puestos_trabajo	Agricultores, Trabajadores agrícolas y Trabajadores en la industria alimentaria.
numero_habitantes	87000
porcentaje_hombres	61.1
porcentaje_mujeres	38.9
ubicacion	Región agrícola en el oeste de Panem.
clima	Clima tropical con estaciones húmedas y secas. Altas temperaturas durante todo el año.
juegos_ganados	5
controlado_por_capitolio	t
tributo_f_fk	T111
tributo_m_fk	T112

Ilustración 33. Registros de la tabla Distrito de la BD