Base de datos ASDgene

Murzi, Ana Sol

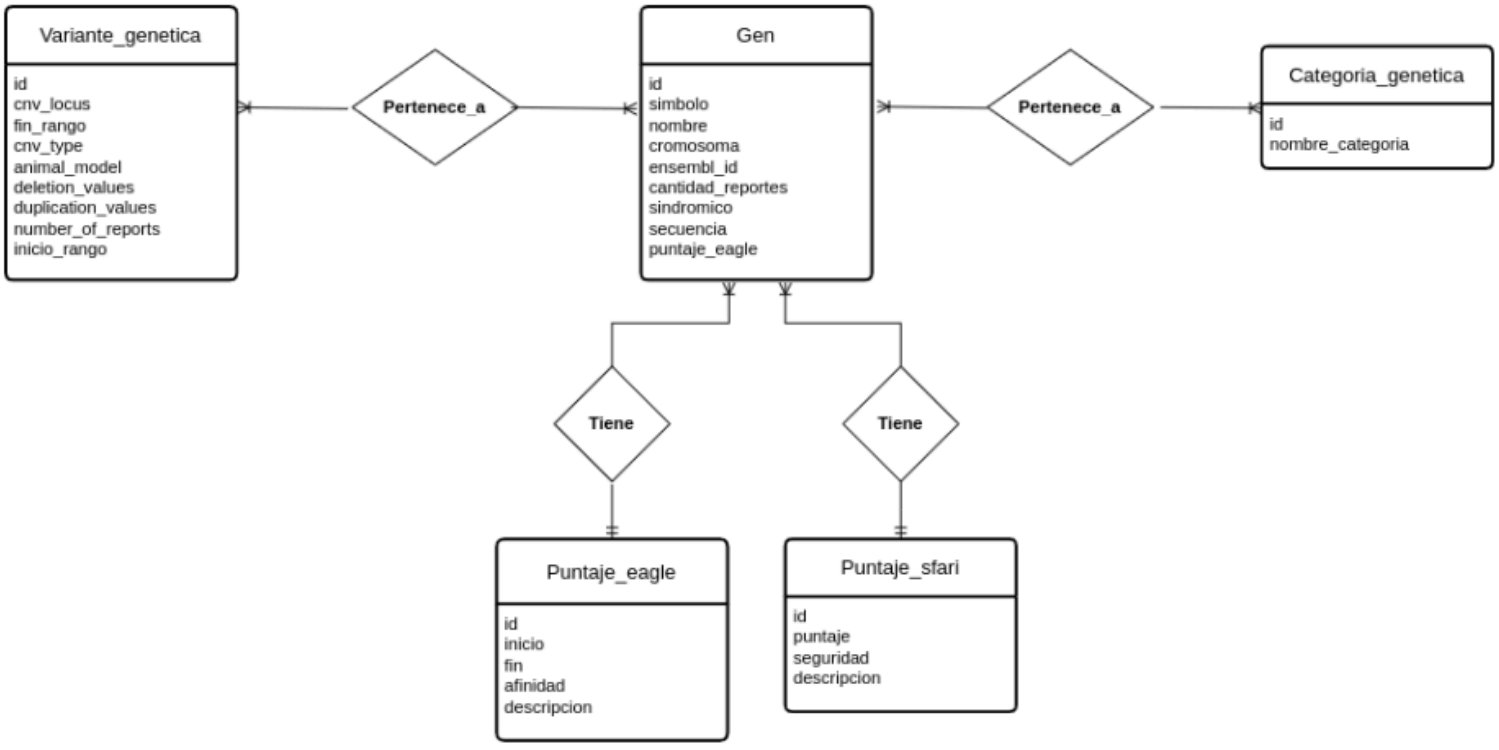
La base de datos ASDgene está diseñada para el análisis de información genética asociada a trastornos del espectro autista (TEA). Su estructura permite almacenar y categorizar datos detallados sobre genes y variantes genéticas que están vinculados específicamente al autismo, lo cual es esencial en investigaciones bioinformáticas orientadas a comprender la genética de este trastorno y a identificar posibles biomarcadores o factores de riesgo.

Los datos principales fueron extraídos de la base de datos humana de SFARI, la cual contiene información de los posibles genes asociados con autismo, cada gen con un puntaje indicando qué tan relacionado está a la patología.

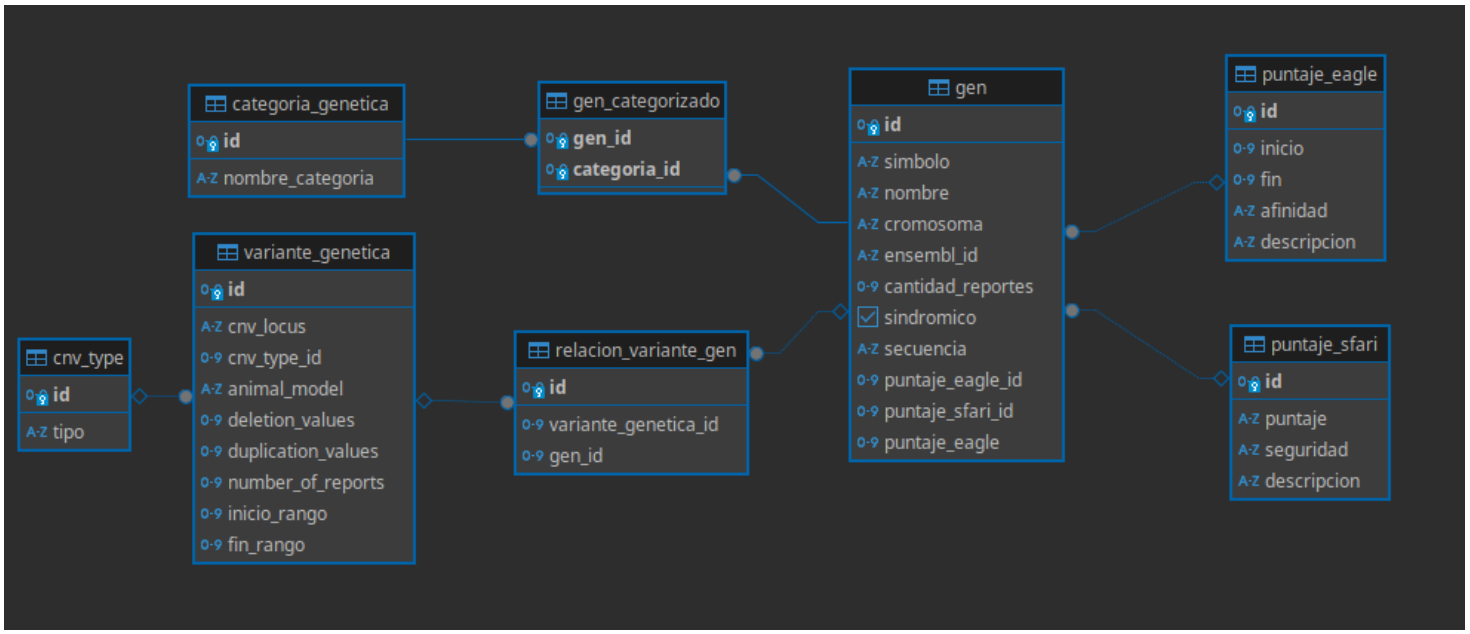
A su vez, incorporé la base de datos CNV, también de SFARI, la cual contiene las variantes genéticas asociadas a la variación en el número de copias (CNV). Las regiones CNV son segmentos de ADN generalmente mayores a 1000pb. Son submicro deleciones y duplicaciones que se cree están muy relacionadas con la patogenicidad de varias enfermedades humanas incluyendo desórdenes psiquiátricos como TEA.

Finalmente, para poder complementar todos estos datos, extraje del NCBI las secuencias de los genes asociados con autismo utilizando la librería Bio.Entrez de python.

Tras haber identificado las entidades y las relaciones que las vinculan, el DER resultante fue:



Posteriormente realicé el DDL, el cual puede verse a continuación:



Con su correspondiente diccionario de datos:

genes.gen			
atributo	**tipo**	**tamaño**	**descripción**
id	serial4	incremento automático de un entero de cuatro bytes	identificador único de cada gen
simbolo	text	serie de caracteres de longitud variable	cada gen tiene un símbolo asociado, suele parecerse a una abreviación del nombre
nombre	text	serie de caracteres de longitud variable	nombre del gen
cromosoma	varchar	serie de caracteres de longitud variable	cromosoma en el que se encuentra el gen
ensembl_id	varchar	serie de caracteres de longitud variable	es un id que identifica genes según el formato: ENS(species)(object type)(identifier)
cantidad_reportes	int4	entero de 4 bytes	cantidad de reportes asociados a ASD del gen
sindromico	bool	booleano lógico (true/false)	indica si el gen es o no sindrómico
secuencia	text	serie de caracteres de longitud variable	secuencia del gen obtenida del NCBI
puntaje_eagle_id	int4	entero de 4 bytes	FK a genes.puntaje_eagle

puntaje_sfari_id	int4	entero de 4 bytes	FK a genes.puntaje_safari
puntaje_eagle	float4	flotante de precisión simple (4 bytes)	puntaje eagle (números positivos) que implican afinidad del gen a ASD

genes.puntaje_eagle			
atributo	**tipo**	**tamaño**	**descripción**
id	serial4	incremento automático de un entero de cuatro bytes	identificador único de puntaje eagle
inicio	float4	flotante de precisión simple (4 bytes)	indica en qué valor de puntaje inicia la clasificación
fin	float4	flotante de precisión simple (4 bytes)	indica en qué valor de puntaje finaliza la clasificación
afinidad	text	serie de caracteres de longitud variable	indica la afinidad que le corresponde al rango de valores del registro
descripcion	text	serie de caracteres de longitud variable	descripción correspondiente al significado de la afinidad

genes.puntaje_sfari			
atributo	**tipo**	**tamaño**	**descripción**
id	serial4	incremento automático de un entero de cuatro bytes	identificador único de puntaje SFARI
puntaje	varchar	serie de caracteres de longitud variable	puntaje determinado por SFARI el cual puede ser 1,2,3 o S
seguridad	text	serie de caracteres de longitud variable	nivel de relación entre el gen y el autismo
descripcion	text	serie de caracteres de longitud variable	significado de la relación anterior

genes.gen_categorizado			
atributo	**tipo**	**tamaño**	**descripción**
gen_id	int4	entero de 4 bytes	id del gen
categoria_id	int4	entero de 4 bytes	id de la categoría a la que corresponde el gen

genes.categoria_genetica			
atributo	**tipo**	**tamaño**	**descripción**
id	serial4	incremento automático de un entero de cuatro bytes	identificador único de la categoría genética
nombre_categoria	text	serie de caracteres de longitud variable	nombre de la categoría

genes.relacion_variante_gen			
atributo	**tipo**	**tamaño**	**descripción**
id	serial4	incremento automático de un entero de cuatro bytes	identificador único
variante_genetica_id	int4	entero de 4 bytes	id de la variante genética
gen_id	int4	entero de 4 bytes	id del gen

genes.variante_genetica			
atributo	**tipo**	**tamaño**	**descripción**
id	serial4	incremento automático de un entero de cuatro bytes	identificador único de la variante genética
cvn_locus	varchar	serie de caracteres de longitud variable	locus donde se ubica la variación en el número de copias (CNV)
cnv_type_id	int4	entero de 4 bytes	FK a genes.cnv_type
animal_model	varchar	serie de caracteres de longitud variable	modelo animal de la variación (si es null es porque se trata de humanos y no un modelo)
deletion_values	int4	entero de 4 bytes	casos de deleciones
duplication_values	int4	entero de 4 bytes	casos de duplicaciones
number_of_reports	int4	entero de 4 bytes	número total de reportes
inicio_rango	int8	entero de 8 bytes	posición de inicio del rango de variación
fin_rango	int8	entero de 8 bytes	posición final del rango de variación

genes.cnv_type			
atributo	**tipo**	**tamaño**	**descripción**
id	serial4	incremento automático de un entero de cuatro bytes	identificador único del tipo de variación
tipo	varchar	serie de caracteres de longitud variable	tipo de variación (deleción, duplicación o ambas)

Cree la base de datos con el motor PostgreSQL y utilizando DBeaver.

```
-- Creación del esquema "genes"
CREATE SCHEMA genes;
-- Creación de la tabla para almacenar los puntajes sfari con seguridad y descripción
CREATE TABLE genes.puntaje_sfari (
   id SERIAL PRIMARY KEY,
   puntaje VARCHAR CHECK (puntaje IN ('1', '2', '3', 'S')),
   seguridad TEXT,
   descripcion TEXT
);
-- Creación de la tabla para almacenar los puntajes eagle con afinidad y descripción
CREATE TABLE genes.puntaje_eagle (
   id SERIAL PRIMARY KEY,
   inicio FLOAT4,
   fin FLOAT4,
   afinidad TEXT,
   descripcion TEXT
);
-- Creación de la tabla de genes con nuevas columnas extendidas en el esquema "genes"
CREATE TABLE genes.gen (
   id SERIAL PRIMARY KEY,
   simbolo TEXT UNIQUE NOT NULL,
   nombre TEXT UNIQUE NOT NULL,
   cromosoma VARCHAR NOT NULL,
   ensembl_id VARCHAR UNIQUE,
   cantidad_reportes INTEGER,
   sindromico BOOLEAN,
   secuencia TEXT,
   puntaje_eagle_id INTEGER REFERENCES genes.puntaje_eagle(id),
   puntaje_sfari_id INTEGER REFERENCES genes.puntaje_sfari(id),
   puntaje_eagle FLOAT4
);
-- Creación de la tabla de categorías genéticas en el esquema "genes"
CREATE TABLE genes.categoria_genetica (
   id SERIAL PRIMARY KEY,
   nombre_categoria TEXT UNIQUE
);
-- Relación muchos a muchos entre genes y categorías en el esquema "genes"
CREATE TABLE genes.gen_categorizado (
   gen_id INTEGER REFERENCES genes.gen(id),
```

```
    categoria_id INTEGER REFERENCES genes.categoria_genetica(id),
    PRIMARY KEY (gen_id, categoria_id)
);
-- Normalización de los tipos de cnv (delections and duplications)
CREATE TABLE genes.cnv_type (
 id SERIAL PRIMARY KEY,
 tipo VARCHAR
);
-- Creación de la tabla para almacenar la información de variantes genéticas desde
genomes.sfari.org en el esquema "genes"
CREATE TABLE genes.variante_genetica (
    id SERIAL PRIMARY KEY,
 cnv_locus VARCHAR,
 cnv_type_id INT REFERENCES genes.cnv_type(id),
 animal_model VARCHAR,
 deletion_values INT,
 duplication_values INT,
 number_of_reports INT,
 inicio_rango BIGINT,
 fin_rango BIGINT
);
CREATE TABLE genes.relacion_variante_gen (
 id SERIAL PRIMARY KEY,
 variante_genetica_id INT REFERENCES genes.variante_genetica(id),
 gen_id INT REFERENCES genes.gen(id)
);
```

Y cargué los datos de cada tabla. Aquellas tablas que tenían pocos datos como por ejemplo la tabla genes.puntaje_sfari que tiene 4 registros, las cargué manualmente. Por otro lado para aquellas tablas que tenían muchos registros (por ejemplo genes.gen que tiene 1203) automaticé la creación de las sentencias INSERT por medio de un script de python.

```
# Import necessary libraries
import csv
from Bio import Entrez
from Bio.Entrez import HTTPError

def generate_insert_statements(csv_file_path, output_txt_file):
    with open(csv_file_path, newline='') as csvfile:
        reader = csv.DictReader(csvfile)
        with open(output_txt_file, 'a') as output_file:
            for row in reader:
                Entrez.email = "anasolm26@gmail.com"
                Entrez.max_tries = 10  # Retries up to 10 times
                Entrez.sleep_between_tries = 5  # Waits for 5 seconds between retries
                try:
                    handle = Entrez.esearch(db="gene", term=row['ensembl-id'])
```

```
                    record = Entrez.read(handle)
                    if record['IdList']:
                        gene_id = record['IdList'][0]
                        handle = Entrez.efetch(db="gene", id=gene_id, retmode="xml")
                        gene_data = Entrez.read(handle)
                        if gene_data:
                            gene_info = gene_data[0]
                            try:
                                        sequence_handle = Entrez.efetch(db="nucleotide",
id=gene_id, rettype="fasta", retmode="text")
                                           sequence = sequence_handle.read().split("\n",
1)[1].replace("\n", "")
                            except HTTPError as e:
                                sequence = 'NULL'
                        else:
                            chromosome = row['chromosome']
                            start_position = 'NULL'
                            end_position = 'NULL'
                            sequence = 'NULL'
                    else:
                        chromosome = row['chromosome']
                        start_position = 'NULL'
                        end_position = 'NULL'
                        sequence = 'NULL'
                except (HTTPError, RuntimeError) as e:
                    sequence = 'NULL'
                chromosome = row['chromosome']

                # Insert gene information
                try:
                    if float(row['eagle']) <= 6:
                        id_eagle=1
                        eagle=row['eagle']
                    elif float(row['eagle']) <= 11:
                        id_eagle=2
                        eagle=row['eagle']
                    elif float(row['eagle']) > 11:
                        id_eagle=3
                        eagle=row['eagle']
                except:
                    id_eagle='NULL'
                    eagle='NULL'
```

```
                insert_statement_gene = f"INSERT INTO genes.gen (simbolo, nombre,
cromosoma, ensembl_id, cantidad_reportes, sindromico, secuencia, puntaje_sfari_id,
puntaje_eagle_id, puntaje_eagle) VALUES ('{row['gene-symbol']}',
'{row['gene-name']}', '{chromosome}', '{row['ensembl-id']}',
{row['number-of-reports']}, {bool(row['syndromic'])}, '{sequence}', (SELECT id FROM
genes.puntaje_sfari WHERE puntaje = '{row['gene-score']}'), {id_eagle}, {eagle});\n"
                output_file.write(insert_statement_gene)
                print(f"soy el gen {row['gene-symbol']}")

                # Insert genetic category information
                categories = row['genetic-category'].split(', ')
                for category in categories:
                                insert_statement_gen_categorizado = f"INSERT INTO
genes.gen_categorizado (gen_id, categoria_id) VALUES ((SELECT id FROM genes.gen WHERE
simbolo = '{row['gene-symbol']}'), (SELECT id FROM genes.categoria_genetica WHERE
nombre_categoria = '{category}'));\n"
                output_file.write(insert_statement_gen_categorizado)

# Usage
csv_file_path =
"/home/ana/Downloads/SFARI-Gene_genes_10-09-2024release_11-07-2024export.csv"
output_txt_file = "insert_statements_gen.txt"
generate_insert_statements(csv_file_path, output_txt_file)
```

A fin de probar la base de datos creada realicé las siguientes consultas complejas:

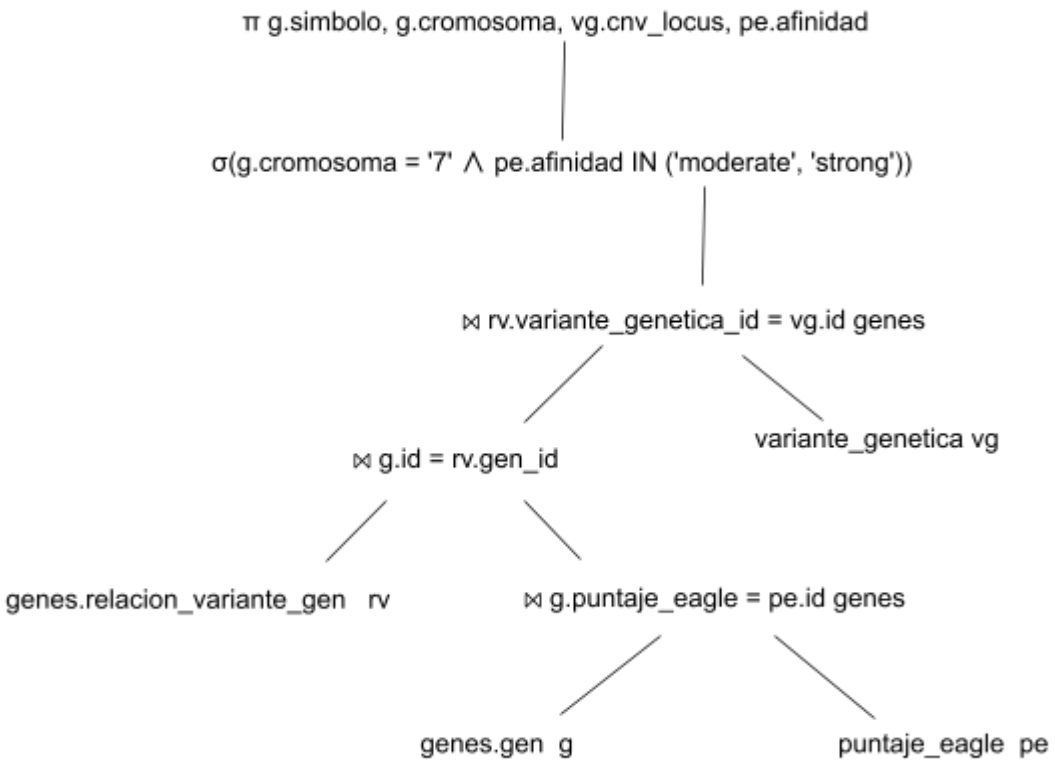
```
-- Consulta para listar genes con sus puntajes en Eagle y SFARI, ordenados por afinidad
SELECT
   g.simbolo,
   g.cromosoma,
   pe.afinidad,
   ps.puntaje AS puntaje_sfari
FROM
   genes.gen g
INNER JOIN
   genes.puntaje_eagle pe ON g.puntaje_eagle = pe.id
INNER JOIN
   genes.puntaje_sfari ps ON g.puntaje_sfari_id = ps.id
ORDER BY
   pe.afinidad DESC;
```

```sql
-- Consulta para para encontrar genes que tengan alguna relación con una variante genética
que tenga más de 5 reportes
SELECT
   g.simbolo,
   g.cromosoma,
   g.cantidad_reportes
FROM
   genes.gen g
WHERE
   EXISTS (
       SELECT 1
       FROM
           genes.relacion_variante_gen rv
       JOIN
           genes.variante_genetica vg ON rv.variante_genetica_id = vg.id
       WHERE
           rv.gen_id = g.id AND vg.number_of_reports > 5
   );



-- Consulta para obtener los genes que pertenecen a la categoría Rare Single Gene Mutation
SELECT
   g.simbolo,
   g.cromosoma,
   cg.nombre_categoria
FROM
   genes.gen g
JOIN
   genes.gen_categorizado gc ON g.id = gc.gen_id
JOIN
   genes.categoria_genetica cg ON gc.categoria_id = cg.id
WHERE
   cg.nombre_categoria IN ('Rare Single Gene Mutation');
```

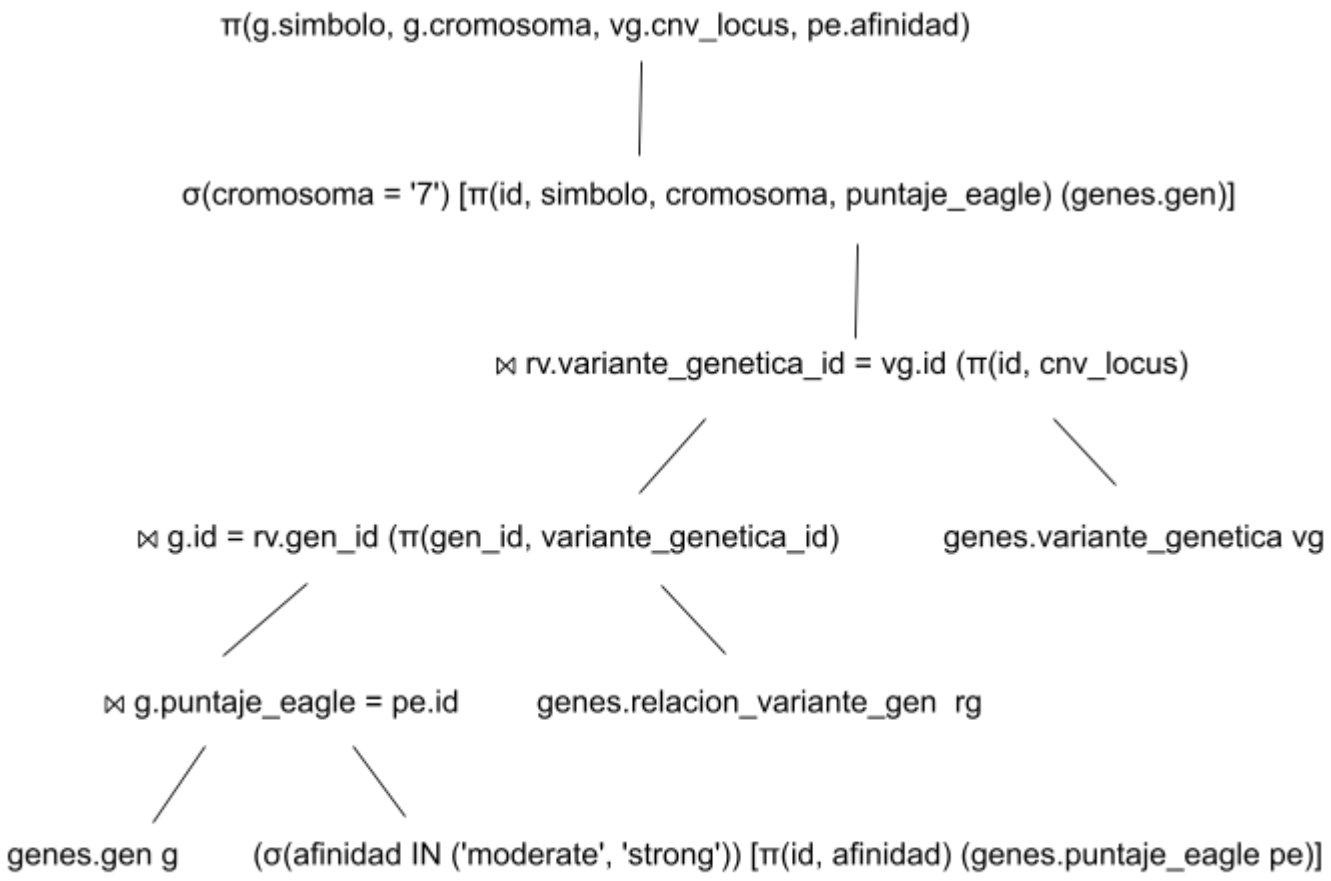
Luego cree una consulta que utilice tres tablas, contenga una condición de igualdad y una condición de rango (>, >=, <, <=, between).

```sql
SELECT
   g.simbolo,
   g.cromosoma,
   vg.cnv_locus,
   pe.afinidad
FROM
   genes.gen g
JOIN
   genes.puntaje_eagle pe ON g.puntaje_eagle = pe.id
JOIN
   genes.relacion_variante_gen rv ON g.id = rv.gen_id
JOIN
   genes.variante_genetica vg ON rv.variante_genetica_id = vg.id
WHERE
   g.cromosoma = '1'            -- Condición de igualdad
   AND pe.afinidad IN ('moderate', 'strong');  -- Condición de afinidad categórica
```

El árbol de álgebra relacional de la consulta anterior es el siguiente:



Haber optimizado heurísticamente el árbol de ejecución anterior, resultó en el siguiente árbol:



En base al árbol de ejecución optimizado, escribí la consulta correspondiente:

```sql
SELECT
   g.simbolo,
   g.cromosoma,
   vg.cnv_locus,
   pe.afinidad
FROM
   (SELECT id, simbolo, cromosoma, puntaje_eagle FROM genes.gen WHERE cromosoma = '1') g
JOIN
    (SELECT id, afinidad FROM genes.puntaje_eagle WHERE afinidad IN ('moderate', 'strong'))
pe ON g.puntaje_eagle = pe.id
JOIN
     (SELECT gen_id, variante_genetica_id FROM genes.relacion_variante_gen) rv ON g.id =
rv.gen_id
JOIN
     (SELECT id, cnv_locus FROM genes.variante_genetica) vg ON rv.variante_genetica_id =
vg.id;
```

Por último, desarrollé una interfaz web utilizando la base de datos creada, Python, Flask, JavaScript, CSS y HTML.

La misma consta de dos pestañas: Main y Genomic

En la pestaña Main se observan datos estadísticos graficados tanto en forma de barra como de dona que representan la cantidad de genes de acuerdo a alguna clasificación del menú lateral izquierdo.

Al acercarse el cursor sobre una barra o segmento de la dona, se aclara lo que está graficado.

Por otro lado, en la pestaña Genomic, se puede elegir un gen del menú lateral izquierdo y se visualizará su secuencia, el %A, %T, %C, %G, %CG y una representación de 8-mers correspondiente a la secuencia. Toda esta información puede ser descargada en PDF desde el botón en la parte superior de la lista de genes.