UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Tesis de Máster

Máster de Posgrado Oficial en Ingeniería del Software, Métodos Formales y Sistemas de Información

Departamento de Sistemas Informáticos y Computación

Diseño e Implementación de un Sistema de Información Genómico para el Diagnóstico de la Catarata Congénita utilizando la Metodología SILE

Septiembre de 2017, Valencia

Manuel Navarrete Hidalgo
manahi@inf.upv.es

Director: Dr. *Oscar Pastor López*
opastor@pros.upv.es

Director experimental: *José Fabián Reyes Román*
jreyes@pros.upv.es

RESUMEN

La biomedicina y el diagnóstico preventivo de enfermedades abren una serie de líneas de investigación tan diversas como soluciones propuestas. Sin embargo, la información que el ser humano contiene dentro del genoma representa un gran desafío relacionado con el procesamiento y gestión de su información biológica, cuyo éxito dependerá directamente de las estructuras que se generen mediante la aplicación de técnicas de modelado conceptual. En este sentido, se desarrolla un prototipo de "Extracción-Transformación-Carga" capaz de obtener información biológica desde múltiples repositorios científicos que no tienen interacción directa. Para ello se parte de un Modelo Conceptual del Genoma Humano (MCGH) propuesto para la generación de Sistemas de Información Genómicos (GeIS), los cuales permiten gestionar de manera eficiente todo el conocimiento existente en el genoma a fin de potenciar la Medicina de Precisión. En este trabajo se define un GeIS para el diagnóstico preventivo de las cataratas congénitas, cuyo padecimiento no está ligado a la edad ni al estilo de vida, sino con el componente genético de cada persona. De esta manera, será posible su diagnóstico anticipado y posible tratamiento.

Palabras clave: Cataratas, diagnóstico preventivo, modelo conceptual, sistema de información, ETL, GeIS.

ABSTRACT

Biomedicine and the preventive diagnosis of diseases open a series of lines of research as diverse as proposed solutions. However, the information that the humans contain within the genome represents a great challenge related to the processing and management of their biological information, whose success will depend directly on the structures that will be generated through the application of conceptual modeling techniques. In this context, we developed a prototype of "Extraction-Transformation-Load", where biological information can be obtained from multiple scientific repositories that don't have direct interaction. For that reason, we use a Human Genome Conceptual Model (HGCM) proposed for the generation of Genomic Information Systems (GeIS), which allow an efficient management of all the existing knowledge in the genome in order to enhance Precision Medicine. This work defines a GeIS for the preventive diagnosis of congenital cataracts, whose condition is not related to age and lifestyle, but to the genetic component of each person. In this way, it will be possible an early diagnosis and treatment.

Key words: Cataracts, preventive diagnosis, conceptual model, information system, ETL, GeIS.

AGRADECIMIENTOS

A mis padres, por permitirme una educación que no pudieron disfrutar, pero que con gran esfuerzo y sacrifico sacaron adelante una gran familia, la primera con estudios universitarios. Sin olvidar a mi hermano Jaime, cuya muestra de responsabilidad fue de gran motivación.

A mi novia Carolina, por su apoyo y paciencia, amor y calor, que sin todo ello no hubiera sido posible acabar este trabajo e ilusionarme con lo que nos deparará el futuro.

A mis amigos Fernando Cervera y Rubén Casatejada, por su visión, tiempo y dedicación. Esta aventura en la que la informática se mezcla con la biología no hubiera sido posible sin vuestra ayuda.

A José F. Reyes Román, por su incondicional apoyo y dedicación, el cual me permitió desarrollar un trabajo de calidad mediante un seguimiento continuo y comunicación directa.

A Oscar Pastor López, por su infinita paciencia y por permitirme trabajar en su grupo de investigación (PROS), donde fui acogido con los brazos abiertos desde el primer día.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABLAS

CAPÍTULO 1: INTRODUCCIÓN

1.1. MOTIVACIÓN

La bioinformática orientada a las ciencias de la salud tiene como objetivo fomentar el desarrollo de nuevas tecnologías computacionales, así como *Sistemas de Información* (SI) de utilidad en la investigación biomédica. Esto es debido a la gran cantidad de datos existente en torno al dominio genómico, cuyo estudio posibilita el desarrollo de una nueva práctica basada en el análisis de su información, la cual debe ser correctamente *estructurada*, *catalogada* y *almacenada* con el objetivo de facilitar su acceso, entendimiento y explotación. Esto es de vital importancia, porque el "*genoma humano*" es el producto de una evolución compleja que ha generado una estructura y modos de funcionamiento difíciles de comprender y, por ende, de modelar.

Uno de los desafíos más importantes y presentes en el dominio de la bioinformática, es la unificación de la información bajo un esquema global que evite situaciones de confusión e inconsistencia de datos y conocimiento, problema que se da actualmente por la cantidad de repositorios biológicos existentes y el caos estructural que albergan. Todo ello se produce porque en sus orígenes no existían técnicas de modelado conceptual que facilitaran su escalabilidad ni unificaran el conocimiento relacionado con el genoma humano, como sí pretende el *Modelo Conceptual del Genoma Humano* (MCGH) propuesto por el Centro PROS, cuya representación conceptual del genoma abre una vía de estandarización en cuanto al acceso, consulta, explotación y generación de herramientas de índole preventivo para enfermedades de origen genético.

En este sentido, se abren vías de investigación que confluyen hacía la *medicina personalizada* (1), (2), que desde un punto de vista fármaco-genómico permitirían la futura existencia de terapias génicas como solución certera y definitiva en enfermedades con tratamientos muy costosos y cuya eficacia depende actualmente del individuo que la padece. Por otro lado, existen implicaciones filosóficas, sociales y éticas estrechamente relacionadas en este ámbito, ya que la medicina personalizada y el estudio de predisposición genética podría provocar discriminación laboral, médica y/o jurídica, lo que hace patente los importantes

aspectos pendientes de resolver, como la privacidad y confidencialidad genética, así como la garantía y relevación de información, entre otros (3). Aunque existen normativas al respecto, países como Canadá, Estados Unidos, Rusia y Japón han optado por no prohibir específicamente el acceso a los datos genéticos (4).

Finalmente, destacar la constante especialización de ingenieros informáticos en materia de biología y salud, así como la adquisición de conocimientos relacionados con los sistemas de información por parte de biólogos, cambios a nivel profesional que han dado lugar a la aparición de estudios universitarios y de postgrado relacionados con esta temática.

1.2. PLANTEAMIENTO DEL PROBLEMA

Desde el inicio de la secuenciación masiva, uno de los retos de la biología ha sido entender las bases moleculares de los organismos. Sin embargo, la gran cantidad de datos recopilados han generado un problema para la gestión de esa información. A pesar de que se han desarrollado multitud de herramientas computacionales para la evaluación de esa información (5), hay una gran problemática asociada al acceso y consulta de datos genómicos almacenados en los repositorios científicos, ya que los ofrecen de forma dispersa, heterogénea y redundante. Para avanzar en el conocimiento y su aplicación, la mayoría de autores apuntan a la necesidad de una mejora en la capacidad de análisis y gestión de esa información (6).

Por ello, el paso de la extracción de información biológica debería automatizarse mediante el desarrollo de herramientas de software para la ejecución del ETL (extracción, transformación y carga), el cual debería ser capaz de consultar los repositorios de información genómicos y adaptar los datos existentes según los requisitos de un modelo teórico, como por ejemplo el Modelo Conceptual del Genoma Humano (MCGH) presentado en este Trabajo Fin de Máster.

Esas herramientas de software permitirían reducir los tiempos de interacción entre el humano y los repositorios genómicos para obtener información, ya que es en esta etapa donde más recursos se invierten y más errores se comenten. Eso permitiría la aplicación directa de la información genómica en temas de salud pública, como en la detección preventiva de las *cataratas*.

La *catarata* es una enfermedad que afecta a la visión humana y animal, producida principalmente por la pérdida de transparencia del cristalino, componente que realiza la función de lente y se encuentra justo detrás de la pupila (7). Las personas que padecen catarata sufren de: una visión turbia, incapacidad para apreciar el contorno de lo que ven, pérdida de intensidad en los colores e hipersensibilidad al resplandor, además de padecer dolores de cabeza y fatiga visual. Esto es producido porque el cristalino se vuelve opaco por la alta concentración de proteínas en sus células y el desarrollo de cuerpos densos (8).

La aparición de la catarata se asocia principalmente a la edad, aunque también existen motivos nutricionales, genéticos y hereditarios, por infecciones intrauterinas, efectos secundarios producidos por medicamentos, colaterales por radioterapia y exposición a rayos UV, así como por trastornos metabólicos como la galactosemia y la diabetes. Las cataratas son la causa de la mitad de la ceguera y ceguera reversible en todo el mundo (9).

Una de las formas de mejorar la detección y brindar mecanismos de prevención y/o curación, además de estudiar su vinculación directa con otras enfermedades, una tarea interesante consistiría en analizar los repositorios genómicos en busca de indicadores para la "*catarata congénita*", cuya aparición no está relacionada con el envejecimiento natural del ser humano sino con el componente *genético* de cada persona.

1.3. OBJETIVOS Y METAS

El estudio genético permite conocer la variación y la diversidad entre los seres vivos, la expresión y la respuesta a diferentes enfermedades mediante el descubrimiento de las alteraciones del metabolismo en cada persona.

Todo ello posibilita y facilita una serie de beneficios:

- *Un pronóstico rápido y preciso.*
- *Una concepción real y acotada de la gravedad y posibles complicaciones.*
- *Conocer las posibilidades de mejora del paciente.*
- *Un manejo, seguimiento y tratamiento personalizado (Medicina preventiva) que sea capaz de estimar la edad de aparición del posible trastorno/afección, así como el grado de expresividad.*

En el presente trabajo se estudiará y analizará el origen, desarrollo y predisposición genética a la hora de padecer *catarata*, centrándose en aquella con un fuerte componente genético-hereditario. Para ello, se empleará la metodología SILE (10), desarrollada en el Centro de Investigación PROS de la Universidad Politécnica de Valencia (UPV). Esta metodología está compuesta por cuatro etapas cuya función y orden es la siguiente:

- **S**: Search: Búsqueda de genes asociados de manera directa o indirecta al padecimiento de la enfermedad de estudio.
- **I**: Identification: Identificación de las variaciones genéticas asociadas mediante el apoyo de personal científico-médico para su correcta identificación y validación.
- **L**: Load: Carga selectiva en la base de datos para su futuro acceso y consulta.
- **E**: Exploitation: Explotación de la información para la obtención de resultados y conclusiones.

De acuerdo con lo anterior, los objetivos específicos que justifican el presente Trabajo Final de Máster son:

- Estudiar el Modelo Conceptual del Genoma Humano (MCGH) empleado para el desarrollo de esta tesis, desde las principales aproximaciones hasta su actual versión (v2).
- Emplear la metodología SILE para la correcta y ordenada búsqueda, identificación, carga y explotación de la información genómica.
- Desarrollar una herramienta que automatice la obtención de los datos provenientes de los repositorios de información biológica más destacables para su posterior carga en la Base de Datos del Genoma Humano (HGDB), la cual es una proyección del MCGH.
- Estudiar y emplear el framework genómico "*VarSearch"* para la explotación y obtención de resultados, demostrando así la validez de la solución propuesta. Para ello, se contrastará la información cargada en la HGDB con las variaciones del paciente, ofrecida en ficheros tipo VCF (*Variant Call Format)* (11).

1.4. ESTRUCTURA DE LA TESINA

Para la obtención de los objetivos anteriormente definidos esta tesis se estructura de la siguiente forma:

En el capítulo 2, se presenta el estado del arte, donde se lleva a cabo una investigación documental de la modelización del genoma humano, desde las primeras aproximaciones hasta el modelo conceptual empleado en este trabajo, detallando su disposición, vistas y conexión entre todas las entidades que dan cabida a la estructura del genoma. Se describe también la forma en la que las variaciones de la secuencia de genes se representan y almacenan digitalmente mediante archivos VCF para su procesamiento y análisis. Este capítulo cierra con el listado, descripción y acceso de las principales fuentes de datos biológicos existentes que incluyen información de secuencia de genomas, nucleótidos, proteínas, estructuras de proteínas, de enfermedades genéticas humanas y bibliográficas, entre otras.

En el capítulo 3, se describen algunos elementos relevantes asociados con la biología molecular. También se detallan los conceptos más importantes del genoma humano, como el ADN, genes, proteínas, rutas metabólicas, variaciones genéticas, genotipo y fenotipo, así como su funcionamiento, comportamiento y evolución.

En el capítulo 4, se desarrolla la metodología SILE aplicada al estudio de la *catarata congénita*. Para ello, se localizan y describen todos los genes asociados con las cataratas congénitas. Posteriormente, se identifican las variaciones existentes en cada gen y se lleva a cabo la carga y la explotación. En el proceso de carga se implementa un software (*prototipo*) cuya misión consistirá en extraer de un conjunto de repositorios científicos toda la información subyacente dada una variación concreta, y alimentar la base de datos del genoma humano (HGDB).

En el capítulo 5, se reflexiona sobre el trabajo realizado, el alcance del software desarrollado, su utilidad, las ventajas asociadas y los objetivos llevados a cabo, así como las posibles mejoras, adaptaciones y futuras líneas de investigación asociadas.

CAPÍTULO 2: ESTADO DEL ARTE

La bioinformática nace de la interacción de la biología molecular y las ciencias de la computación, siendo su objetivo dar cabida al tratamiento de datos biológicos. Estos datos se caracterizan por su gran tamaño y continuo crecimiento, motivo por el que surge la necesidad de desarrollar herramientas que permitan gestionar la información de manera ágil y eficaz, además de nuevos algoritmos y soluciones estadísticas orientadas al análisis de la secuencia de ADN y sus variaciones.

En este sentido, existen diversos estándares y formatos para la representación de secuencias de nucleótidos y proteínas, programas de comparación de variaciones y frameworks de exploración de enfermedades genéticas, así como bases de datos con todas las secuencias a disposición pública para su estudio e investigación, como Ensemble (12), USCS (13), GenBank (14) y DDBJ (15).

Por otro lado, toda esta información biológica debe de estar perfectamente estructurada y acotada en un *Sistema de Información* (*SI*) que permita su tratamiento y explotación de forma eficiente. *El entendimiento del genoma es una tarea compleja y se debe conocer el dominio para generar una definición conceptual* (16) *correcta que aborde todo el conocimiento genómico actual*. Además, este modelo conceptual debe estar sujeto a constantes evoluciones producto de nuevos descubrimientos en el dominio.

En el ámbito de la bioinformática genómica los primeros trabajos de modelado conceptual fueron dados por Paton (17). Sus ensayos estaban apoyados en los trabajos previos de modelado de estructuras de proteínas (18), resultando pionero en el diseño conceptual de la célula eucariota, su organización genómica, el modelado del transcriptoma, proteoma y metaboloma, entre otros; mediante la utilización del lenguaje de modelado unificado -UML-.

Por otro lado, Ram (19), orientando su enfoque de modelado al contexto de las proteínas, asienta que el empleo del modelado conceptual facilita la representación sin pérdida semántica y las operaciones de comparación y búsqueda en estructuras complejas, como en el modelado de la estructura tridimensional de las proteínas, caracterizada por el gran volumen de datos que aguardan.

Con esta base, el grupo de *Sistemas de Información Genómicos del Centro de Investigación PROS*, de la Universidad Politécnica de Valencia, inicia en 2008 una línea de investigación centrada en el modelado del genoma humano para el análisis y estudio desde su expresión más básica, los genes y sus mutaciones dentro de un segmento cromosómico (20). Sin embargo, este modelo no soporta la existencia de procesos como la regulación o codificación de una misma proteína por dos genes diferentes, así como la acción combinada de múltiples genes. Tal y como se detalla en el trabajo de 2011 (21), en este esquema existen tres vistas:

- *Genome View*, donde se modela los genomas humanos.
- *Gene Mutation View*, donde se modela las entidades Gene y Allele, y el conocimiento subyacente de la estructura de los genes y sus variantes.
- *Transcription View*, que modela los procesos de transcripción. Esta versión del esquema conceptual del genoma humano recibe el nombre de modelo "esencial" (22).

A continuación, este esquema es ampliado con una cuarta vista denominada "*Phenotype View*" que da cabida a la representación fenotípica, es decir, la manifestación concreta/visible de un genotipo en un contexto determinado.

Posteriormente, el MCGH migró de un modelado orientado al estudio de los genes a otro centrado en el concepto del cromosoma (21). Este nuevo enfoque recibe el nombre de MCGH versión 2, y sus características y estructura se describen en el siguiente apartado.

2.1. MODELO CONCEPTUAL DEL GENOMA HUMANO (MCGH)

El Modelo Conceptual del Genoma Humano desarrollado (MCGH) versión 2, se compone de cinco vistas independientes (pero relacionadas entre sí). Estas vistas son: estructural, variaciones, fenotípica, bibliográfica y de rutas metabólicas. Su función e información se describe a continuación:

Vista estructural: Detalla la estructura, organización y composición de los cromosomas, sus bandas, elementos regulatorios, elementos transcriptos, los genes implicados, así como la información del ser vivo al que pertenece, entre otros. Esta vista se relaciona con la vista transcripción y la vista variación a través de la clase *Chromosome_Element* y la clase *Variation*, respectivamente (Figura 1).

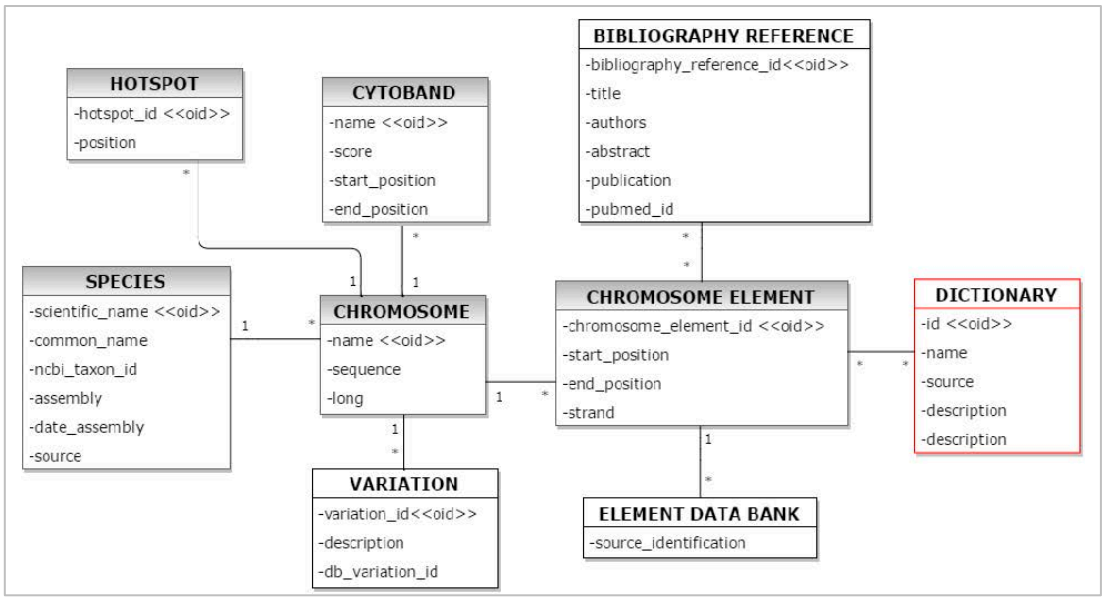

Figura 1. Vista estructural del MCGM. Fuente: (22)

Vista transcripción: Detalla la información de los exones, los elementos transcritos que lo forman, y las proteínas que codifican los diferentes transcritos (Figura 2).

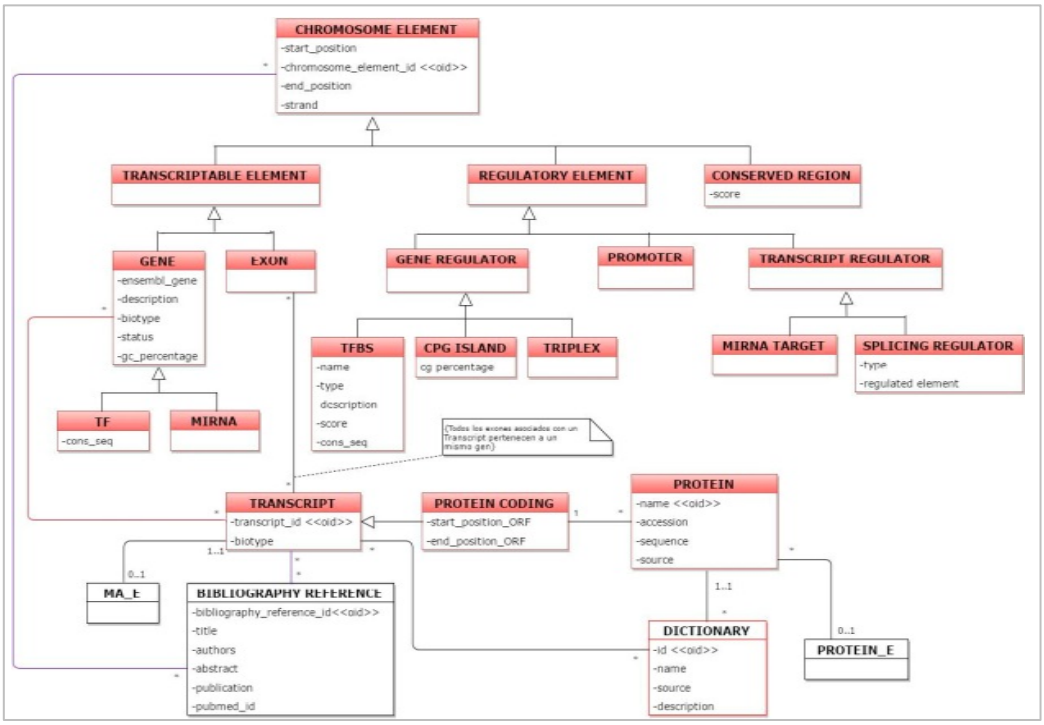

Figura 2. Vista transcripción del MCGM. Fuente: (22)

Vista variaciones: Detalla las variaciones que puede tener una secuencia de ADN, los polimorfismos genéticos SNP (Polimorfismo de Nucleótido Simple, cuya variación sólo afecta a una base) o CNV (Variación en el Número de Copias, cuyo

número de copias es variable), así como las mutaciones genéticas precisas (deleción, inserción o inversión) e imprecisas, entre otros (Figura 3).

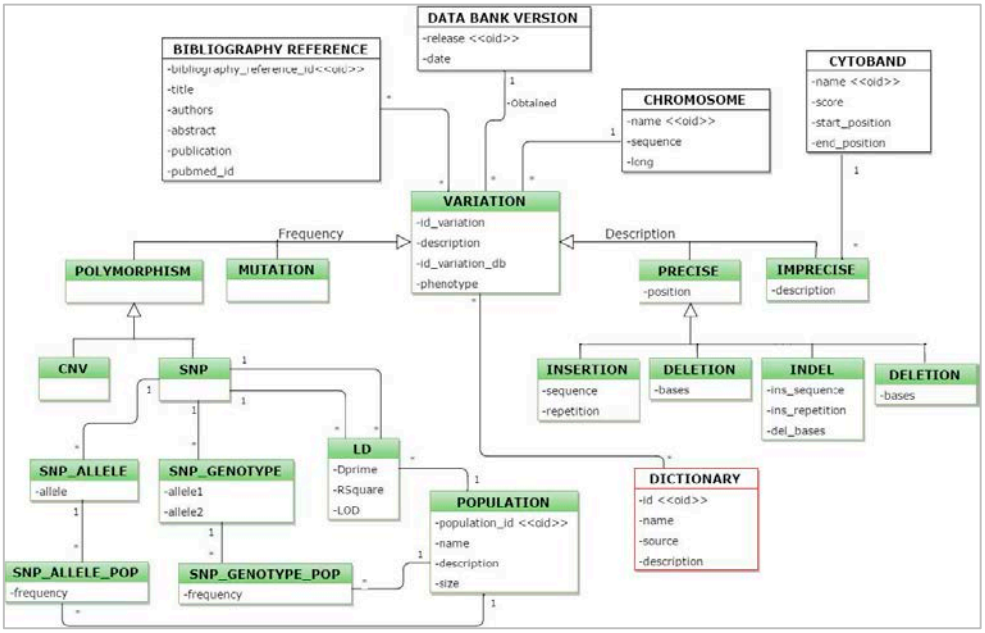

Figura 3. Vista variaciones del MCGM. Fuente: (22)

Vista fenotípica: Muestra la manifestación de un genotipo en un contexto determinado, la relación con las variaciones que lo provocan, y el síndrome asociado (Figura 4).

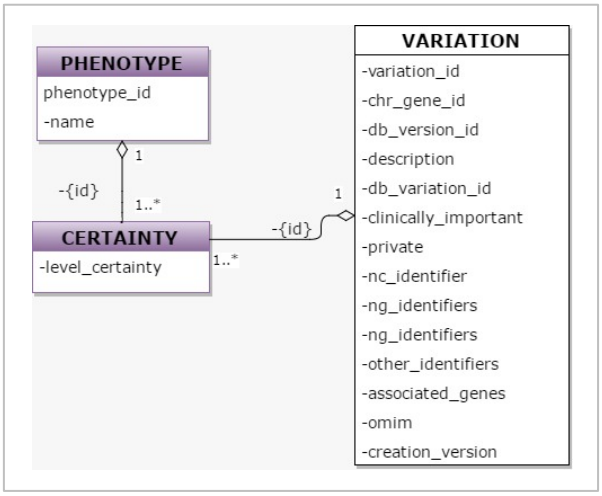

Figura 4. Vista fenotípica del MCGM. Fuente: (22)

Vista bibliográfica: En ella se describe las fuentes de información, publicaciones, autores e investigadores implicados en las investigaciones de las cuales se ha extraído toda la información genética (Figura 5).

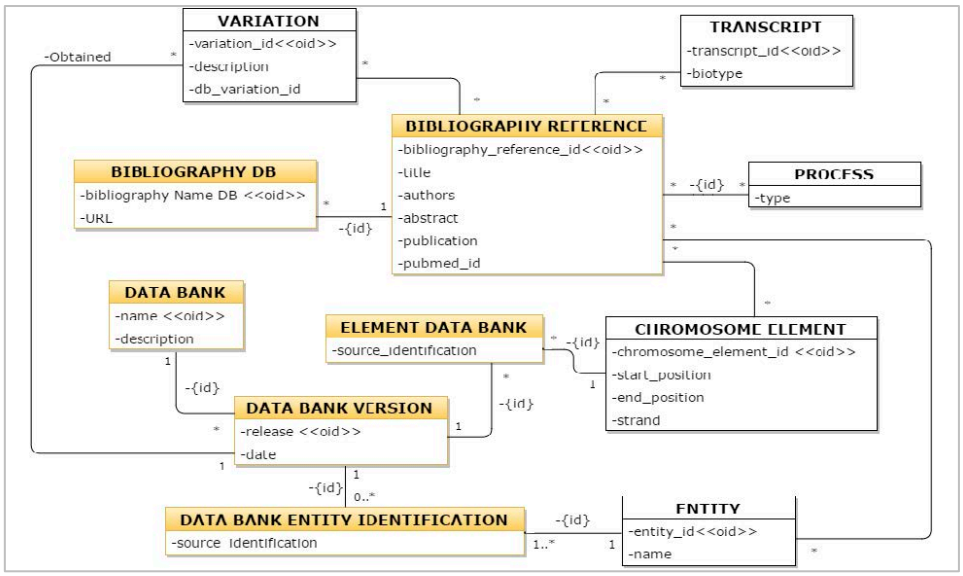

Figura 5. Vista bibliográfica del MCGM. Fuente: (22)

Vista rutas metabólicas: Muestra las relaciones enzimáticas y bioquímicas que suceden en relación con el gen estudiado (Figura 6).

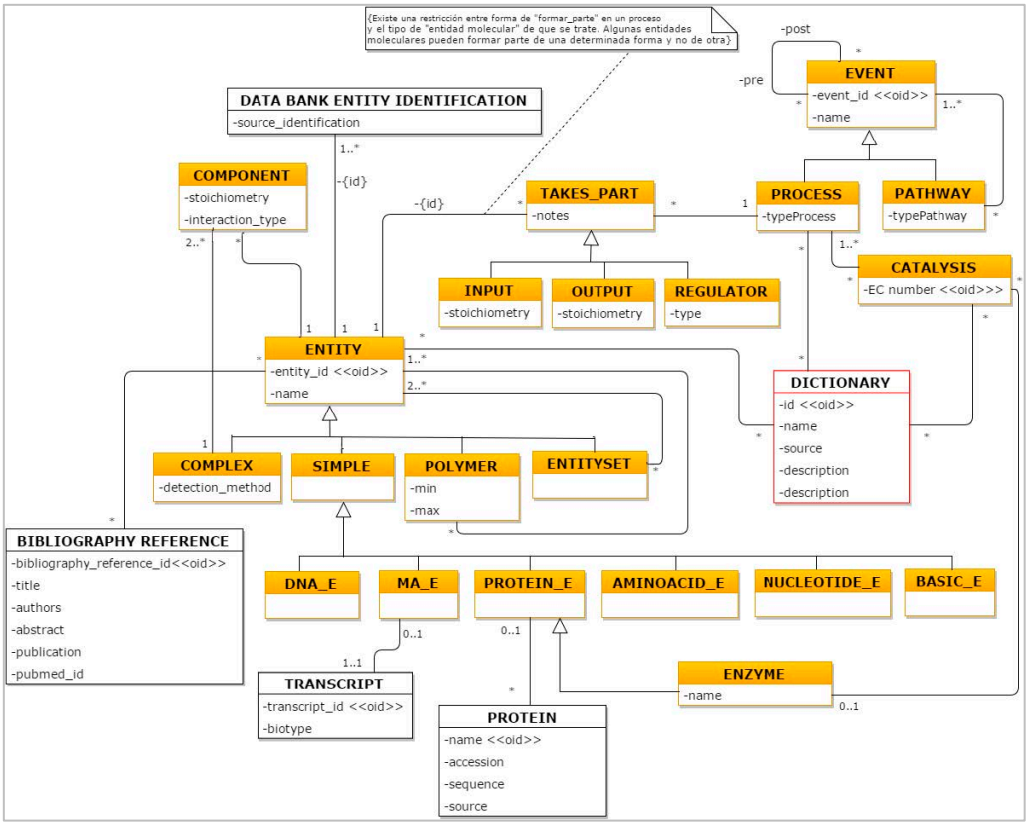

Figura 6. Vista rutas metabólicas del MCGM. Fuente: (22)

2.1.1. ESQUEMA DE BASE DE DATOS DEL MODELO CONCEPTUAL

Una vez definido el MCGH, resulta necesaria la existencia de una entidad que permita el almacenamiento y acceso a la información de forma rápida y estructurada. Estas operaciones se garantizan mediante un esquema de bases de datos relacional que actualmente modela la vista estructural, variaciones, fuentes de datos y bibliografía, y usuarios y validaciones mediante tablas. El detalle de cada una ellas, las claves primarias y ajenas, las restricciones de actualización y eliminación, así como el tipo de datos soportados están debidamente descritos en el documento Resumen: Presentación de la base de datos v3 (Estructura) (23).

Vista estructural: La vista estructural (Figura 7) está formada por las siguientes tablas:

- *Genome*, que indica la versión de ADN de estudio.
- *Chromosome*, que da cabida a los 23 pares de cromosomas que tiene el ser humano.
- *Chr_Elem*, que contiene la información de los elementos de estudio dentro de cromosoma.
- *Gene*, que estructura toda información relacionada con los genes, como su abreviatura única, nombre oficial, descripción, etc.
- *Sequence_Ng*, que representa las secuencias de cada una de los genes.
- *Transcript*, que almacena el transcrito resultante del proceso de transcripción.
- *Exon*, la unidad básica de transcripción.
- *Exon_Transcript*, necesaria para asociar qué exones componen el transcrito asociado a un gen.
- *Protein*, que contiene la información de las proteínas, como su identificador, fuente de datos y secuencia de aminoácidos.

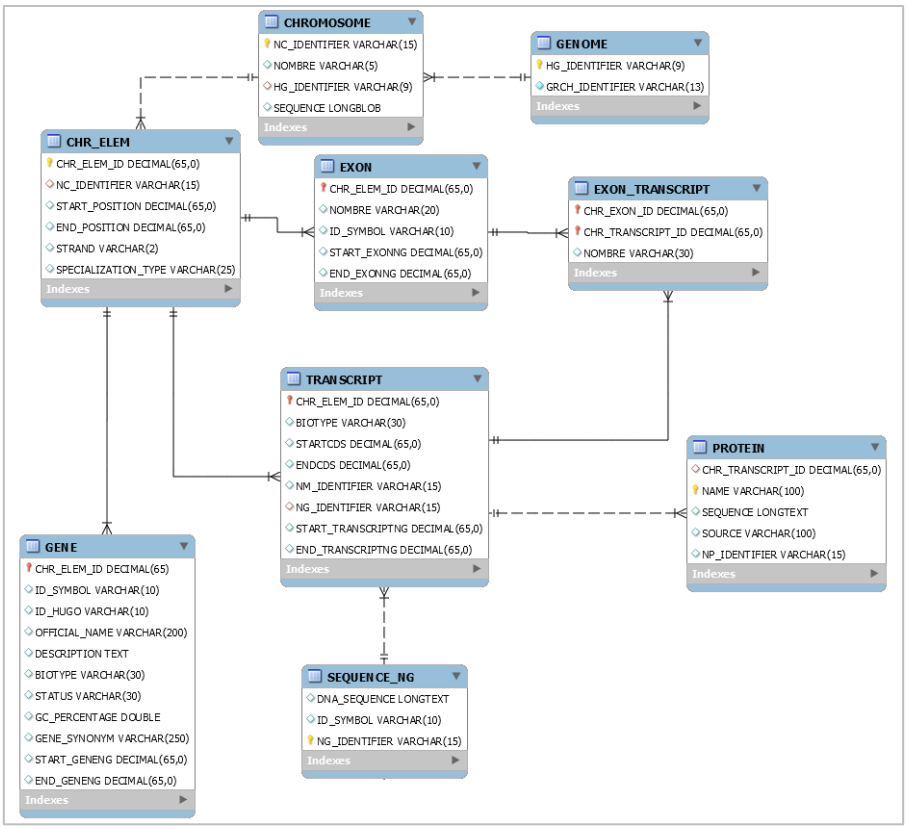

Figura 7. Vista estructural de la DB versión 3. Fuente (23)

Vista variaciones: La vista variaciones (Figura 8) está compuesta por las tablas:

- *Variation*, que da cabida a las diferencias existentes entre los distintos individuos.
- *Precise*, que representa las variaciones encontradas en una posición concreta dentro del cromosoma.
- *Imprecise*, cuyo objetivo es albergar aquellas variaciones cuya posición dentro de la secuencia de ADN es desconocida.
- *Precise_SeqNg*, encargada de asociar la variación y su posición dentro de la secuencia a la de un gen.
- *Phenotype* y *Certainty*, representan los fenotipos asociados a las variaciones y su nivel de certeza, respectivamente.

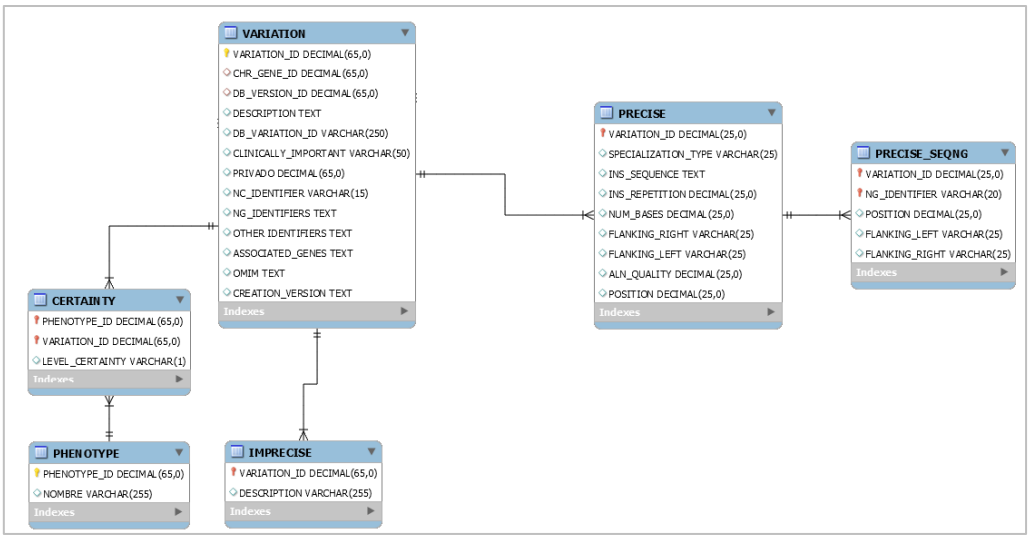

Figura 8. Vista variaciones de la DB versión 3. Fuente: (23)

Vista fuente de datos y bibliografía: La vista fuente de datos (Figura 9) ofrece información de las distintas fuentes de datos empleadas para alimentar el modelo, mientras que la vista bibliográfica informa de los artículos científicos y la fuente de origen. Su estructura de tablas es la siguiente:

- *Databank*, que muestra información básica del origen de la fuente, su nombre, descripción y enlace directo.
- *Databank_version*, que hace referencia a la versión empleada de la base de datos utilizada.
- *Element_databank*, que hace de enlace entre los elementos del cromosoma y la fuente de datos que la documenta.
- *Bib_Ref*, que informa del título, autores, resumen, fecha y publicación de los artículos de investigación.
- *Bibliography_Db,* que muestra de la base de datos donde se alojan los artículos.
- *Ref_Chr_Elem y Reference_Variation*, cuya función es asociar los elementos cromosómicos y las variaciones encontradas con sus respectivas fuentes bibliográficas.

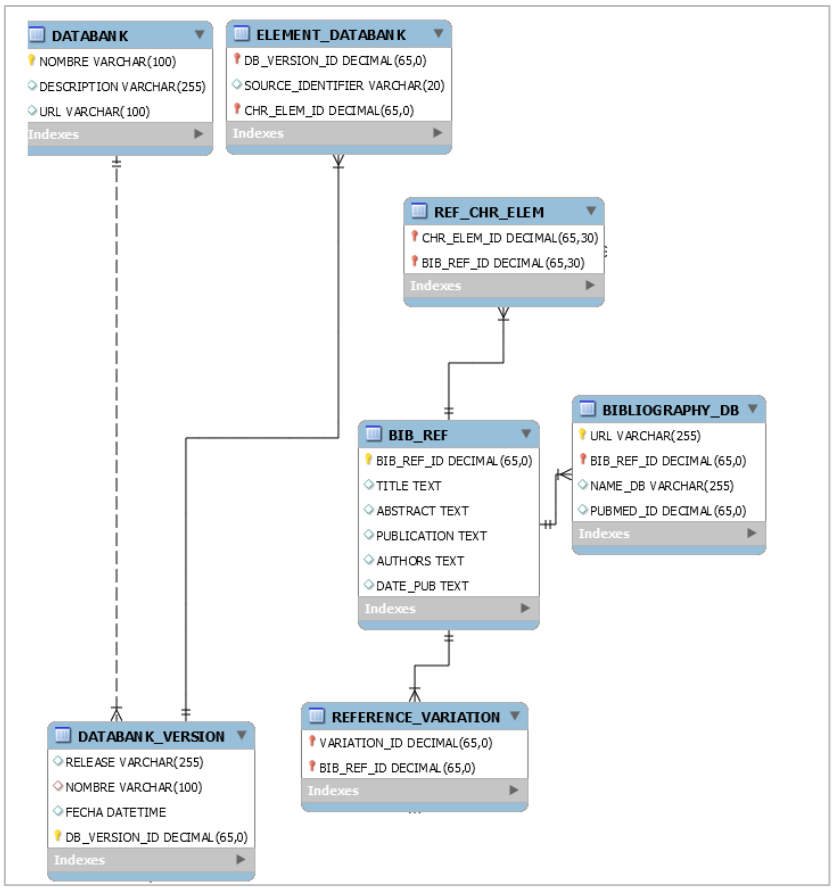

Figura 9. Vista fuente de datos y bibliografía de la DB versión 3. Fuente: (23)

Vista de usuarios y validaciones: Compuesta por las tablas *Curator* y *Validation*, tienen como función validar la relevancia clínica de las variaciones encontradas a través de un especialista médico-científico, cuyos datos de usuario son almacenados en la tabla Curator (Figura 10).

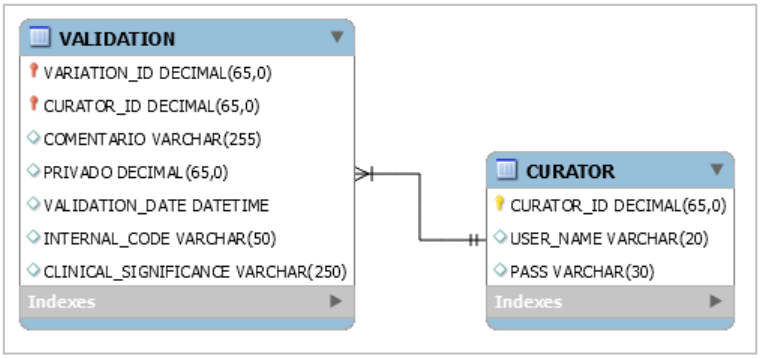

Figura 10. Vista usuarios y validaciones de la DB versión 3. Fuente: (23)

2.2. ARCHIVOS VCF Y HERRAMIENTAS

Los archivos VCF son empleados para la representación y almacenamiento de variaciones de la secuencia de genes. Los archivos están formados únicamente por

texto, sin ningún formato, delimitando la información por tabulaciones. Por su sencillez, se ha convertido en el formato estándar para la descripción de variaciones (24).

Internamente organizan la información en dos bloques, cabecera y registros de variaciones. Los campos de la cabecera están denotados por una doble almohadilla, siendo la mayoría de ellos indicadores de formato y tipo de información aceptada en los registros de variaciones. El único campo obligatorio de la cabecera es *fileformat*, indicador de versión del archivo VCF empleada (11).

La información presentada de cada campo se detalla a continuación (Figura 11):

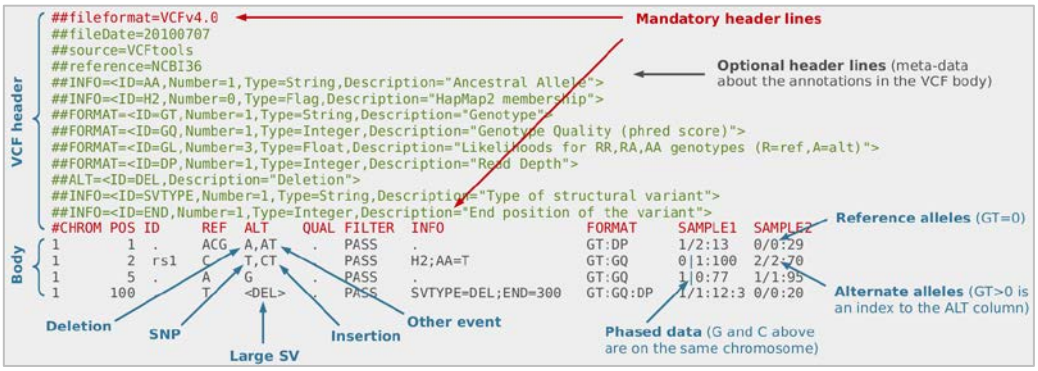

Figura 11. Estructura fichero VCF. Fuente: (25)

Tabla 1. Información representada en un fichero VCF

Columna	Información representada
CHROM	Indica el cromosoma estudiado
POS	Indica la posición de las variaciones. En el caso de los SNP, la posición es la base de referencia con la variante. En inserciones o deleciones, la posición es la base de referencia justo antes de la variante
ID	Identificador único de cada variación
REF	Indica la base o bases de referencia, pudiendo ser A, C, G, T o N
ALT	Indica el tipo de variación encontrada (nucleótido, INV, DEL, etc.). Puede haber varias variaciones separadas por comas.
QUAL	Indica la calidad de la variación ALT en la escala Phred, es decir, la probabilidad de que el nucleótido secuenciado sea erróneo.
FILTER	Indica si ha superado todos los filtros y, en caso contrario, cuáles no.
INFO	Indica información adicional de las variaciones, como el alelo ancestral, el número de veces que cada alelo ALT es representado, etc.
FORMAT	Indica información del genotipo

A continuación se muestra un archivo VFC y la información subyacente a seis variaciones ubicadas en el cromosoma 1 y que afectan a cuatro posiciones nucleotidicas (1, 2, 5 y 100):

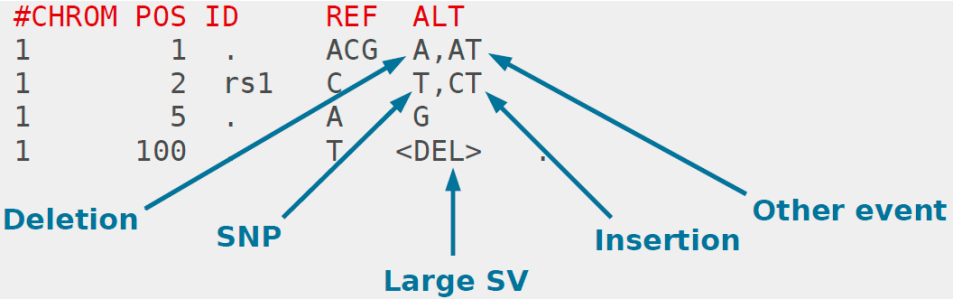

Figura 12. Ejemplo de fichero VCF con variaciones cromosómicas. Fuente: (25)

En la primera posición hay dos eventos (Tabla 2). El primero es una deleción de las bases nitrogenadas (C y G) que están justo después de la posición 1 (A). En el segundo evento dos bases nitrogenadas de la sencuencia original (C y G) han sido sustituidas por otra base nitrogenada (T).

Tabla 2. Eventos producidos en la 1ª variación VCF

POS	REF	ALT	Alineamiento
1	ACG	A	A C G > A ~~C G~~
		AT	A C G > A T

En la segunda variación tenemos dos eventos. En el primero ocurre una SNP (Tabla 3) donde la base nitrogenada original (C) es sustituida por otra base (T). En el segundo evento ocurre una inserción de una base nitrogenada (T).

Tabla 3. Eventos producidos en la 2º variación

POS	REF	ALT	Alineamiento
2	C	T	C > T
		C	C > C T

En la tercera variación encontramos una SNP (Tabla 4), donde la base nitrogenada original (A) es sustituidad por otra diferente (G).

Tabla 4. Evento producido en la 3º variación

POS	REF	ALT	Alineamiento
5	A	G	A > G

En la cuarta posición tenemos una deleción de gran tamaño que comienza en la posición nucleotídica 100 y acaba en posición X (ver tabla 5).

Tabla 5. Evento producido en la variación posición 100

POS	REF	ALT	Alineamiento
100	T	DEL	DEL; END=X

Por otro lado, en Internet existen múltiples aplicaciones, librerías y herramientas de gestión, procesamiento y análisis de variaciones en secuencias de genes, destacando:

- **VCFtools**: Conjunto de utilidades de procesamiento y análisis de archivos VCF desarrollado en Perl por el *1000 Genomes Project* (26). Permite el filtrado de variantes, la comparación y conversión de archivos, así como la creación de subconjuntos de variaciones.
- **vcflib**: Librería desarrollada en C++ para el análisis y manipulación de archivos VCF (24). Permite la validación, filtrado, generación de estadísticas, visualización de haplotipos, intersecciones, uniones y comparación de variaciones (27).
- **Beagle Utilities**: Conjunto de utilidades de procesamiento y análisis de variaciones programadas en Java por la Universidad de Washington (28). Permite
- **Plink/Seq**: Librería escrita en C/C++ para el análisis de variaciones en secuencias de genes con grandes cantidades de datos. Desarrollada por la Universidad de Harvard (29).

2.3. FUENTES DE DATOS

En el siguiente apartado se listan las principales bases de datos biológicas que incluyen datos de secuencias de genomas, nucleótidos, proteínas, estructura de proteínas, de enfermedades genéticas humanas, y bibliográficas, entre otras; así como una breve descripción de las más populares y utilizadas:

Tabla 6. Bases de datos más populares y utilizadas

Nombre	Tipo BD	Origen	Inicio	Estado
Ensembl	Genómica	UE	2000	Activa
UCSC	Genómica	EEUU	2000	Activa
GenBank	Nucleótidos	EEUU	1982	Activa
EMBL	Nucleótidos	UE	1992	Activa
DDBJ	Nucleótidos	Japón	1986	Activa
dbSNP	Polimorfismos	EEUU	1998	Activa
SNPedia	Polimorfismos	EEUU	2006	Activa
Uniprot	Proteínas	UE	1986	Activa
PDB	Proteínas	EEUU	1971	Activa
KEGG	Rutas metabólicas	Japón	1995	Activa
Reactome	Rutas metabólicas	UE	2004	Activa
BioCyc	Rutas metabólicas	EEUU	2005	Activa
PubMed	Bibliográfica	EEUU	1996	Activa
OMIM	Enfermedades genéticas	EEUU	1995	Activa

- **Ensembl**: Esta base de datos de datos, disponible en internet desde el año 2000, recibe el nombre del proyecto de investigación bioinformática Ensembl,

iniciado en 1999 con el apoyo del Instituto Europeo de Bioinformática (EBI-EMBI) y el Instituto Sanger de Cambridge (WTSI). El proyecto Ensembl ofrece a la comunidad científica un punto de acceso a la secuencia del genoma y variaciones de varios organismos, con especial énfasis en el humano y otras especies eucariotas (30). Actualmente trabaja con la versión 38 parche 7 del genoma humano (GRCh38.p7). El acceso a la información de Ensembl se puede realizar descargándose vía FTP toda la base de datos o a través de la herramienta online BioMart, que permite ejecutar consultas y aplicar filtros sobre el conjunto de datos existentes. Asimismo, Ensembl dispone de una API para el lenguaje Perl, un servidor REST y conexión directa online a sus bases de datos MySQL (31).

- **UCSC**: La base de datos de secuencias de genes UCSC se inició en el año 2000 en la Universidad de California, Santa Cruz, Estados Unidos. Ofrece herramientas muy potentes como (13): Genome Browser, para la representación gráfica de genes a través de búsquedas por posición cromosómica, nombre de gen y otros elementos descriptivos; Blat, para la alineación de secuencias; Table Browser, para el filtrado y descarga de contenidos según el contexto de la búsqueda, cuya presentación se realiza en texto plano mediante la tabulación de resultados. Actualmente trabaja con la versión 38 del genoma humano de diciembre de 2013 (GRCh38/hg38). El acceso a los datos se puede realizar mediante conexión directa a su servidor MySQL, descargándose la base de datos mediante el protocolo FTP, o a través de las herramientas descritas anteriormente (32).
- **GenBank**: Es la base de datos genética más conocida y longeva. Gestada en 1982 por el Laboratorio Nacional de Los Alamos de EEUU, actualmente es soportada por el Centro Nacional para la Información Biotecnológica (NCBI) (14) del mismo país. Intercambia información con la base de datos de ADN de Japón (DDBJ) y el Laboratorio Europeo de Biología Molecular (EMBL). Genbank actúa como repositorio de todas las secuencias de proteínas y nucleótidos disponibles, incluyendo secuencias de ARNm con regiones codificantes, ADN genómico correspondiente a uno o varios genes y ARN ribosómico. El acceso de información se puede realizar mediante el buscador web Entrez, a través de la herramienta BLAST o vía FTP. La información obtenida en las búsquedas se

organiza en campos, aunque también permite su representación sin anotaciones en formato FASTA. A través del servidor FTP se permite la descarga en los formatos ASN.1 y GenBank (33).

- **EMBL**: Es una base de datos que incorpora, organiza y distribuye todas las secuencias públicas de ADN disponibles. Pertenece al Laboratorio Europeo de Biología Molecular (EMBL) e intercambia diariamente datos de nuevas secuencias con GenBank y DDBJ. Sin embargo, cada una almacena diferente información sobre los alineamientos de secuencias y las secuencias de proteínas deducidas. Los principales contribuyentes son científicos individuales y grupos de investigación genómicos. El acceso a los datos se puede realizar a través de su página web o vía FTP. EBML dispone de un tipo de archivo de secuencia propio que consta de cuatro bloques: identificación y descripción, citas, características biológicas y la propia secuencia, cuyo inicio está marcado con la etiqueta "SQ" y su final con dos barras "//". Un archivo de secuencia en formato EBML puede contener varias secuencias. Por otro lado, también ofrece la salida en formato FASTA y XML. También dispone de servicios web REST, SOPA y WSDL, y una amplia documentación para su uso (34).
- **DDBJ**: El banco de datos de ADN de Japón mantiene y proporciona servicios de archivado, recuperación y análisis de información biológica. El contenido de las bases de datos de DDBJ se comparte con el Centro Nacional de Información Biotecnológica (NCBI) y el Instituto Europeo de Bioinformática (EBI). Los datos de secuencias son recopilados principalmente por investigadores japoneses, aunque también se admite datos y acceso de investigadores de cualquier país. DDBJ dispone de un servidor FTP, múltiples buscadores online con opciones de búsqueda por palabras clave, taxonómicas, condicionales, por secuencias BLAST, etc. y un API para su uso y explotación mediante herramientas informáticas de desarrollo propio. DDBJ dispone de un formato de representación propio denominado Flat, cuya estructura es muy similar a GenBank (35).
- **dbSNP**: Es una base de datos de polimorfismos documentados de nucleótido simple y múltiples variaciones de pequeña escala iniciada en 1998. Cada variación dispone de una breve descripción, enlaces a la fuente bibliográfica, a

otras bases de datos biológicas relacionadas con esas variaciones, así como herramientas de navegación, visualización y localización dentro del gen y del cromosoma. La información de las variaciones se puede exportar en formato ASN.1, Fasta y XML, y descargarse localmente a través del servidor FTP. dbSNP forma parte del Centro Nacional de Información Biotecnológica (NCBI) de Estados Unidos (36).

- **Uniprot**: Es una base de datos de proteínas fruto de la unión de tres bases de datos: Swiss-Prot, TrEMBL y PIR-PSD. Swiss-Prot y TrEML continúan siendo dos secciones dentro de Uniprot, donde la primera proporciona información de calidad anotada manualmente, y la segunda información anotada automáticamente no revisada procedente de EMBL. Por este motivo Swiss-Prot es mucho más pequeña que TrEMBL. Por otro lado, UniProt se halla integrada con PRIDE, base de datos del EMBL-EBI que cumple con todos los estándares para proteómica basada en espectometrías de masa (37).
- **PDB**: Iniciado en 1971, PDB se trata del único banco de proteínas que ofrece información 3D sobre proteínas y ácidos nucleicos. PDB es de acceso gratuito, sin coste alguno y su actualización es semanal. Forma parte del proyecto wwPDB, que unifica los distintos bancos de proteínas de Estados Unidos (RCSB PDB), de Europa (PDBe) y de Japón (PDBj). PDB emplea un formato de archivo propio en el que se denotan las coordenadas atómicas, conectividad atómica, nombres de las moléculas, citas bibliográficas, etc. (38).
- **PubMed**: Lanzada en 1996, Pubmed es una base de datos de referencias bibliográficas relacionadas con la bioquímica, biología celular y la medicina. Incluye títulos, resúmenes y autores de los artículos publicados. Contiene las referencias de MEDLINE, base de datos de bibliografía médica producida por la Biblioteca Nacional de Estados Unidos, así como referencias de revistas que no están en MEDLINE. Pubmed asigna un identificador único denominado PMID a cada cita de un artículo.
- **OMIM**: Esta base de datos cataloga aquellas enfermedades con un componente genético conocido, y cuando es posible realiza la asociación de genes con el genoma humano. Disponible en internet desde 1987, permite realizar búsquedas web a partir del nombre de enfermedades, genes, fenotipo y localización cromosómica, entre otras. Además, dispone de un servidor REST

que permite la conexión y uso con herramientas bioinformáticas de desarrollo propio. El resultado de la búsqueda es seleccionable en distintos formatos: XML, JSON, JSONP, Python o Ruby. Aunque su uso es gratuito, es necesario el registro y validación por parte del equipo de OMIM para hacer uso de este servicio web (39).

CAPÍTULO 3: ESTUDIO BIOLÓGICO

3.1. DOMINIO GENÓMICO

Uno de los mayores desafíos de la presente Tesis de Máster es mejorar el entendimiento del genoma humano. Su estudio permite una mayor compresión y avance en el conocimiento de aquellos aspectos directamente relacionados con la salud humana. A continuación, se describirá un conjunto de conceptos relevantes, los cuales poseen un factor importante para entender el comportamiento del genoma humano, por ejemplo, el ADN, genes, proteínas, rutas metabólicas, variaciones genéticas, genotipo y fenotipo, así como su funcionamiento, comportamiento y evolución.

3.1.1. ADN

El Ácido Desoxirribonucleico es una compleja molécula formada por dos cadenas complementarias de nucleótidos compuestas por cuatro tipos de moléculas denominadas adenina, citosina, guanina y timina, comúnmente representadas por su abreviatura como A, C, G y T, respectivamente. Generalmente se les denomina bases porque químicamente son bases nitrogenadas, midiéndose el tamaño de la molécula de ADN en base, kilobases o megabases, haciéndose referencia a la unidad o a los miles o millones de bases nitrogenadas que la forman (40).

Figura 13. Detalle de la cadena de ADN y elementos que lo forman. Fuente: (41)

Morfológicamente tiene forma de doble hélice como si de una escalera de caracol se tratara (Figura 13), donde las cadenas o hebras de ADN son sucesiones de pares de bases que se empaquetan en el interior de la doble hélice de la forma más favorable energéticamente. Los miembros de cada par de bases sólo pueden unirse si las dos cadenas son antiparalelas, esto es, si la polaridad de una cadena está orientada de manera opuesta a la otra, de manera de que, si el orden de la secuencia de una de las cadenas es A-T-G-C-A, el complementario será T-A-C-G-T, uniéndose A siempre con T, y C con G mediante enlaces de hidrógeno (Figura 14).

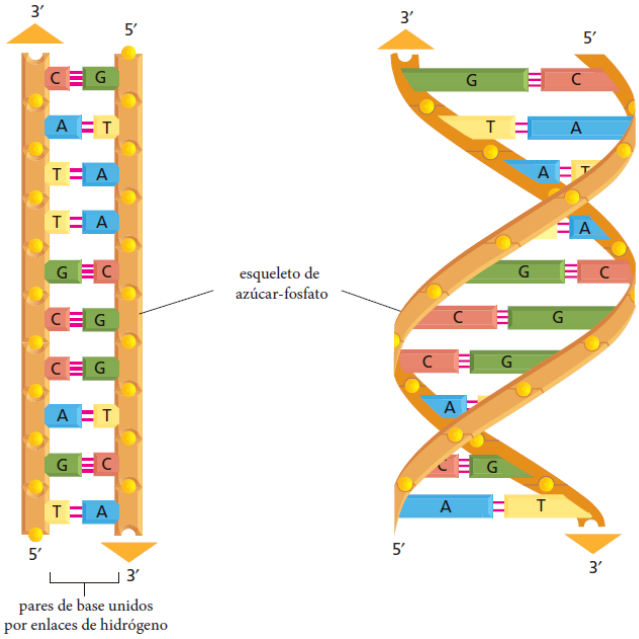

Figura 14. ADN de doble cadena y doble hélice de ADN. Fuente: (41)

31.2. GENES

Los genes son elementos que contienen la información que determina las características de un ser vivo. Están constituidos por ADN y se localizan en los cromosomas (Figura 15). Se consideran la unidad fundamental de la herencia porque codifica un producto (proteína) cuya función e información es transmitida a la descendencia (42).

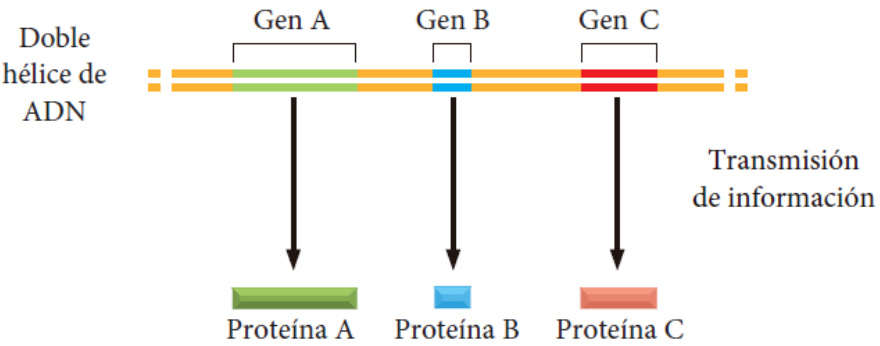

Figura 15. Expresión genética. Fuente: (41)

Existen dos grandes grupos de genes:

- **Genes estructurales**: Situados en regiones de ADN no repetitivo y cuya función es la de codificar la producción de moléculas de proteínas.
- **Genes reguladores**: Cuya función es regular o suprimir la actividad de uno o más genes estructurales.

Asimismo, existen tres grupos de genes según el tipo de expresión que realizan:

- **Específicos de tejido**: Corresponden al 80% del genoma y codifican proteínas que mayoritariamente se expresan en un tejido determinado.
- **Mantenimiento**: Corresponden al 20% del genoma. Codifican proteínas necesarias para las funciones estructurales y metabólicas básicas de las células.
- **Homeóticos**: Codifican los factores de transcripción que se unen a ciertas secuencias de ADN, activando y desactivando la transcripción, proceso por el cual el ADN sintetiza ARN.

3.1.2. ALELOS

Los genes suelen presentarse en versiones diferentes denominadas alelos. Todos lo alelos para un gen en particular se localizan en un sitio específico del cromosoma denominado locus, y controlan el mismo rasgo o carácter, poseyendo la misma función, pero con acción diferente. Por ejemplo, un gen determinado que codifica el pelaje de un animal puede existir como alelos que codifican el pelaje de color negro o el pelaje blanco (42).

Los alelos se denominan con una o más letras y algún símbolo y se clasifican en alelos dominantes y alelos recesivos en función de quién se expresa en presencia de ambos alelos y de manera simultánea.

- **Alelos dominantes**: son aquellos que solo necesitan una copia del gen para expresarse y se nombran con letras mayúsculas. Cuando dos alelos tienen la misma actividad se denominan *codominantes o isomorfos*.
- **Alelos recesivos**: son aquellos que necesitan dos copias del gen para expresarse y se nombran con letras minúsculas.

Según la localización de las mutaciones, dos alelos pueden ser *homoalelos o isoalelos* cuando se presentan en el mismo sitio, o *heteroalelos* cuando las tienen en diferentes lugares. Por otro lado, los alelos pueden ser *amorfos* cuando carecen de actividad genética o *hipoamorfos* cuando tienen niveles bajos de actividad genética.

3.1.4. PROTEÍNAS

Las proteínas son biomoléculas formadas por la unión de aminoácidos. A nivel de distribución, una proteína puede contener miles de aminoácidos de los 20

diferentes existentes que utilizan la mayoría de seres vivos, lo que no significa que puedan producirse todas las combinaciones posibles, ya que existirían combinaciones sin sentido. Cabe recordar que la secuencia de aminoácidos que contiene una proteína viene determinada por secuencia de ADN (40).

Los 20 aminoácidos diferentes necesarios para la síntesis de las proteínas en todos los seres vivos son (Tabla 7):

Tabla 7. Lista de aminoácidos y abreviaturas de unas tres letras

Aminoácido	
Ácido aspártico	Asp / D
Ácido glutámico	Glu / E
Arginina	Arg / R
Lisina	Lys / K
Histidina	His / H
Asparagina	Asn / N
Glutanina	Gln /Q
Serina	Ser / S
Treonina	Thr / T
Tirosina	Tyr / Y

Aminoácido	
Alanina	Ala / A
Glicina	Gly / G
Valina	Val / V
Leucina	Leu / L
Isoleucina	Ile / I
Prolina	Pro /P
Fenilalanina	Phe / F
Metionina	Met / M
Triptófano	Trp / W
Cisteína	Cys / C

El proceso por el cual el ADN se traduce en una secuencia de proteínas está compuesto por dos fases, transcripción y traducción. En la primera, la secuencia de ADN es utilizada para sintetizar ARN, mientras que en la segunda se utiliza la secuencia de ARN para sintetizar la proteína. Este proceso se explicará con detalle en el siguiente apartado.

3.1.5. DEL ADN A LAS PROTEÍNAS

Muchas propiedades de las proteínas están determinadas por la secuencia de aminoácidos que la forman. Esta secuencia es definida por las instrucciones genéticas que contiene el ADN.

El ADN no sintetiza directamente la proteína, sino que una secuencia determinada es copiada en otro tipo de ácido nucleico, denominado *ácido ribonucleico* (ARN), mediante el proceso de *transcripción*. Posteriormente, estas copias de segmentos cortos de ARN son utilizados como moldes para dirigir la síntesis de la proteína mediante el proceso de *traducción* (43).

TRANSCRIPCIÓN

En el proceso de transcripción, una proteína llamada ARN Polimerasa transcribe el ADN a ARN mensajero (ARNm). Este ARNm es una copia complementaria de la hebra del ADN que contiene la secuencia del gen (Figura 16).

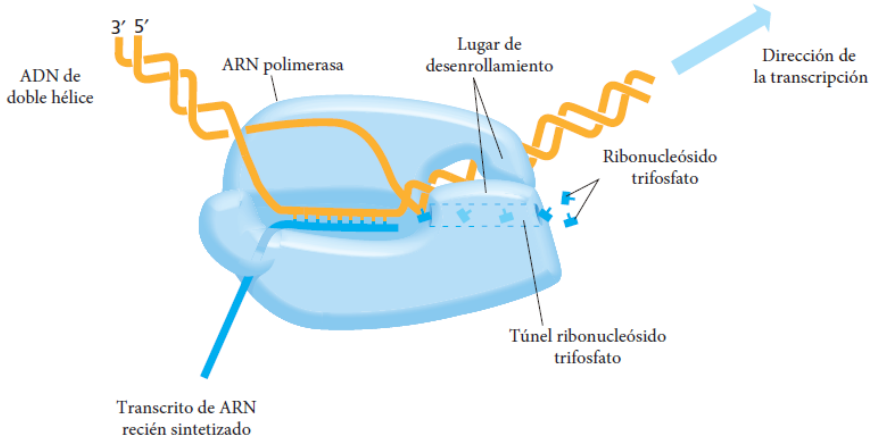

Figura 16. Detalle proceso transcripción. Fuente: (41)

TRADUCCIÓN

Una vez se ha producido un ARNm, el siguiente paso es utilizar la información almacenada en su secuencia de nucleótidos para sintetizar una proteína. Sin embargo, el ARNm tiene 4 nucleótidos diferentes mientras que en una proteína existen 20 aminoácidos distintos. Este hecho provoca que no se pueda realizar una correspondencia directa uno a uno entre los nucleótidos del ARN a aminoácido de las proteínas. Para ello, existen una reglas que permiten la traducción de una secuencia nucleótido de un gen a una secuencia de aminoácidos de una proteína, conocidas comúnmente como código genético (44).

La siguiente imagen (figura 26) recoge el código genético universal común para todos los organismos. La unidad de codificación está compuesta por tres nucleótidos y se denomina codón. Los codones corresponden a un aminoácido determinado, excepto tres tipos, UAA, UAG y UGA (Figura 17).

GCA GCC GCG GCU	AGA AGG CGA CGC CGG CGU	GAC GAU	AAC AAU	UGC UGU	GAA GAG	CAA CAG	GGA GGC GGG GGU	CAC CAU	AUA AUC AUU	UUA UUG CUA CUC CUG CUU	AAA AAG	AUG	UUC UUU	CCA CCC CCG CCU	AGC AGU UCA UCC UCG UCU	ACA ACC ACG ACU	UGG	UAC UAU	GUA GUC GUG GUU	UAA UAG UGA
Ala	Arg	Asp	Asn	Cys	Glu	Gln	Gly	His	Ile	Leu	Lys	Met	Phe	Pro	Ser	Thr	Trp	Tyr	Val	stop
A	R	D	N	C	E	Q	G	H	I	L	K	M	F	P	S	T	W	Y	V	

Figura 17. Código genético universal. Fuente: (41)

Los codones de una molécula de ARNm no reconocen directamente al aminoácido que especifican, aunque estos señalan dónde empezar (codón de inicio) y finalizar la síntesis de proteínas (codón de paro). Por contrario, la traducción del ARNm en proteínas depende de determinadas moléculas adaptadoras que sí son capaces de reconocer y unirse tanto al codón como al aminoácido. Estas moléculas son denominadas ARN de transferencia (ARNt).

La traducción de un ARNm a proteína requiere de una macromolécula denominada ribosoma que se desplaza sobre el ARNm capturando moléculas del ARNt, manteniéndolas en posición y uniendo los aminoácidos que transportan para la generación de proteínas (Figura 18).

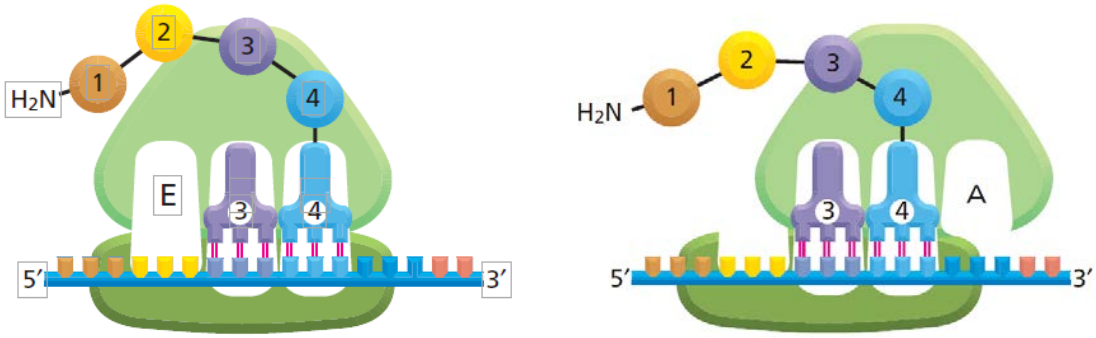

Figura 18. Ribosoma durante el proceso de traducción. Fuente: (41)

3.1.6. RUTAS METABÓLICAS

Las rutas metabólicas se desarrollan en lugares específicos de las células y son una serie de reacciones químicas catalizadas enzimáticamente, existiendo en la ruta un precursor que se convierte en producto, a través de una serie de intermediarios denominados metabolitos. Las rutas metabólicas pueden ser convergentes, si una gran variedad de componentes celulares acaba en una ruta final común con pocos productos finales, o divergentes si se forman muchos productos finales diferentes a partir de una pequeña variedad de precursores (41).

Las rutas metabólicas están reguladas por el principio de máxima economía, donde la velocidad de degradación de moléculas (catabolismo) está controlada por las necesidades de la célula y no por la concentración de moléculas a degradar.

Las rutas metabólicas están estrechamente relacionadas con la regulación de la expresión génica, ya que están catalizadas por enzimas, que son proteínas, y su

presencia o ausencia para completar la ruta depende de si previamente han sido transcritas.

3.1.7. VARIACIONES

La mayoría de las especies tienen un número característico de cromosomas, cada uno con su propio tamaño y estructura. En el caso de los seres humanos, nuestro juego de cromosomas o cariotipo está formado por 46 cromosomas organizados en 22 autosómicos y un par sexual (Figura 19).

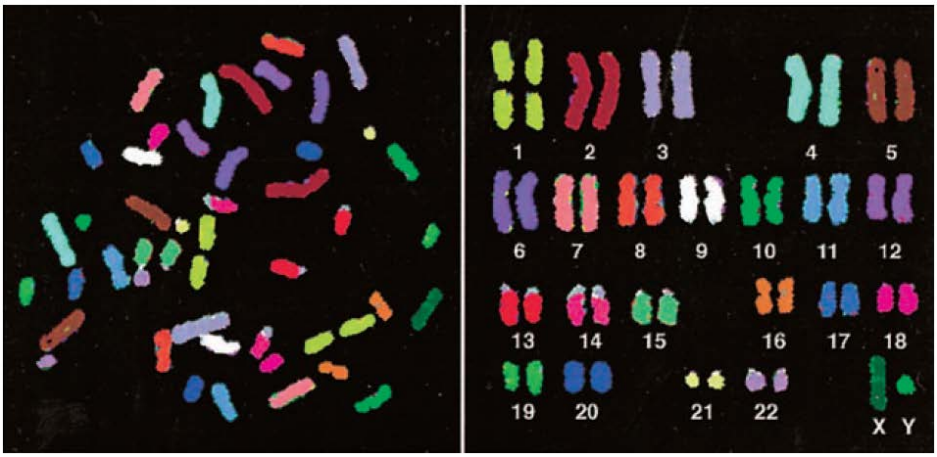

Figura 19. Cariotipo humano formado por 46 cromosomas. Fuente: (41)

Existen variaciones en el número y estructura de los cromosomas, pudiendo perder o ganar partes, alterarse la secuencia de genes dentro del cromosoma e incluso perderse o ganar cromosomas enteros. Estas variaciones son denominadas *mutaciones cromosómicas* y existen tres tipos: *mutación de reordenamiento cromosómico* si se altera la estructura de cromosomas, *aneuploidías* si se altera la estructura de los cromosomas, y *poliploidías* si agrega o elimina cromosomas individuales (41).

Las mutaciones son divididas en dos niveles, mutaciones génicas si la lesión sobre el ADN es relativamente pequeña y afecta a un único gen, y mutación cromosómica si la alteración genética es a gran escala.

Las mutaciones génicas son clasificadas en tres tipos básicos, sustitución de bases, inserciones y deleciones:

- **Sustitución de bases**: Existen dos tipos de sustituciones y ambas afectan a un solo nucleótido en el ADN.

- **Transición**: Cuando se reemplaza una base nitrogenada purina (A, G) por otra diferente, o alternativamente una pirimidina (T, C, U) se reemplaza por otra pirimidina diferente.
- **Transversión**: Cuando una purina es reemplazada por una pirimidina o una pirimidina es reemplazada por una purina.

- **Inserciones** y **deleciones**: Cuando se produce una adición o eliminación de uno o más pares de nucleótidos. Las inserciones y deleciones dentro de secuencias que codifican proteínas pueden provocar mutaciones de cambio de marco de lectura de un gen, alterando los aminoácidos codificados por los codones.

TIPO DE MUTACIÓN	C O N S E C U E N C I A S							
SIN MUTACIÓN	ADN ARNm Proteína Símil lingüístico	GAT CUA Leu dos	GGT CCA Pro por	CGT GCA Ala dos	CAG GUC Val son	ACG UGC Cys más	TCT AGA Arg que	TGT ACA Thr uno
TRANSICIÓN	ADN ARNm Proteína Símil lingüístico	GAT CUA Leu dos	GGT CCA Pro por	CGT GCA Ala dos	CGG GCC **Ala** sen	ACG UGC Cys más	TCT AGA Arg que	TGT ACA Thr uno
TRANSVERSIÓN	ADN ARNm Proteína Símil lingüístico	GAT CUA Leu dos	GGT CCA Pro por	CGT GCA Ala dos	CCG GGC **Gly** sin	ACG UGC Cys más	TCT AGA Arg que	TGT ACA Thr uno
INSERCIÓN	ADN ARNm Proteína Símil lingüístico	GAT CUA Leu dos	GGT CCA Pro por	CGT GCA Ala dos	TCA AGU **Ser** sso	GAC CUG Leu nmá	GTC CAG Gln squ	TTG T AAC A Asn eun o
DELECIÓN	ADN ARNm Proteína Símil lingüístico	GAT CUA Leu dos	GGT CCA Pro por	CGT GCA Ala dos	CAG GUC Val son	ACT UGA **Stop**	CTT GAA	GT CA

Figura 20. Tipo de mutaciones y consecuencias. Fuente: (41)

Por otro lado, dentro del genoma humano existen diferencias de una sola base (adenina (A), timina (T), citosina (C) o guanina (G)) entre alelos homólogos denominados polimorfismo de un nucleótido único (SNP). Originados por mutación, los SNP (Figura 21) son heredados como variantes alélicas aunque no se suelen producir diferencias fenotípicas.

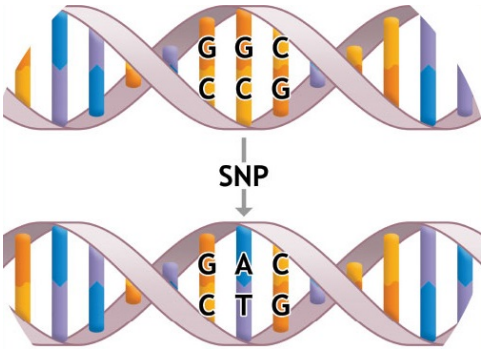

Figura 21. Diferencia de una sola base SNP. Fuente: (41)

El conjunto específico de SNP y otras variantes genéticas en un cromosoma individual o en una parte del cromosoma se denomina haplotipo. Dentro de este, los SNP están ligados físicamente y, por consiguiente, tienden a heredarse juntos. Sin embargo, pueden originarse haplotipos nuevos por mutación y entrecruzamiento.

Los SNP son valiosos marcadores en estudios de relación, ya que, cuando un SNP se relaciona físicamente a un locus que causa una enfermedad, se elevan las probabilidades de que sea heredado junto con el alelo que la produce.

3.1.8. GENOTIPO Y FENOTIPO

El término genotipo hace referencia a un conjunto de alelos que constituyen a un individuo o a una especie, generalmente referidos a uno o varios genes relevantes en un determinado contexto. El fenotipo, en cambio, es la manifestación externa de los caracteres hereditarios que posee cada individuo en coordinación con el ambiente (45).

El genotipo determina el potencial para el desarrollo y establece ciertos límites o fronteras para este. La manera en la que el fenotipo se desarrolla dentro de estos límites está determinada por el efecto de otros genes y los factores ambientales.

Aunque el fenotipo está influido por el genotipo, los organismos no le transmiten el fenotipo a la siguiente generación, pero el genotipo sí. La distinción entre ambos conceptos es uno de los principios cruciales de la genética moderna.

CAPÍTULO 4: SOLUCIÓN PROPUESTA

El origen de la información que nutre el Modelo Conceptual del Genoma Humano (MCGH) reside en las fuentes de datos biológicas empleadas: *Gene*, *Clinvar*, *dbSNP*, *Nucleitode*, *Protein* y *Pubmed*. Estos datos se rigen bajo una estructura diversa en función del propósito y la información que ofrecen, ya sea de tipo *genómico*, *nucleótidos*, *proteínas*, *polimorfismos* o *bibliográficas*, ofreciendo también distintos formatos de salida como ASN.1, FASTA, y XML (Tabla 8). Así mismo, no existe un sistema de referencias que vincule las distintas fuentes de datos entre sí de forma ágil y automática, y que permita la extracción de la información subyacente de una variación concreta, como su posición cromosómica, genes relacionados, fenotipos asociados, etc., sino que es necesaria la existencia de un componente humano que realice este trabajo, como la ejecución de carga en la base de datos (HGBD) de forma manual para su posterior explotación y generación/obtención de conocimiento.

Tabla 8. Formatos de datos de las DB utilizadas.

Formato	Descripción
ASN.1	Norma de descripción de información compuesta por tres tipos de datos. *Primitivos*, con un único valor; *Construidos*, usados para crear conjuntos de datos, como arrays y tablas; y *Definidos*, similares a los compuestos, pero más descriptivos y con una etiqueta para identificarlos (46).
FASTA	Formato utilizado para la representación de secuencias de nucleótidos en texto plano. La secuencia almacenada comienza con una descripción en la cabecera con utilidad descriptiva (47).
XML	Lenguaje de marcado empleado para dotar de estructura a la información, tanto en procesos de intercambio de información entre diferentes plataformas, y como de almacenamiento de datos (48).

Esta búsqueda, identificación, carga y explotación de información genómica son las etapas fundamentales de la Metodología SILE, cuyo objetivo es proveer un pronóstico médico preventivo. La Metodología SILE, descrita a continuación en el apartado 5.2, realiza los pasos de Búsqueda (*Search*), Identificación (*Identification*), Carga (*Load*) y Explotación (*Exploitation*) con un equipo de profesionales e investigadores que son apoyados por un grupo de ingenieros biomédicos, biológicos y biotecnólogos para la identificación y validación

individual de genes y variaciones. Esta información es cargada de forma "selectiva" en la HGDB, la cual es una proyección del MCGH.

En este sentido, la solución propuesta se organiza en dos tareas fundamentales:

1. Automatizar la etapa de carga de la metodología SILE mediante un software capaz localizar toda la información del MCGH desde los distintos repositorios seleccionados (*dbSNP*, *Clinvar*, *Gene*, *Nucleotide*, *Protein* y *Pubmed*), dando como parámetro de entrada una variación genética con suficiente peso científico-medico.
2. Realizar una búsqueda intensiva de genes y variaciones ligados a *cataratas*, y posteriormente emplear el prototipo de carga desarrollado para finalizar el proceso ETL en la base de datos del genoma humano. Finalmente, se validará la información cargada utilizando la herramienta "*VarSearch*", concluyendo de esta forma la metodología SILE y facilitando un nuevo diagnóstico genético basado en modelos conceptuales.

5.1. METODOLOGÍA SILE

SILE (*Search-Indentification-Load-Exploitation*) es una metodología compuesta por cuatro etapas: búsqueda, identificación, carga y explotación (10). La búsqueda de información se realiza en repositorios genómicos ampliamente utilizados en la comunidad científica. Para la tarea de identificación se integra un grupo de expertos en áreas de biología molecular, biomedicina y/o biotecnología provenientes de distintos centros hospitarlarios con los que el Centro PROS tiene colaboraciones, como el Hospital Universitario y Politécnico de La Fe[1]; el Instituto de Investigación Sanitaria – INCLIVA[2]; así como empresas especialiadas en estudios genómicos, como Imegen[3] y TellmeGen[4]. Todo ello con el objetivo de realizar una identificación y validación precisa de genes y variaciones asociadas a una enfermedad de origen genético. Los datos obtenidos en estas etapas previas serán cargados en la base de datos del genoma humano (HGDB) y finalmente explotados mediante el framework genómico "*VarSearch*".

[1] http://www.hospital-lafe.com
[2] http://www.incliva.es
[3] https://www.imegen.es
[4] http://www.tellmegen.com

La metodología SILE se aplicará en este Trabajo de Fin de Máster a la patología de la "catarata congénita", pudiendo ser esta: a) recesivas (R); b) autosómica dominante (AD); c) esporádicas (E); y ligadas al cromosoma X (XL).

5.2.1. SEARCH

En primer lugar, se listan todos los genes asociados de manera directa o indirecta a las cataratas congénitas junto a una pequeña descripción (Tabla 9).

Tabla 9. Genes asociados a las Cataratas Congénitas

Gen	Descripción
BCOR	Gen implicado en el síndrome de microftalmia de Lenz. Su mutación cambia un único aminoácido en el correpresor BCL6 alterando la estructura de la proteína. Este correpresor desempeña un papel esencial en la formación de los ojos.
BFSP2	Este gen codifica las proteínas faquinina y felensina localizadas en las fibras del cristalino.
CHMP4B	Este gen codifica una proteína relacionada con la cromatina y la modificación de cuerpos multivesiculares.
CRYAA	Este gen codifica una proteína de choque térmico llamada alfa-cristalina. Su función es proteger los componentes celulares en condiciones de stress. También ejerce un papel estabilizador en las células del cristalino. Su expresión es fundamental para el correcto desarrollo embrionario del ojo.
CRYAB	Gen implicado en la creación de la proteína alfa B cristalina
CRYGC	Este gen codifica una serie de proteínas de la familia beta y gamma cristalina, cuya función es preservar la transparencia y el correcto índice de refracción del cristalino.
CRYBB1	Gen implicado en la creación de la proteína cristalina beta 1
CRYBB2	Gen implicado en la creación de la proteína cristalina beta 2.
CRYBB3	Gen implicado en la creación de la proteína cristalina beta 3.
CRYBA4	Gen implicado en la creación de la proteína cristalina beta 4.
CRYGD	Gen implicado en la creación de la proteína cristalina gamma D.
CRYGS	Este gen codifica la proteína gamma más importante en el tejido del cristalino en un ojo adulto.
GCNT2	Este gen codifica la enzima responsable de la formación los antígenos i e I, responsables en la generación de anticuerpos. El primero es abundante en los primeros meses de vida, hasta los 18 meses, momento en el que aumenta I en detrimento de i.
GJA3	Este gen codifica una proteína conexina que forma parte de las uniones gap de las fibras de cristalino.
GJA8	Este gen codifica una proteína transmembrana responsable del crecimiento del cristalino y la maduración de sus fibras.
HSF4	Este gen codifica la proteína de factor de transcripción de choque térmico 4, activada cuando se produce un aumento de temperatura, como en situaciones de estrés.
LIM2	Este gen codifica una proteína localizada en las uniones de las fibras del cristalino, actuando como receptor de calmodulina y teniendo una importante función en el desarrollo del cristalino y el proceso de

	cataratogénesis.
MAF	Este gen codifica una proteína que participa en varios procesos celulares, como en el desarrollo embrionario de las fibras del cristalino.
NHS	Este gen codifica una proteína que contiene cuatro sitios de señalización nuclear. La proteína codificada tiene su principal actividad durante el desarrollo de los ojos, los dientes y el cerebro. Relacionado con el síndrome Nance-Horan.
PITX3	Este gen codifica una proteína RIEG/PITX, de la familia de las proteínas homeodominio Bicoid. Actúa como factor de transcripción y regula la expresión de genes procolágeno-lisina 4-dioxigenasa, enzimas que catalizan la hidroxilación de la lisina. Está implicado en el desarrollo de los órganos del ojo.
VSX2	Este gen codifica una proteína homeodominio descrita como un factor de transcripción específico de retina.

Estos datos se han obtenido de dos revisiones (49) (50) y diversos artículos sobre cataratas congénitas (51), (52), (53), (54), (55), (56), (57), (58). (59), (60).

Además, hay estudios que destacan la utilidad clínica de las variaciones genómicas, indicando que la tasa de detección de mutaciones en pacientes afectados es del 70% utilizando técnicas masivas de secuenciación, por lo cual, los estudios genómicos serían útiles para la detección temprana de la enfermedad y mejorar el asesoramiento clínico, ya que proporcionan información diagnóstica adicional significativa, permitiendo la existencia de una medicina personalizada a nivel genómico (2). Además, actualmente el *Eye Genetics Research Group* del *Children's Medical Research Institute* (61) está ampliando el conocimiento genómico de las cataratas congénitas, lo cual ofrece oportunidades de análisis de esa información.

5.2.2. IDENTIFICATION

Una vez localizados los genes relacionados con las cataratas, se procede a la identificación de variaciones (Tabla 10), localización y tipo de catarata (AD, autosómica dominante; R, recesiva; E esporádica; LX, ligada al cromosoma X).

Tabla 10. Identificación de Variaciones (catarata congénita)

Gen	Cromosoma	Tipo	Variación SNP	Cambio nucleótido
BCOR	Xp11.4	X	rs864309680 rs144736705 rs864309702	c.4390_4393del c.3277G>A c.1136_1139del
BFSP2	3q22	AD	rs104893685	c.859C>T
CHMP4B	20q11.2	AD AD	rs118203965 rs118203966	c.386A>T c.481G>A
CRYAA	21q22.3	R	rs74315440	c.27G>A

		AD	rs397515624	c.34C>T
		AD	rs397515626	c.62G>A
		AD	rs74315441	c.145C>T
		AD	rs397515623	c.160C>T
		AD	rs398122947	c.292G>A
		AD	rs74315439	c.346C>T
		AD	rs121912973	c.347G>A
		AD	rs864309685	c.142T>G
		AD	rs397515625	c.61C>T
CRYAB	11q23.1	AD	rs387907336	c.418G>A
		AD/R	rs387907337	c.58C>T
		AD	rs144451841	c.320G>T
CRYGC	2q33	AD	rs28931604	c.502C>T
		AD	rs137853924	c.385G>T
		AD	rs104893618	c.13A>C
	2q33.3	AD	rs587778872	c.497C>T
		AD	rs398122944	c.471G>A
	2q33-q35	AD	rs398122392	c.470G>A
CRYBB1	22q12.1	R	rs74315488	c.658G>T
		E/AD	rs864309682	c.368G>A
CRYBB2	22q11.2	AD	rs74315489	c.463C>T
	22q11.23	E/AD	rs864309683	c.556T>C
CRYBB3	22q11.2	AD	rs74315490	c.493G>C
			rs587777601	c.581T>A
	22q11.23	AD	rs864309700	c.634T>C
CRYBA4	22q12.1	AD	rs74315487	c.206T>C
			rs74315486	c.281T>C
CRYGD	2q33	AD	rs121909596	c.176G>A
			rs28931605	c.70C>T|A
	2q33-q35		rs398122948	c.402C>A
CRYGS	3q27	AD	rs104893736	c.53G>T
GCNT2	6p24	R	rs56141211	c.1043G>A
			rs55940927	c.1148G>A
GJA3	13q12.1	AD	rs121917825	c.560C>T
			rs121917823	c.188A>G
			rs121917827	c.227G>A
			rs140332366	c.563A>C|T
	13q12.11	AD	rs864309687	c.260C>T
			rs864309691	c.176C>T
			rs864309694	c.7G>C
GJA8	1q21.2	AD	rs80358200	c.262C>T
			rs80358203	c.68G>C
			rs80358204	c.131T>A
			rs397515627	c.566C>T
			rs80358205	c.593G>A
			rs80358202	c.741T>G
			rs864309677	c.119C>T
		E/AD	rs864309684	c.89dupT
		AD/AR	rs864309688	c.134G>C
		E/AD	rs864309703	c.151G>A

HSF4	16q22	AD	rs121909049	c.56C>A
			rs121909050	c.256A>G
			rs121909048	c.341T>C
		X	rs28937573	c.355C>T
LIM2	19q13.4	R	rs121913555	c.439T>G
MAF	16q23.2	AD	rs864309678	c.819G>C
			rs786205222	c.908A>C
			rs864309692	c.915C>T
			rs864309695	c.880C>T
	16q23		rs786205221	c.895C>A
			rs121917736	c.890A>G
			rs121917735	c.863G>C
MIP	12q13.3	E/AD	rs864309693	c.97C>T
NHS	Xp22.1	XL	rs104894881	c.115C>T
			rs398124608	c.400delC
			rs132630322	c.1117C>T
		AD	rs864309679	c.2707delG
	Xp22.13	XL/AD	rs111534978	c.3624C>A
PITX3	10q24.3	AD	rs104894175	c.38G>A
VSX2	14q24.3	R	rs121912543	c.599G>C | A

De todos los genes localizados y variaciones identificadas, a continuación, se listan aquellas que han sido objeto de estudio reciente, comprendiendo para ello como intervalo los años 2014 – 2016 (Tabla 11).

Tabla 11. Variaciones detectadas para la catarata congénita entre 2014 y 2016

Gen	Cromosoma	Tipo	Variación SNP	Cambio nucleótido
BCOR	Xp11.4	X	rs864309680	c.4390_4393del
			rs864309702	c.1136_1139del
CRYAA	21q22.3	AD	rs864309685	c.142T>G
		AD	rs397515625	c.61C>T
CRYAB	11q23.1	AD	rs144451841	c.320G>T
CRYGC	2q33.3	AD	rs587778872	c.497C>T
CRYBB1	22q12.1	E/AD	rs864309682	c.368G>A
CRYBB2	22q11.23	E/AD	rs864309683	c.556T>C
GJA3	13q12.11	AD	rs864309687	c.260C>T
			rs864309691	c.176C>T
			rs864309694	c.7G>C
GJA8	1q21.2	AD	rs80358205	c.593G>A
			rs864309677	c.119C>T
		E/AD	rs864309684	c.89dupT
		AD/AR	rs864309688	c.134G>C
		E/AD	rs864309703	c.151G>A
MAF	16q23.2	AD	rs864309678	c.819G>C
			rs786205222	c.908A>C
			rs864309692	c.915C>T
			rs864309695	c.880C>T
MIP	12q13.3	E/AD	rs864309693	c.97C>T
NHS	Xp22.13	AD	rs864309679	c.2707delG

		XL/AD	rs111534978	c.3624C>A

Estas variaciones serán empleadas en la fase de explotación con tal de poner a prueba el framework y su utilidad de pronóstico preventivo. Se han filtrado teniendo en cuenta su presencia en más de un estudio y para los casos donde su relación causal estaba fundamentada (2).

5.2.3. LOAD

La etapa de carga (*metodología SILE*) en la base de datos del genoma humano está compuesta por tres subprocesos denominados *extracción*, *transformación* y *carga*. Estos subprocesos forman parte del concepto ETL, (*Extraction-Transformation-Load*) (62), que permiten y garantizan la extracción de información desde distintas fuentes de datos para su análisis, conversión y cumplimiento de las restricciones del sistema destino, es decir, la base de datos del genoma humano.

La estructura de tablas de la HGDB descrita en el apartado 2.4, actualmente da cabida a la vista estructural, vista variaciones, vista fuente de datos y bibliografía, y a la vista de usuarios y validaciones, siendo su descripción la siguiente:

- Vista Estructural: Modela la estructura del genoma de los organismos.
- Vista de Variaciones: Representa las diferencias encontradas en la secuencia de ADN.
- Vista fuente de datos y bibliografía: Informa del origen de los datos que almacenará el modelo.
- Vista de usuarios y validaciones: Permite que personal cualificado pueda certificar las variaciones identificadas en las secuencias de ADN del genoma.

Respecto al origen de la información, para la presente Tesis de Máster se han utilizado los repositorios biológicos del Centro Nacional para la Información Biotecnológica de Estados Unidos, NCBI (63) (del inglés *National Center for Biotechnology Information)*. Estas son, "*Gene"* (64), que ofrece toda la información asociada a los genes; "*dbSNP*" (65) y "*ClinVar*" (66), que muestran información de las variaciones, tipos, cambios de nucleótidos, etc., "*Nucleotide*" (67), que relaciona datos y secuencias del genoma con los genes y sus transcritos, y "*Pubmed*" (68), base de datos de artículos de investigación de referencia. Aunque cada fuente contiene información muy concreta y alientan a una localización de datos trivial,

las características de las vistas exigen la necesidad de consultar varios repositorios y poder relacionarlos de forma ágil y concurrente.

Por otro lado, el lenguaje de programación empleado para el desarrollo de la aplicación y los subprocesos de Extracción-Transformación-Carga ha sido Java en la versión 8.144. La elección de este lenguaje es debido a su amplia documentación, librerías existentes, así como por su robustez, seguridad y portabilidad, ya que es independiente de la arquitectura y plataforma.

Dado que el formato de salida de los repositorios es el lenguaje de meta-marcado XML (69), Java dispone de una API de procesamiento XML denominada SAX (70), de acceso libre y basado en eventos. Esto es, que SAX analiza el documento secuencialmente y conforme localiza un inicio o final de etiqueta lanza un evento para leer los atributos de la misma, o realizar otra acción. SAX se caracteriza por tener un consumo de memoria reducido, ya que, a diferencia de otros procesadores de XML como DOM, JDOM o XOM, SAX no genera un árbol de nodos que permita modificar o crear desde cero un XML, sino que está orientado a la lectura de archivos XML de gran tamaño.

Las clases que forman cada elemento de las vistas del MCGH se describen gráficamente mediante el modelado UML (Figuras 22-27), indicando en la parte superior los atributos y en la parte inferior los métodos de acceso y modificación de datos. Dependiendo de la clase, existen atributos relacionales empleados en la fase de extracción de datos que se detallará en el siguiente apartado (5.2.3.2). Esto es debido a que hay clases cuyo origen de información proviene de distintas fuentes de datos.

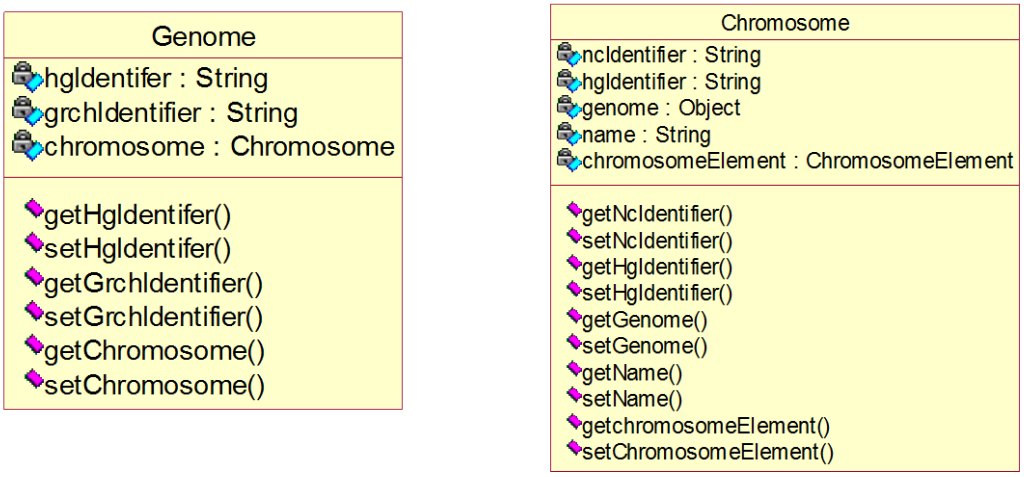

Figura 22. Clases Genome y Chromosome

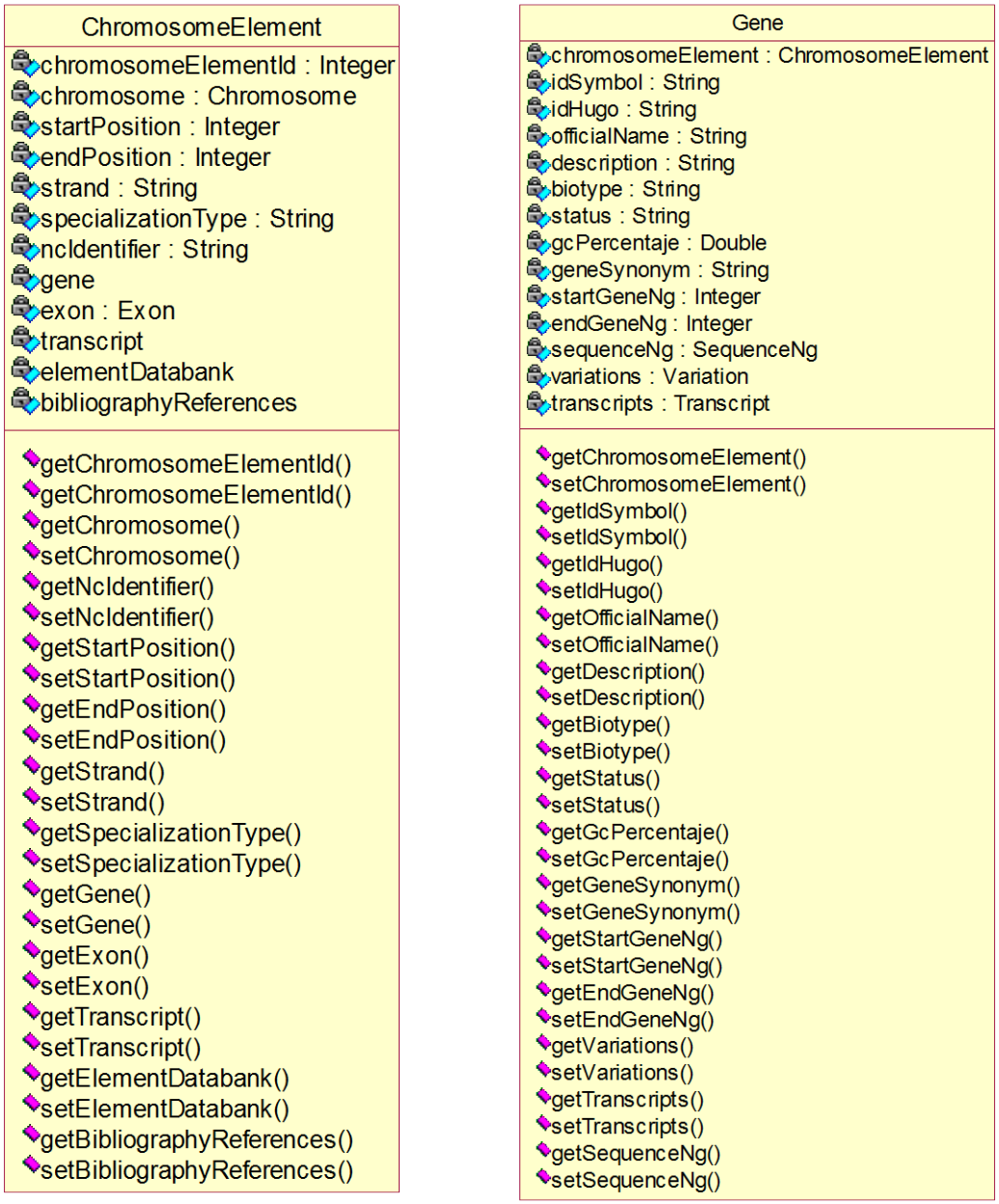

Figura 23. Clases ChromosomElement y Gene

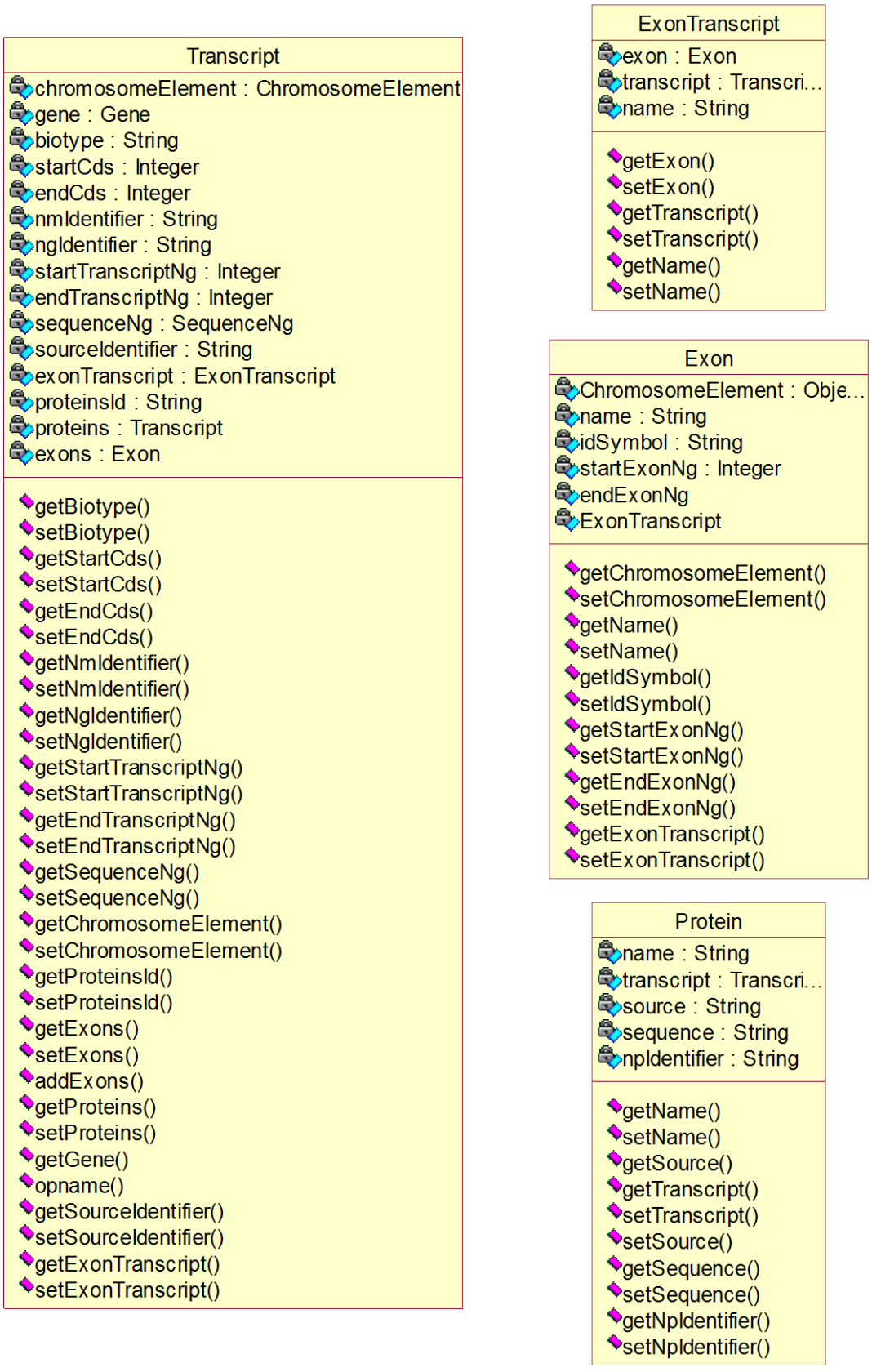

Figura 24. Clases Transcript, ExonTranscript, Exon y Protein

Figura 25. Clases SequenceNg, PreciseNg, Variation y Precise

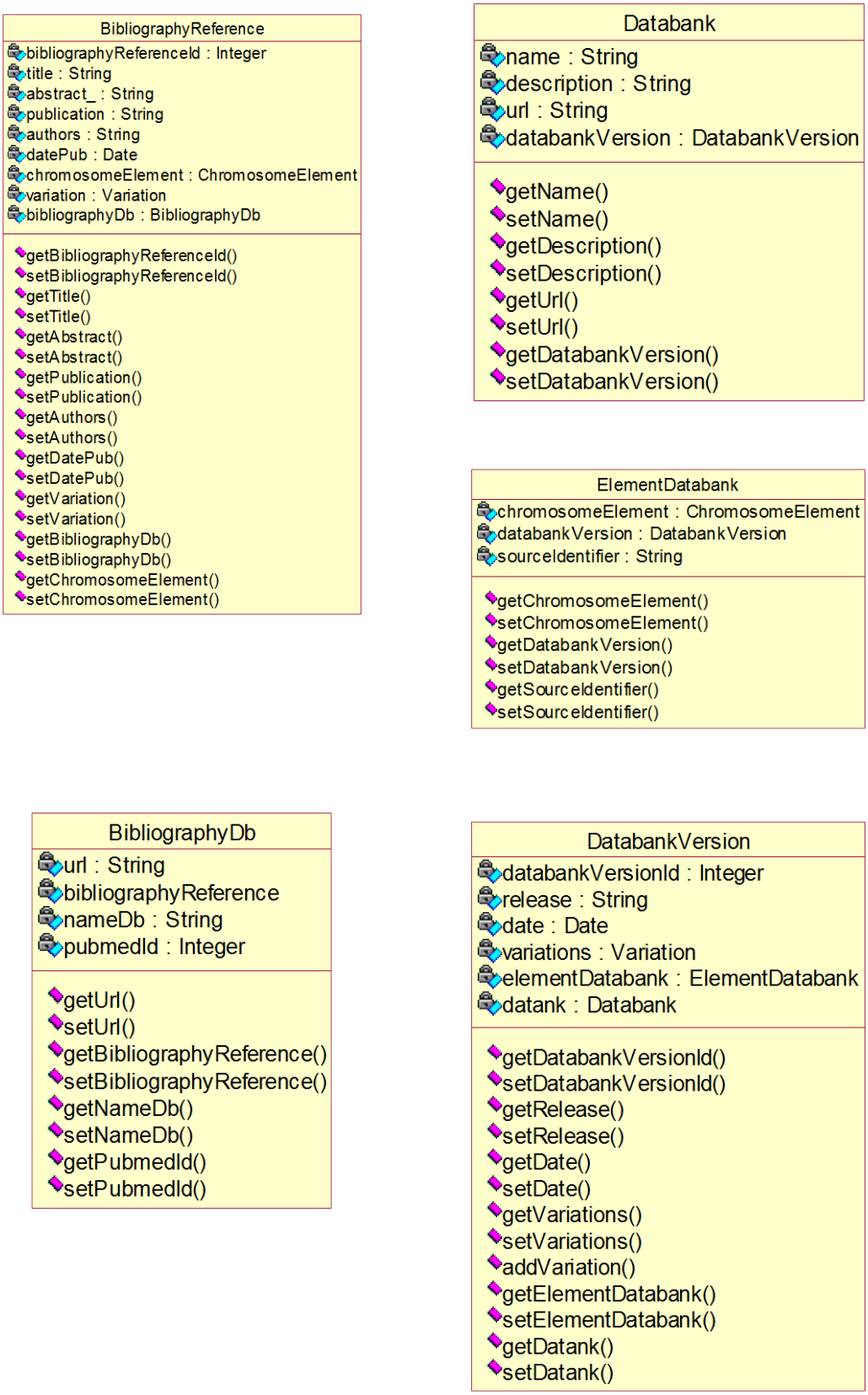

Figura 26. Clases BibliographyReference, BibliographyDb, Databank, ElementDatabank y DatabankVersión

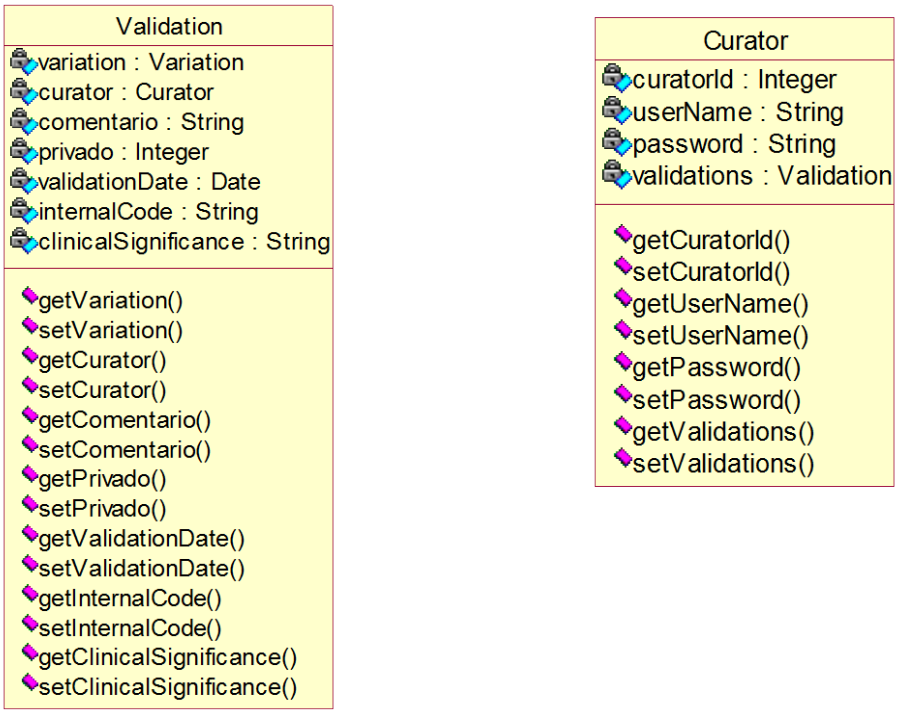

Figura 27. Clases Validation y Curator.

5.2.4. PROCESO ETL

Esta sección describe las etapas que forman el proceso ETL. En ella, se especifica la función de cada uno de sus procesos internos, siendo estos, la *Extracción*, *Transformación* y *Carga* de datos en la HGDB.

5.2.4.1. EXTRACCIÓN DE DATOS

La extracción es la primera etapa del proceso ETL y su principal función es la de extraer los datos desde las distintas fuentes de información, pudiendo ser repositorios locales o ubicados en Internet. Además de acceder a ellos, la etapa de extracción debe analizarlos, comprobar su estructura, y en caso de no cumplir con los requisitos necesarios, rechazarlos, ya que durante esta fase los datos son convertidos a un formato previo que sirve de antesala para la etapa de transformación.

En este sentido, el origen de la información requerida proviene de los repositorios biológicos y científicos del NCBI: *Gene*, *ClinVar*, *dbSNP*, *Nucleotide* y *Protein*. El acceso a ellos se lleva a cabo mediante una consulta HTTP cuya ULR está formada por cuatro elementos (Figura 28), que permiten unos argumentos concretos (Tabla 12):

1. *Cuerpo de la URL que de acceso al explorador Efetch del NCBI.*
2. *Selección de la base de datos que se desea consultar.*
3. *Identificador del elemento que se está buscando.*
4. *El formato de salida de la consulta en caso de existir resultados.*

Figura 28. Estructura de consulta al NCBI

Tabla 12. Argumentos permitidos en la URL de consulta

Argumento	Parámetros permitidos	Descripción
db	gene snp pubmed nuccore protein	Selección de la base de datos a consultar.
id	Identificador existente	Identificador del elemento a consultar.
retmode	xml txt json	Formato de salida de los datos. Por defecto es XML. También permite, según la base de datos, texto plano y JSON.

En el caso de *ClinVar*, la construcción de la URL varía ligeramente. Mientras que el acceso a *dbSNP* se realiza con el identificador único de la variación, el paso a *Clinvar* puede derivar en una o más consultas con uno o más identificadores dependiendo de si la variación realiza uno o más cambios nucleótidos, como se puede visualizar en el siguiente ejemplo (Figura 29):

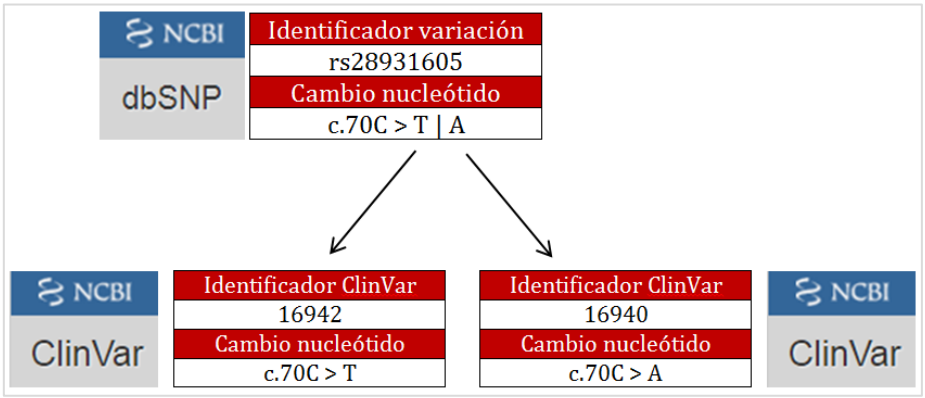

Figura 29. Identificadores entre *dbSNP* y *Clinvar* según la variación.

Para conocer estos identificadores, debe realizarse una consulta intermedia en la que se mantiene el cuerpo de la URL anterior, pero con nuevos argumentos:

1. *Base de datos de origen: dbSNP*
2. *Base de datos de destino: ClinVar*
3. *Identificador de variación de referencia en dbSNP.*

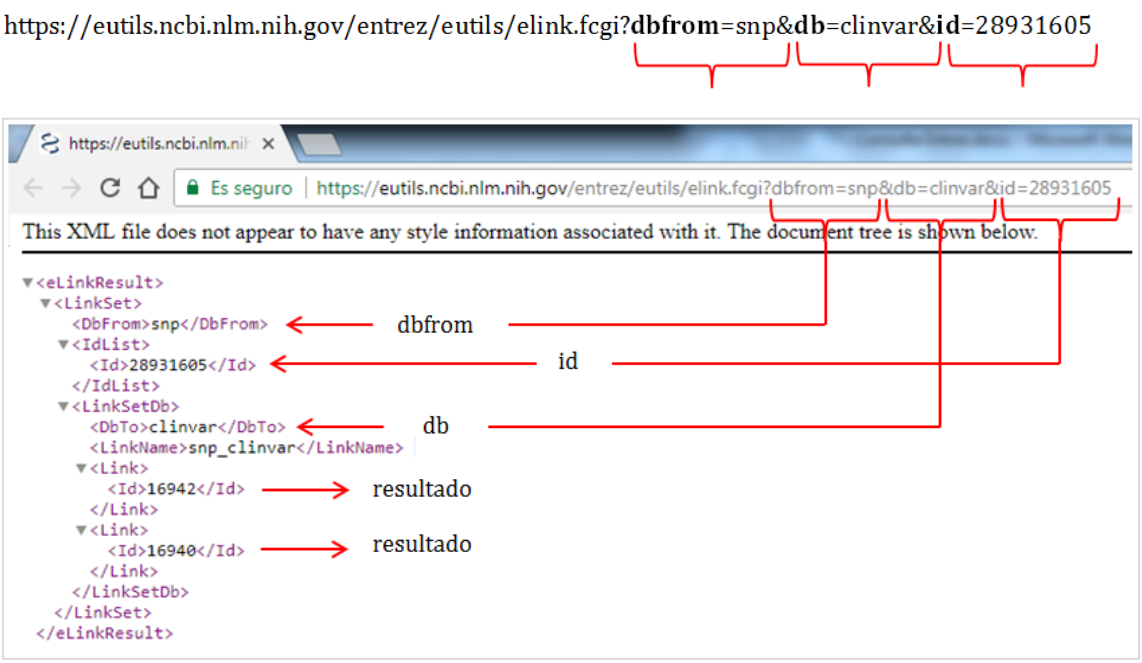

Figura 30. Ejemplo de consulta y resultados obtenidos en Clinvar

Esta consulta (Figura 30), genera un XML con las variaciones de *ClinVar* dentro de las etiquetas *<Link><Id> y </Id></Link>.* Además, se observan los parámetros de consulta empleados, como la base de datos de origen (*SNP*), el identificador (28931605) y la base de datos de consulta (ClinVar).

Una vez obtenidos los identificadores Clinvar el siguiente paso es parametrizar la URL sobre dicha base de datos, cuya estructura sería la siguiente (Figura 31):

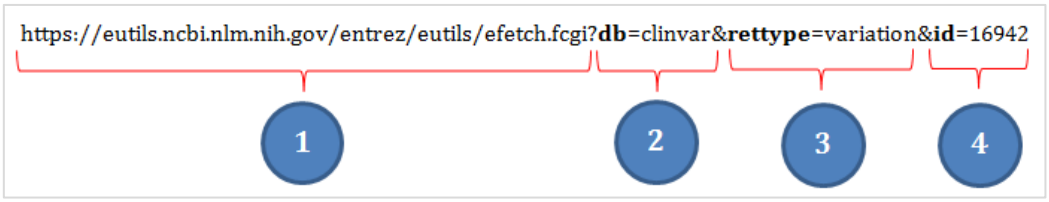

Figura 31. Ejemplo de consulta a ClinVar

1. *Cuerpo de la URL que de acceso al explorador Efetch del NCBI.*
2. *Selección de la base de datos que se desea consultar: ClinVar.*
3. *Tipo de registro devuelto: Variation.*
4. *Identificador de la variación de que se quiere obtener información: 16942 y 16940.*

5.2.4.2. EXTRACCIÓN DE DATOS:

Como se comentó anteriormente, la información obtenida de los repositorios *ClinVar*, *dbSNP*, *Gene*, *Nucleotide*, *Protein* y *Pubmed* es en formato XML bajo unas consultas parametrizadas según el tipo de base de datos. A continuación, se mostrará la estructura de esa información y su destino dentro del MCGH. Como se podrá observar, hay clases en las que la información puede proceder de uno o más repositorios. Para más ser descriptivo este apartado, por cada repositorio consultado y su información acotada, se indicará la URL de consulta, qué vistas alimentan y bajo qué etiquetas XML se encuentran la información.

Tabla 13. Extracción de datos desde *dbSNP*

Repositorio	**dbSNP**
URL de consulta	https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=snp&*id*=***rsVariationId***&retmode=xml
Información obtenida	**Precise**: Flanking left, Flanking right, Aln_Quality
Dato de estudio	**rsVariationId = 28931605**

Del repositorio *dbSNP* se obtiene la calidad del alineamiento en el gen y la secuencia de nucleótidos localizados a izquierda y derecha de la variación. Estos datos solo se encuentran en *dbSNP* (Tabla 13). A continuación se muestra bajo qué etiquetas se localiza la información (Figura 32):

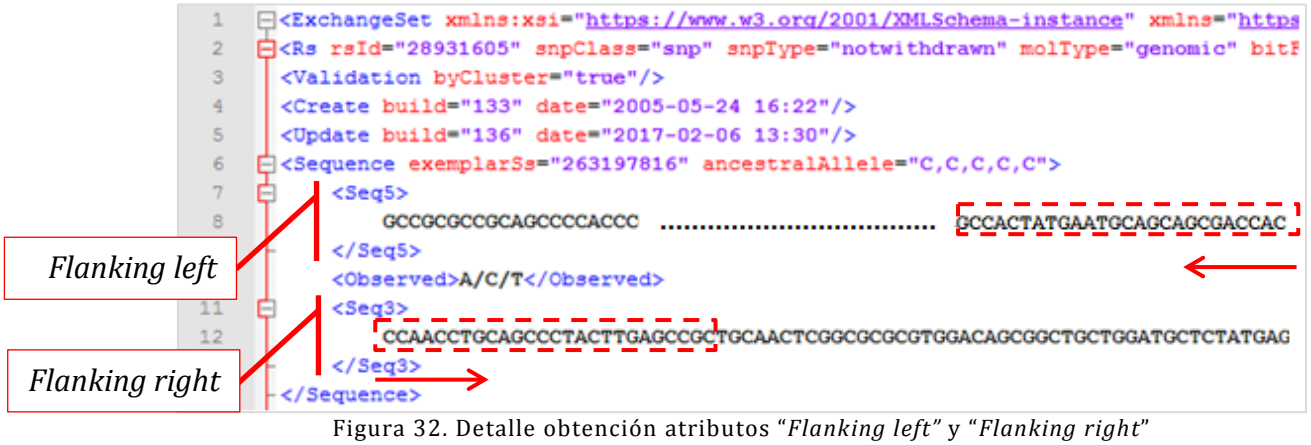

Figura 32. Detalle obtención atributos "*Flanking left*" y "*Flanking right*"

Los atributos (valores) de "*Flanking left*" y "*Flanking right*" están almacenados en las etiquetas *Seq5* y *Seq3*, pero debido a la direccionalidad de la hélice de ADN, la secuencia de nucleótidos 5' (*Flanking right*) se obtendrá leyendo los últimos 20 caracteres de la cadena, mientras que para la secuencia 3' (*Flanking Left*) se leerán los primeros 20 caracteres.

La calidad del alineamiento dentro del gen es referenciada en el atributo *alnQuality,* perteneciente de la etiqueta *MapLoc*, que a su vez es etiqueta hija de *PrimarySequence*. La obtención es rápida y sencilla, ya que no precisa de un procesamiento previo y el acceso es directo (Figura 33).

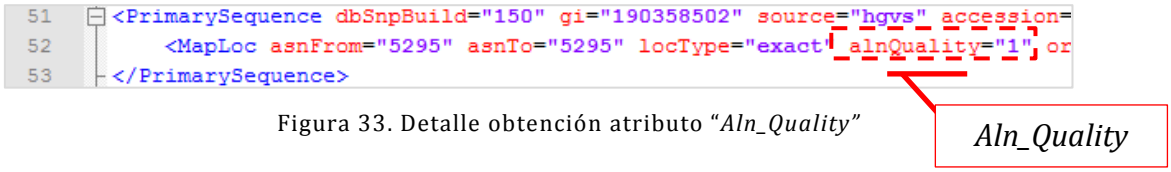

Figura 33. Detalle obtención atributo "*Aln_Quality*"

Tabla 14. Extracción de datos desde *ClinVar*

Repositorio	**ClinVar**
URL de consulta	https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=clinvar&rettype=variation&id=**clinVariationId**
Información obtenida	**Variation**: Description, Nc_Identifier, Ng_Identifiers, Others_Identifiers, Clinically_Imporant, Omim. **Precise**: Position, Specialization_Type, Ins_Sequence, Ins_Repetition, Num_Bases, **PreciseNg**: Ng_identifier, Position **Phenotype**: Nombre **Chr_Elem**: Start_Position, End_Position, Strand **Bibliography_Db**: Pudmed_Id **Genome**: Grch_Identifier
Dato de estudio	**clinVariationId = 16942**

Del repositorio ClinVar se obtiene la mayoría de información biológica de las clases *Variation*, *Precise*, *PreciseNg* y *Phenotype*, pertenecientes a la Vista *Variation*, pero también facilita datos de las clases *Gene*, *Chr_elem*, *Bibliogrpahy_Db* y *Genome* de la *Vista Estructural*, así como identificadores que serán utilizados para la formulación de URLs (Tabla 14).

De las primeras líneas del documento XML (Figura 34), leyendo el atributo "*VariationName*" del elemento *VariationReport* se obtiene el nombre de la variación y el identificador del transcrito, que será empleado para formular la URL de consulta al repositorio *Nucleotide* para extraer toda su información asociada. Del elemento *Gene* se obtiene el identificador del gene asociado a la variación, que será empleado como identificador en la URL de consulta del repositorio *Gene*, así como la hebra en la que se encuentra la variación. El dato de la hebra será procesado en la etapa de transformación para cumplir con las especificaciones del MCGH, que denota a la hembra con M (*minus*) y P (*plus*).

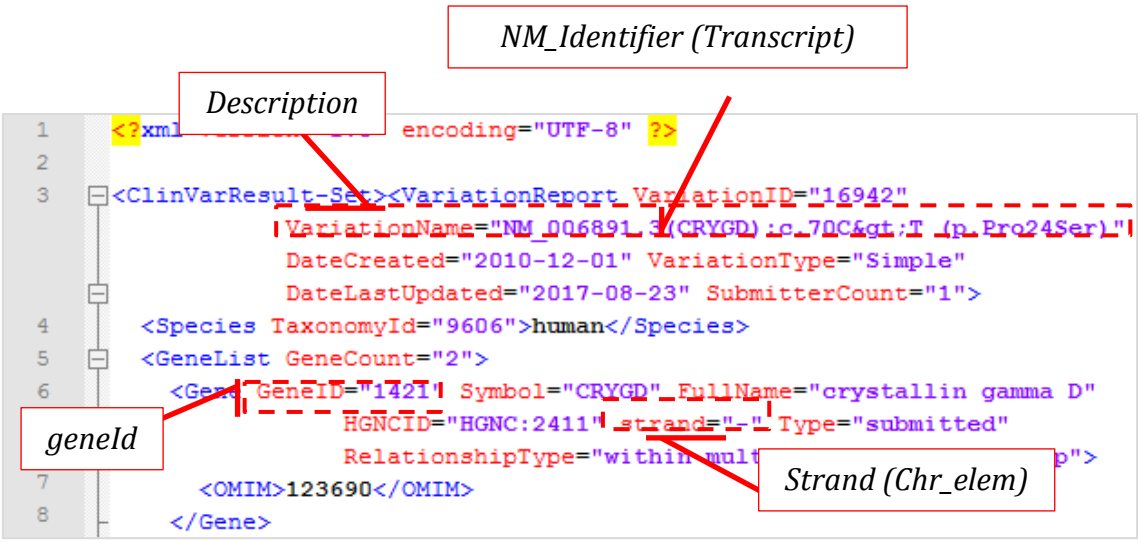

Figura 34. Detalle obtención atributos "*Description*", "*geneId*", "*NM_Identifier*" y "*Strand*"

Por otro lado, el tipo de variación que tuvo lugar dentro del genoma es descrito por la etiqueta *VariationType* (Figura 35). Este dato será procesado en la etapa de transformación con el fin de asociarse a los tipos de variaciones que se contempla para la clase "*Precise*" dentro del MCGH, que son *Deletion*, *Indel* e *Insertion*.

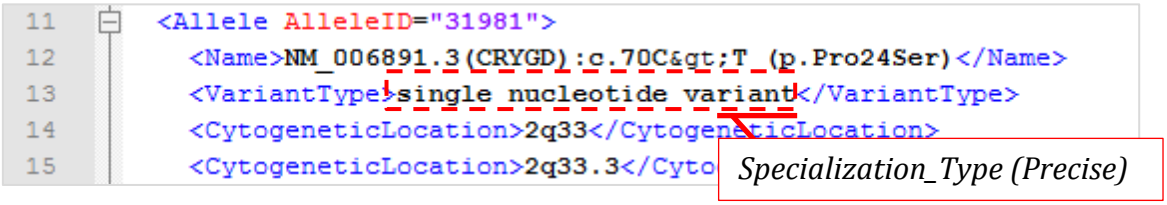

Figura 35. Detalle obtención atributo "*Specilization_Type*"

A continuación, dentro de la etiqueta *SequenceLocation*, se encuentra la posición de la variación dentro de la secuencia del cromosoma, la secuencia de nucleótidos insertados en dicha secuencia, el identificador actual del genoma y el identificador del cromosoma, todo ello a través de los atributos "*Assembly*", "Accession", "*start*", y "*arternateAllele*" (Figura 36).

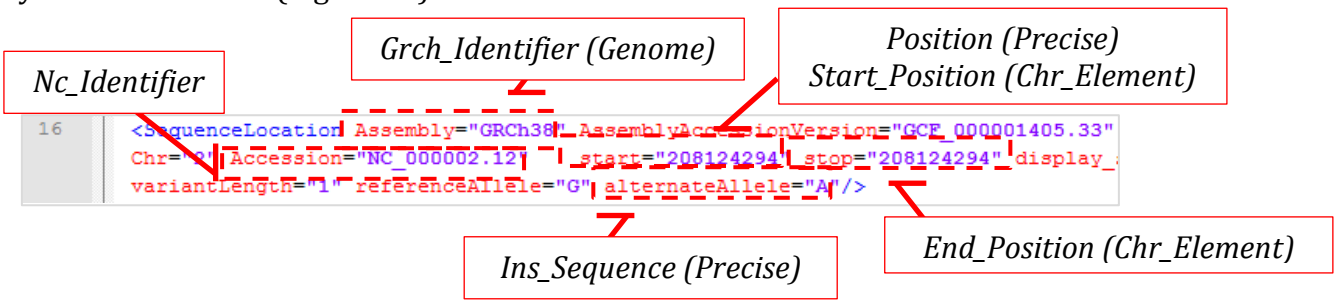

Figura 36. Detalle obtención atributos "*Nc_Identifier*", "*Grch_Identifier*", "*Ins_Sequence*" y "*Positions*"

Por otro lado, las etiquetas *HGVSList*, hijas del elemento HGVS, contienen los identificadores alternativos con los que las fuentes de datos denominan a esta variación (Figura 37).

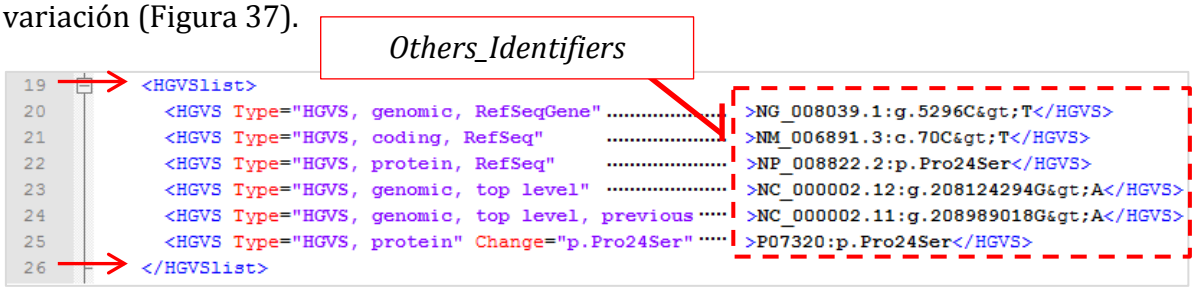

Figura 37. Detalle obtención atributo "*Others_Identifiers*"

Una vez procesada la etiqueta *HGVSList*, a continuación, se encuentra el identificador con el que *OMIM* referencia la variación de estudio. El dato de interés se encuentra en el atributo "*ID*" del elemento *XRef*, que a su vez pertenece a *XRefList* (Figura 38).

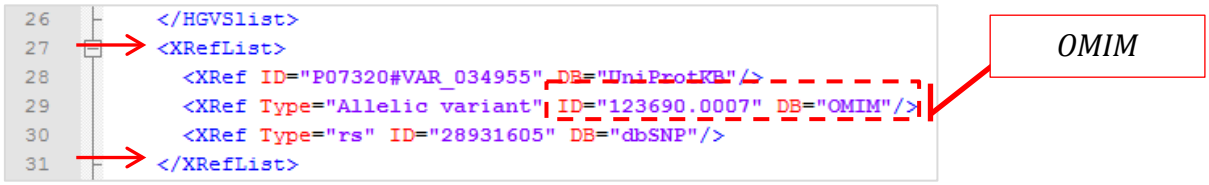

Figura 38. Detalle obtención atributos "*OMIM*"

Por otro lado, de la etiqueta *ObservationList* se puede extraer la información referente a la importancia clínica de la variación y el fenotipo asociado leyendo el elemento *Description* y el atributo "*Name*" de *Phenotype* (Figura 39).

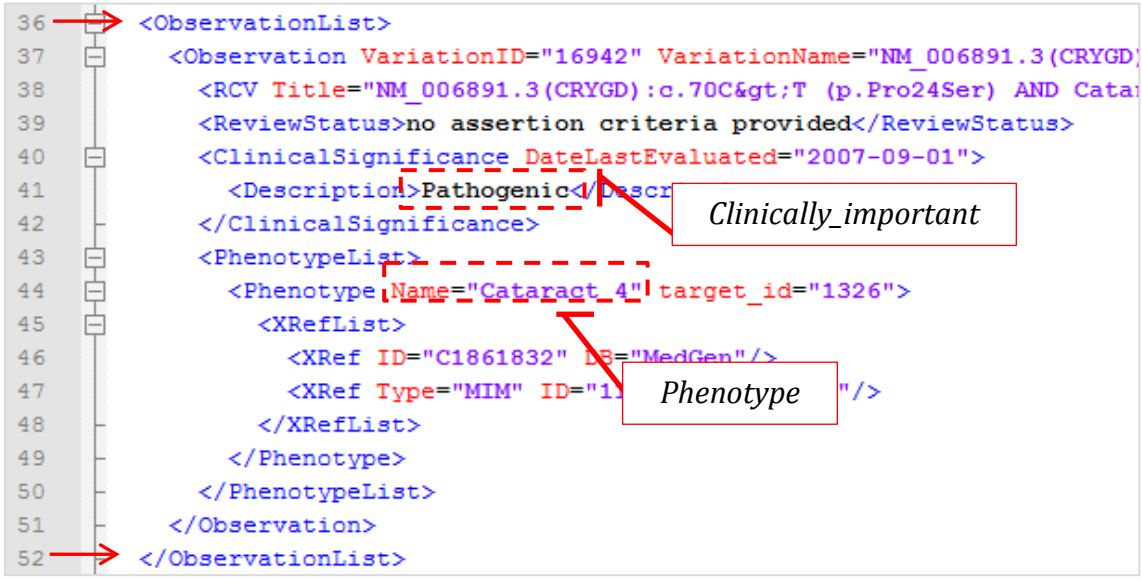

Figura 39. Detalle obtención atributos "*Clinically_important*" y "*Phenotype*"

Concluyendo la extracción del repositorio *ClinVar*, cabe destacar que las publicaciones científicas que documentan la variación de estudio están

referenciadas mediante el identificador que *Pubmed* asigna a todas ellas. Para extraer este dato, hay que analizar el XML y buscar aquellas etiquetas denominadas *ID* cuyo atributo sea "*Pubmed*". A continuación, se muestra un fragmento del XML (Figura 40) con las referencias de las publicaciones que servirían para construir la URL de consulta al repositorio *Pubmed* para la extracción de toda la información que contiene la Vista Bibliográfica.

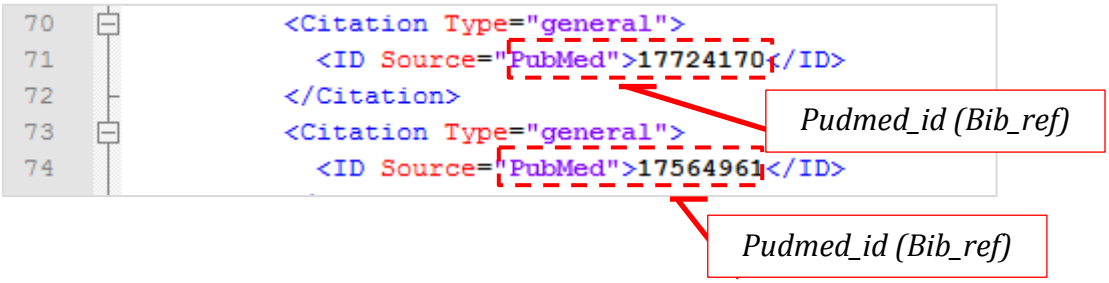

Figura 40. Detalle obtención atributos "*Pubmed_id*"

Tabla 15. Extracción de datos de *Gene*

Repositorio	**Gene**
URL de consulta	https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=gene&***id***=***geneId***&retmode=xml
Información obtenida	**Gene:** Id_Hugo, Official_Name, Description, Biotype, Status, Gene_Synonym, Start_GeneNg, End_GeneNg
Dato de estudio	**geneId** = **1421** (obtenido de Clinvar)

La clase "*Gene*", compuesta por un total de 10 atributos, obtiene la totalidad de su información del repositorio *Gene* del NCBI (Tabla 15). El acceso a esta fuente de datos se realiza mediante el identificador del gen que fue previamente extraído de *ClinVar*. En las primeras líneas del documento XML (Figura 41), el elemento *Entrezgene_type* contiene el tipo de codificación del gen, tanto en texto leyendo a través el atributo "*value*", como por código numérico, cuya equivalencia será explicado en la etapa de transformación.

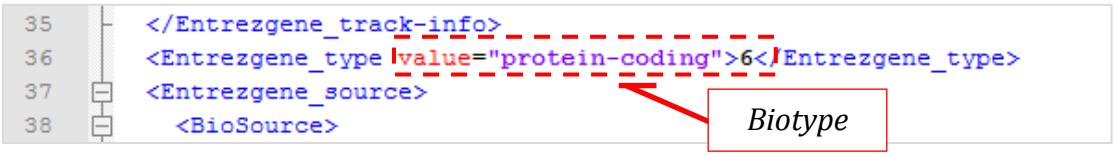

Figura 41. Detalle obtención atributo "*Biotype*"

Por otro lado, el símbolo del gen, el valor único asignado por el consorcio HGNC, así como su nombre oficial se obtienen procesando las etiquetas *Gene-ref_locus*, Gene-

ref_desc y *Objetc-id_str*, mientras que los genes sinónimos se encuentran dentro de *Gene-ref_syn_E*, todo ello dentro del elemento *Gene-ref* (Figura 42).

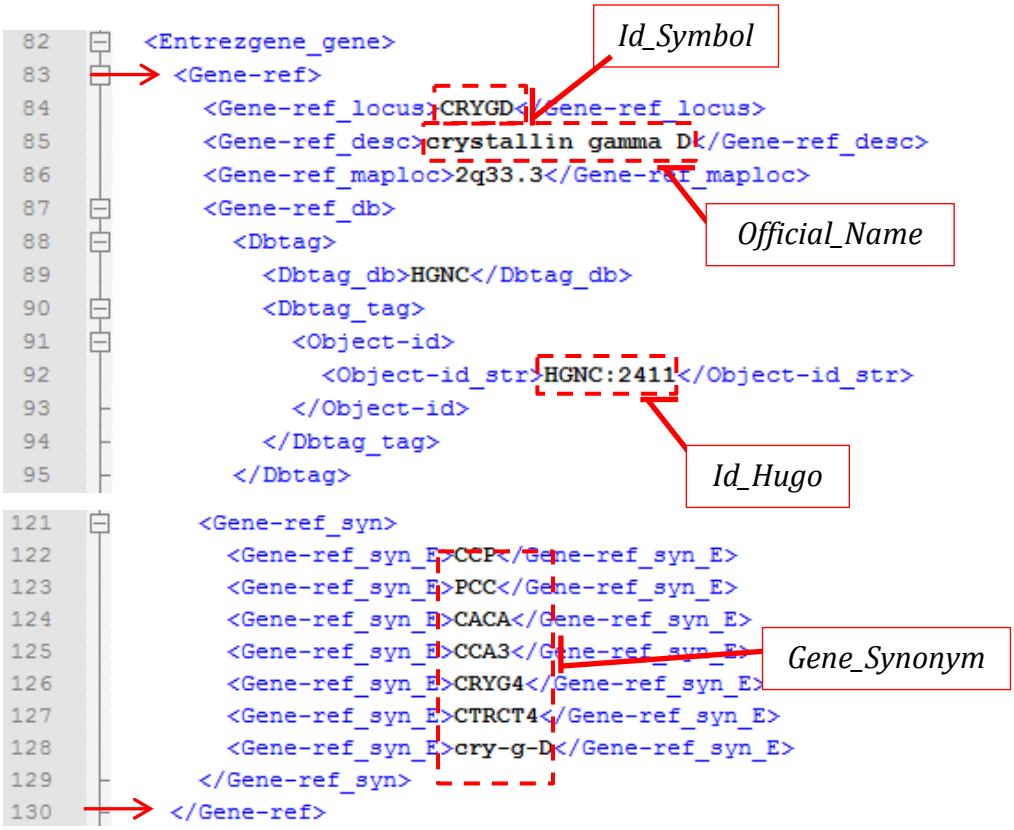

Figura 42.Detalle obtención "*Id_Symbol*", "*Official_Name*", "*Id_Hugo*", "*Gene_Synonym*"

A continuación, la etiqueta *Entrez_sumary* recoge un breve resumen de la funcionalidad del gen (Figura 43), que será depositado en el atributo "*Description*" de la clase "*Gene*".

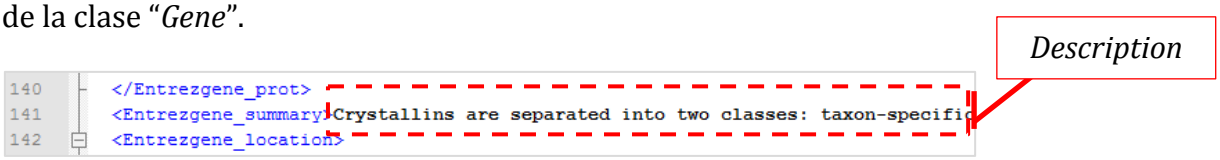

Figura 43. Detalle obtención atributo "*Description*"

Por otro lado, las posiciones del gen dentro de la secuencia se localizan bajo las etiquetas *Seq-interval_from* y *Seq-interval_to*, a continuación del identificador NG (Figura 44).

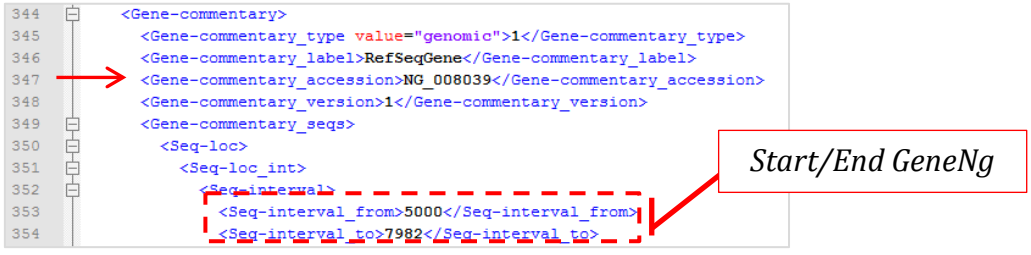

Figura 44. Detalle obtención atributo "*Start_GeneNG*" y "*End_GeneNG*"

Finalmente, el estado en el que se encuentra el gen es obtenido del elemento *Gene-commentay_label*, dentro del elemento padre *Gene-commentary*.

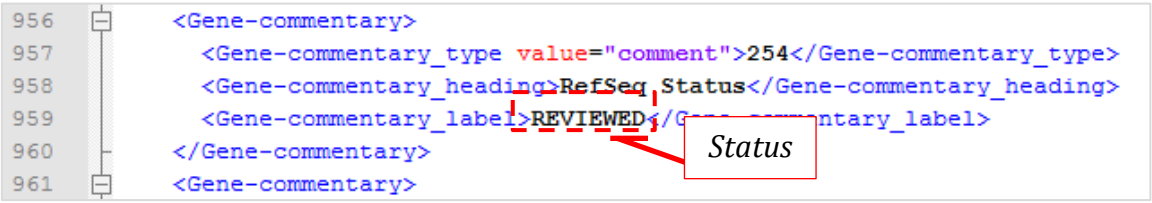

Figura 45. Detalle obtención atributo "*Status*"

Tabla 16. Extracción de datos de *Nucleotide*

Repositorio	**Nucleotide**
URL de consulta	https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nuccore&***id*=*ngIdentifier***&retmode=xml
Información obtenida	**Transcript:** StartCds, EndCds, Start_TranscriptNg, End_TranscriptNg **Exon**: Start_ExonNg, End_ExonNg **Sequence_Ng**: DNA_Secuence
Dato de estudio	**ngIdentifier = NG_008039.1** (obtenido del Clinvar)

El repositorio *Nucleotide* recapitula toda la información de las secuencias del genoma, genes y transcritos. Su búsqueda se realiza mediante los identificares que el NCBI asigna a los elementos cromosoma, *NM_identifier*, para la búsqueda de transcritos, *NG_identifier*, para la búsqueda génica y *NC_identifier* para la búsqueda cromosómica.

De la base de datos de *Nucleotide* se extrae toda la información de los transcritos dentro de la secuencia del gen, por lo que su consulta se realiza a través del *NG_identifier* obtenido del repositorio *ClinVar* (Tabla 16). En primer, se obtiene la posición inicial y final del proceso de traducción a proteína. Para ello hay que localizar el elemento que describe el *CDS*, su operación y las etiquetas de inicio y fin (Figura 46).

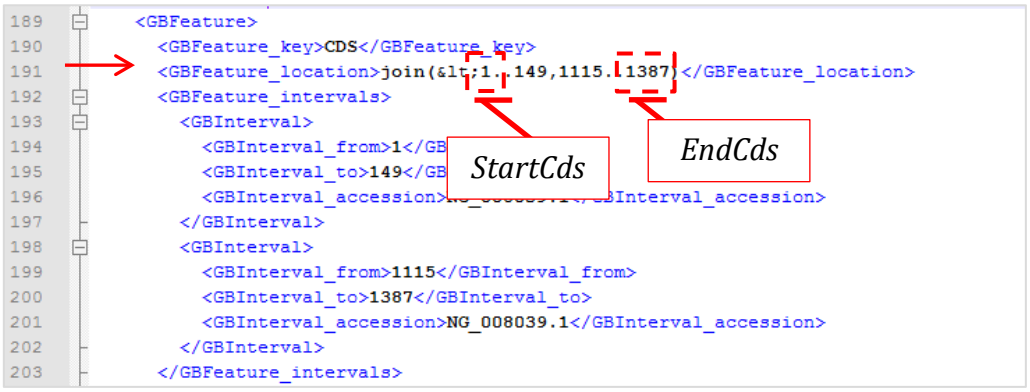

Figura 46. Detalle obtención atributos "*StartCDS*" y "*EndCDS*"

Por otro lado, las posiciones de inicio y fin del transcrito dentro de la secuencia del gen se encuentran a continuación de la cadena *mRNA,* localizada por la etiqueta *GBFeature_key*. Las posiciones se extraen de la etiqueta *GBFeature_location*, que indica el tipo de operación y las posiciones, a los extremos de la cadena (Figura 47)

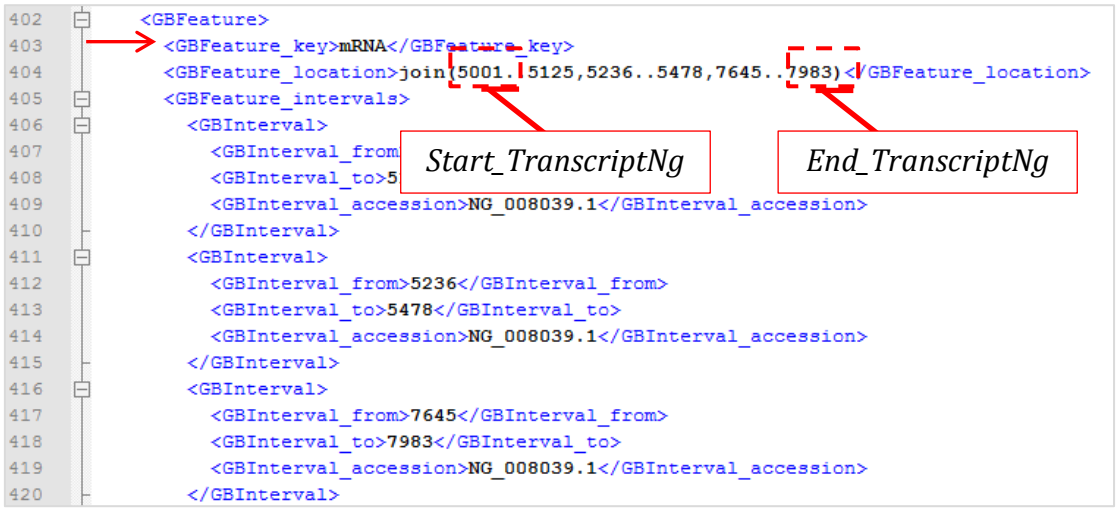

Figura 47. Detalle obtención atributos "*Start_TranscriptNg*" y "*End_TranscriptNg*"

A continuación, se extrae la informacion de los exones (Figura 48). La clase Exón está compuesta por el nombre y las posiciones dentro del gen. Para obtener estos datos hay que localizar la etiqueta que contiene la palabra *exon* y cuatro etiquetas *GBQualifier_name* después, crear el objeto exón con toda la información extraída, ya que Nuccleotide describe los exones de forma parcial en otras posiciones.

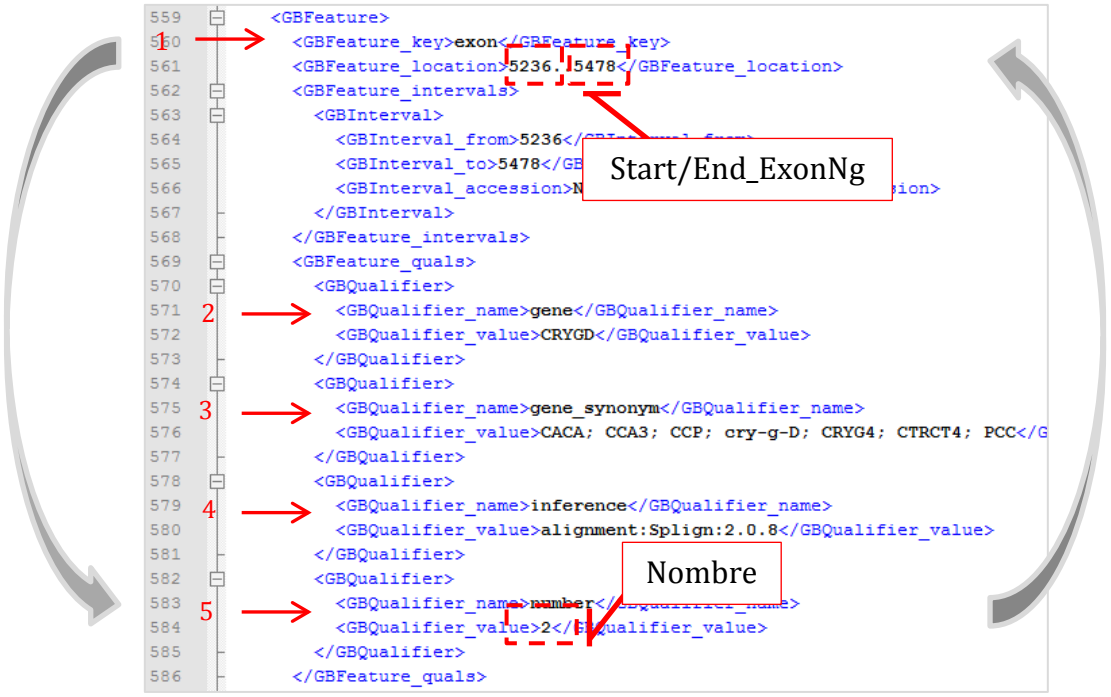

Figura 48. Detalle obtención atributos "*Start_ExonNg*", "*End_ExonNg*" y "*Nombre*"

Del repositorio *Nucleotide* se obtiene también la secuencia de ADN del gen, para ello hay que desplazarse hasta el final del documento, identificar el elemento *GBSeq_sequence* y leer su contenido (Figura 49).

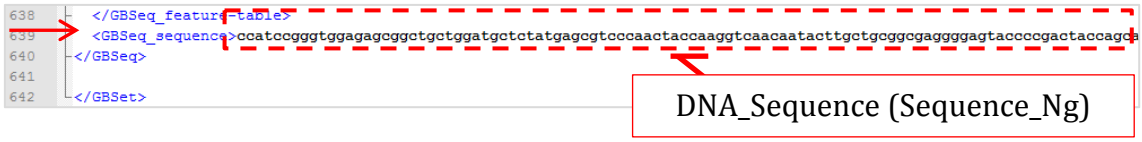

Figura 49. Detalle obtención atributo "*DNA_Sequence*"

De esta forma se obtiene toda la información necesaria para alimentar las clases *Transcript*, *Sequence_Ng* y *Exon* de la Vista Estructural, concluyendo así la consulta a la base de datos *Nucleotiode*.

Tabla 17. Datos extraídos de Protein

Repositorio	**Protein**
URL de consulta	https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=protein&***id*=*npIdentifier***&retmode=xml
Información obtenida	**Protein:** Name, Sequence
Dato de estudio	***npIdentifier*** = **NP_008822.2** (obtenido de Clinvar)

La clase "protein" está compuesta los atributos *Name*, *Sequence*, *NP_Identifier* y *Source*, los cuales hacen referencia al nombre de la proteína, la secuencia de aminoácidos que la forman, la fuente de datos de la que se extrae la información, y el identificador que proporciona el NCBI. Mientras que el identificar y la fuente de datos se conocen a priori, el resto de información debe de ser extraída del repositorio *Protein* (Tabla 17). Estos se obtienen de los elementos *GBSeq_definition* y *GBSeq_sequence* (Figura 50)

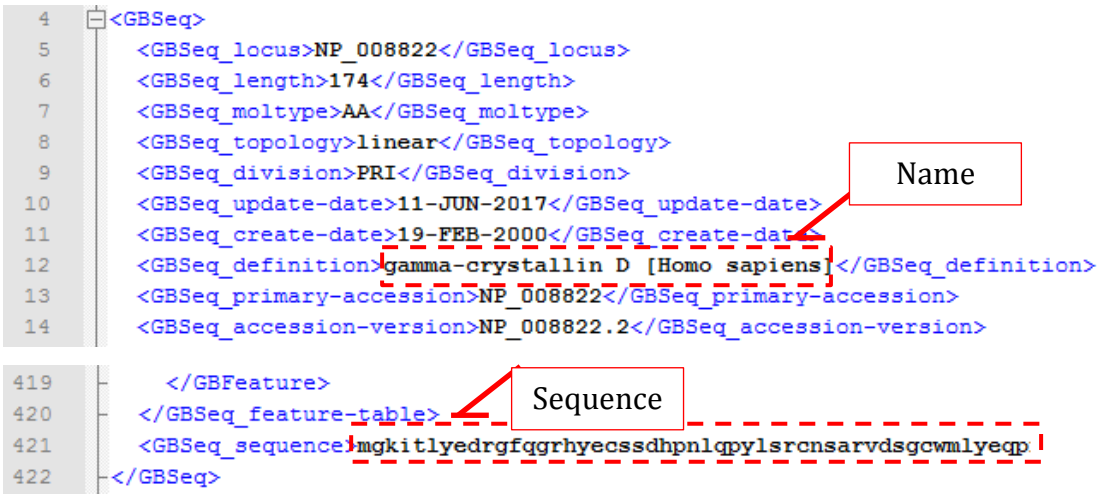

Figura 50. Detalle obtención atributos "*Name*" y "*Sequence*"

Tabla 18. Datos extraídos de *Pubmed*

Repositorio	**Pubmed**
URL de consulta	https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=pubmed&***id*=*pubmedId***&retmode=xml
Información obtenida	**Bib_ref:** Title, Abstract, Authors, Publication, Date
Dato de estudio	***pubmedId*** = **17724170** (obtenido de Clinvar)

Del repositorio *Pubmed* (Tabla 18) se obtiene toda la información descrita en la Vista Bibliográfica: *Title*, *Abstract*, *Authros*, *Publication* y *Date*; que corresponden al título del artículo, su resumen, autores, publicación y fecha de publicación. En primer lugar (Figura 51), el título del artículo se localiza dentro del elemento *ArticleTitle*.

```
<ArticleTitle>Conversion and compensatory evolution of the gamma-crystallin
    genes and identification of a cataractogenic mutation that reverses the sequence
    of the human CRYGD gene to an ancestral state.
</ArticleTitle>
```

Figura 51. Detalle obtención atributo "*Title*"

A continuación, del elemento *AbstractText* se obtiene el resumen del artículo (Figura 52).

```
<Abstract>
    <AbstractText>We identified a mutation in the CRYGD gene (P23S) of the gamma-crystallin gene
    cluster that is associated with a polymorphic congenital cataract that occurs with frequency
    of approximately 0.3% in a human population. To gain insight into the molecular mechanism of
    the pathogenesis of gamma-crystallin isoforms, we undertook an evolutionary analysis of the
    available mammalian and newly obtained primate sequences of the gamma-crystallin genes. The
    cataract-associated serine at site 23 corresponds to the ancestral state, since it was found
    in CRYGD of a lower primate and all the surveyed nonprimate mammals. Crystallin proteins include
    two structurally similar domains, and substitutions in mammalian CRYGD protein at site 23 of the
    first domain were always associated with substitutions in the structurally reciprocal sites
    109 and 136 of the second domain. These data suggest that the cataractogenic effect of serine
    at site 23 in the N-terminal domain of CRYGD may be compensated indirectly by amino acid changes
    in a distal domain. We also found that gene conversion was a factor in the evolution of the
    gamma-crystallin gene cluster throughout different mammalian clades. The high rate of gene
    conversion observed between the functional CRYGD gene and two primate gamma-crystallin
    pseudogenes (CRYGEP1 and CRYGFP1) coupled with a surprising finding of apparent negative
    selection in primate pseudogenes suggest a deleterious impact of recently derived pseudogenes
    involved in gene conversion in the gamma-crystallin gene cluster.</AbstractText>
</Abstract>
```

Figura 52. Detalle obtención atributo "*Abstract*"

Mientras que la información de los autores es almacenada por los elementos *Authors* y las etiquetas *LastName*, *ForeName* e *Initials*, creándose tantos autores como elementos *Authors* existan (Figura 53).

```
<AuthorList CompleteYN="Y">
    <Author ValidYN="Y">
        <LastName>Plotnikova</LastName>
        <ForeName>Olga V</ForeName>
        <Initials>OV</Initials>
        <AffiliationInfo>
            <Affiliation>Laboratory of M
        </AffiliationInfo>
    </Author>

</AuthorList>
```

Figura 53. Detalle obtención atributo "*Authors*"

La publicación del artículo se encuentra dentro de las etiquetas *ISOAbbreviation*, *Year*, *Month*, *Volume*, *Issue* y *MedlinePgn* (Figura 54). En la etapa de transformación se formará la cadena de texto con la estructura típica de publicación, añadiendo los separadores oportunos entre los elementos que la integran.

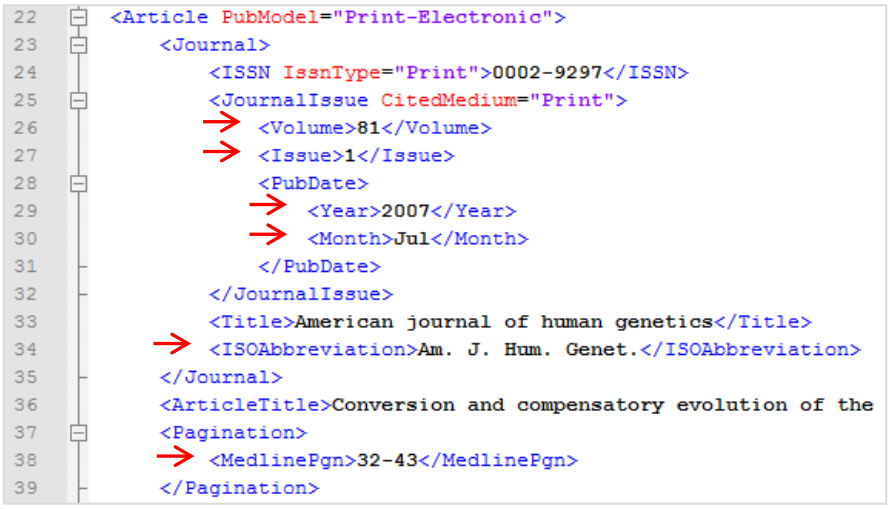

Figura 54. Obtención atributo "*Publication*"

Por último, la fecha del artículo se obtiene del elemento *ArticleDate* y las etiquetas *Year*, *Month*, *Day* (Figura 55).

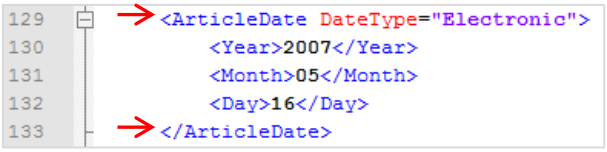

Figura 55. Detalle obtención atributo "*Date*"

Tabla 19. Información estática de página principal de ClinVar

Repositorio	**Página web de Clinvar**
URL de consulta	https://www.ncbi.nlm.nih.gov/clinvar/
Información obtenida	**Databank:** Nombre, Descripción

A diferencia del resto de información, la proporcionada por la Vista Fuente de datos es dada de manera automática por la aplicación sin consulta alguna al exterior, ya que se trata de información estática (Tabla 19), como el nombre de la fuente de datos de la que se obtuve toda la información de la variación (ClinVar), la descripción de la fuente de datos (*ClinVar aggregates information about genomic variation and its relationship to human health*" y su URL de enlace, (*https:// www.ncbi.nlm.nih.gov/clinvar/*).

5.2.4.3. TRANSFORMACIÓN DE DATOS

Una vez extraída la información de los distintos repositorios biológicos, la tapa de transformación lleva a cabo el procesamiento y validación de los datos con el fin de que la información cumpla con los requisitos y reglas del sistema de destino. Para ello, se realizan cambios de formato, codificación de valores, obtención de datos a partir de otros, así como generación de nuevos identificadores, combinación o división de atributos.

En la presente tesis la transformación de datos es vital, ya que la información proviene de distintos repositorios con una estructura distinta a la planteada por el modelo conceptual del genoma humano. En este sentido, a continuación, se describen las transformaciones llevadas a cabo para cumplir con los requisitos del modelo y permitir la etapa de carga en la base de datos.

VISTA VARIATION

Clase	Precise
Atributo	Specialization_Type
Origen datos	Clinvar

El esquema conceptual del genoma humano contempla cuatro tipos de variaciones posibles: *insertion*, *deletion*, *indel* e inversión. En cambio, *ClinVar* representa diez. Esta transformación se lleva a cabo mediante una correspondencia directa de datos según las especificaciones (Figura 56).

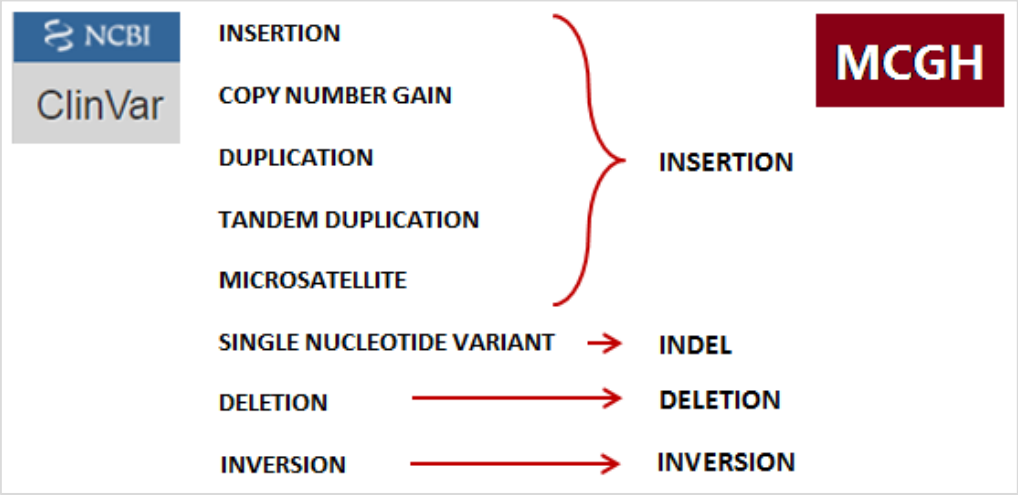

Figura 56. Correspondencia entre Clinvar y el MCGH para el tipo de variación

Clase	Precise
Atributos	Ins_Sequence, Ins_Repetition y Num_bases
Origen datos	Clinvar

Los atributos *Ins_Sequence*, *Ins_Repetition* y *Num_bases* representan la secuencia de nucleótidos insertados, el número de veces que se repite la secuencia insertada y el número de bases borradas de la secuencia. Los resultados de *Ins_Sequence*, *Ins_Repetition* y *Num_bases* se obtienen de la manipulación de *variationLength*, *referenceAllele* y *alternateAllele* en función de la variación.

En caso de que la variación sea un ***Inde*** (Figura 57)**,** Ins_Sequence tendrá el contenido de la etiqueta alternateAllele, "A", mientras Ins_Repetition y Num_bases se les asignará el valor "0" al no producirse una repetición ni un borrado de bases.

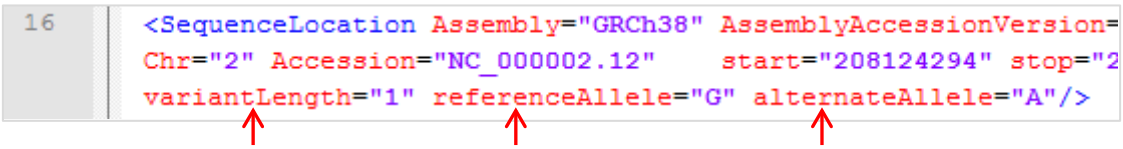

Figura 57. Ejemplo Indel con variación NM_006891.3(CRYGD):c.70C>A

En caso de que la variación sea una **Deleción** (Figura 58), el atributo Ins_Sequence será "-", Ins_repetition "0" y Num_bases el valor de variationLength, "4".

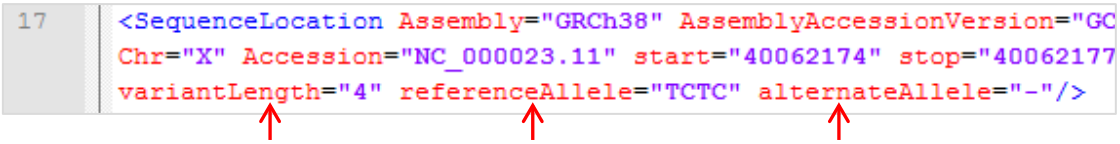

Figura 58. Ejemplo Deleción con variación NM_001123383.1(BCOR):c.4288_4291delGAGA

En caso de que la variación sea un **Inserción** (Figura 59), el atributó Ins_Sequence tendrá el contenido de la etiqueta referenceAllele (T), Ins_Repetition tendrá el resultado de la división del número de caracteres de la etiqueta alternateAllele entre el valor de variantLength, "2" y Num_bases contendrá la diferencia del número de caracteres de la etiqueta altenateAllele y el valor variationLength, "1".

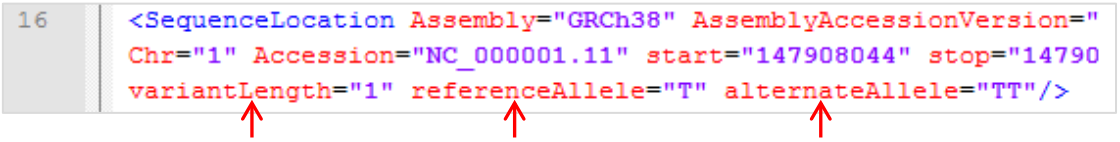

Figura 59. Ejemplo Inserción con variación NM_005267.4(GJA8):c.89dupT

Clase	Genome
Atributo	HG_Identifier
Origen datos	Clinvar

La obtención del identificador *HG_Identifier* (Figura 60) se obtiene mediante correspondencia directa con la versión de identificador del genoma humano CRGh_Identifier.

Figura 60. Correspondencia entre "CRGh_Identifier" y "HG_Identifier"

Clase	Chromosme
Atributo	Nombre
Origen datos	Clinvar

El nombre del cromosoma (Figura 61) se obtiene procesando el atributo *NC_Identifier*, eliminando el prefijo *NC_00000*, la versión del cromosoma y añadiendo la cadena "*Chr*" al resultado del procesado anterior.

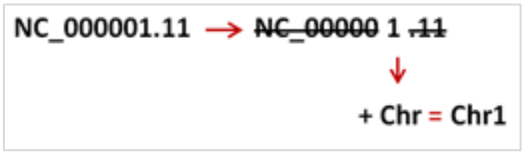

Figura 61. Obtención del atributo "nombre" del cromosoma

Clase	Chr_Element
Atributo	Strand
Origen datos	Clinvar

ClinVar representa la hebra mediante los signos "-" y "+", mientras que el MCGH humano emplea los caracteres "M", de minus para "-", y "P", de plus, para "+". (Figura 62)

Figura 62. Correspondencia entre Clinvar y el MCGH del atributo "hebra"

Clases	Transcript y Precise_SeqNg
Atributos	NM_Identifier y NG_Identifier

Origen datos	Clinvar

El identificador que el NCBI asigna a los transcritos y genes se obtiene procesando el atributo "*Others_identifiers*" (Figura 63) de la clase *Variation*, eliminando el cambio producido por la variación en cada contexto.

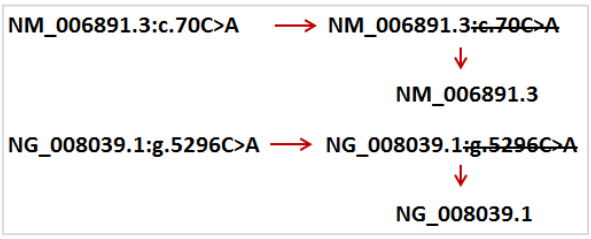

Figura 63. Obtención de los atributos "*NM_Identifier*" y "*NG_Identifier*"

Clase	Exon
Atributos	Nombre, Start_ExonNg y End_ExonNg
Origen datos	Nucleotide

En la etapa de extracción de datos, a partir de una secuencia de referencia de un gen se obtiene todos los exones existentes dentro de la misma. Sin embargo, el exón a documentar es aquel en cuya posición de inicio y fin se produce el cambio. Para ello, la etapa de transformación constará en la lectura de todos los exones extraídos y en la selección de aquel cuyo inicio y fin contenga la posición exacta en la que se produce la variación dentro del gen.

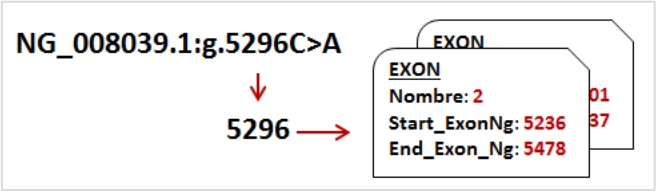

Figura 64. Obtención de los atributos *"Nombre"*, *"Start_ExonNg"* y *"End_ExonNg"*

Clase	Exon_Transcript
Atributos	Nombre
Origen datos	Exon

El atributo "nombre" de la clase *Exon_Transcript* se obtiene directamente del atributo homónimo de la clase *Exon*, pero eliminando la letra que pueda acompañar al número, si se da este caso.

Clase	Protein
Atributos	NP_Identifier
Origen datos	Clinvar

De forma análoga a la obtención de los identifidores *NM_Identifier* y *NG_Identifier*, el identificador que el NCBI asigna a la proteína, se obtiene procesando el contenido del atributo "*Others_identifiers*" de la clase *Variation*, eliminando el cambio en la proteína (Figura 65).

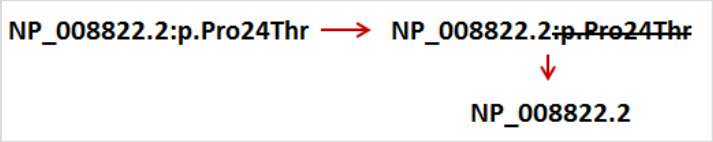

Figura 65- Obtención del atributo "*NP_Identifier*"

VISTA FUENTE BIBLIOGRÁFICA

Clases	Bib_Ref, Bibliography_Db
Atributos	Publication, Authors, URL, Date
Origen datos	Pubmed

Como se observó en la etapa de extracción, todos los datos que forman el atributo publicación están segmentados de forma arbitraria, resultando necesario su procesamiento. Dicho procesamiento consta en la definición del orden de los datos y asignación de separadores (Figura 66).

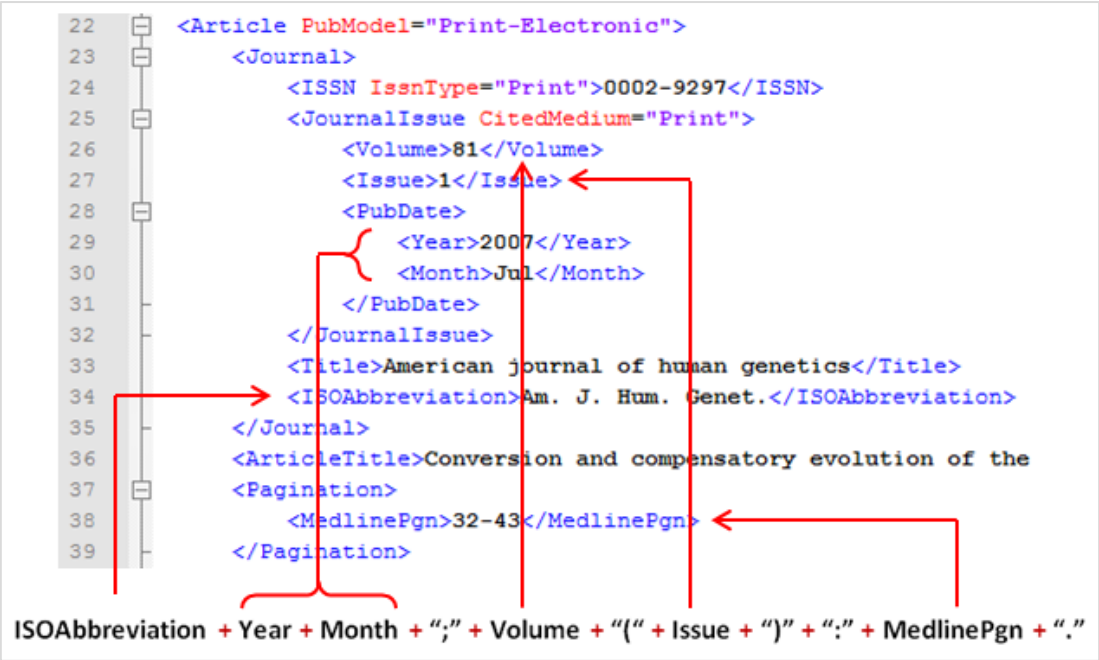

Figura 66. Detalle obtención atributo "*Publicacion*"

En cuanto al atributo "*Authors*", la etapa de extracción ofrecía esta información segmentada mediante las etiquetas nombre, apellido e las iniciales. En consonancia con la estructura clásica de referencias, los datos deben organizarse de este modo (Figura 67).

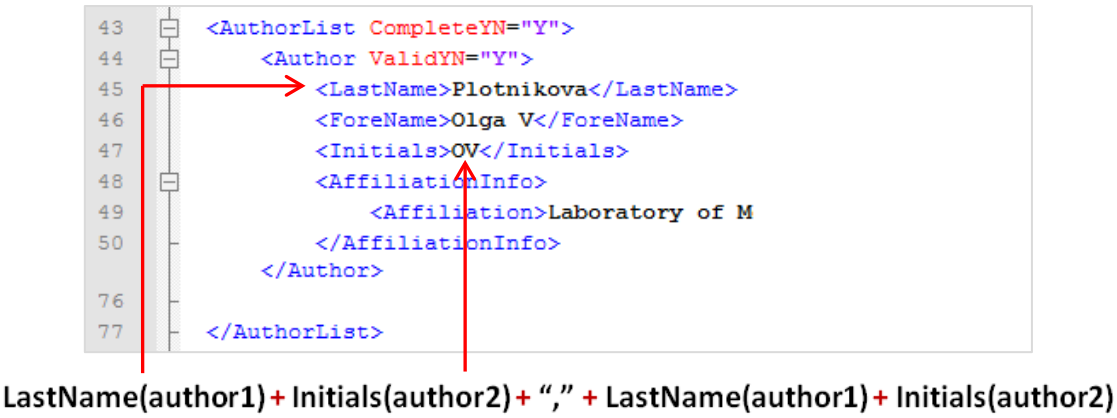

Figura 67. Detalle obtención atributo "*Authors*"

Por otro lado, la obtención de la URL de la fuente de datos de la que web de la que se extrae las publicaciones, está formada por el cuerpo de la URL base de *Pudmed*, más el identificar único de la publicación (Figura 68).

Figura 68. Detalle obtención atributo "*URL*"

La transformación de datos de la vista fuente bibliográfica concluye adaptando la fecha de publicación de los artículos al formato *Date* que la etapa de carga espera recibir. Para ello, hay que leer el contenido de las etiquetas *Year*, *Month* y *Day* del XML y formar el siguiente tipo de dato.

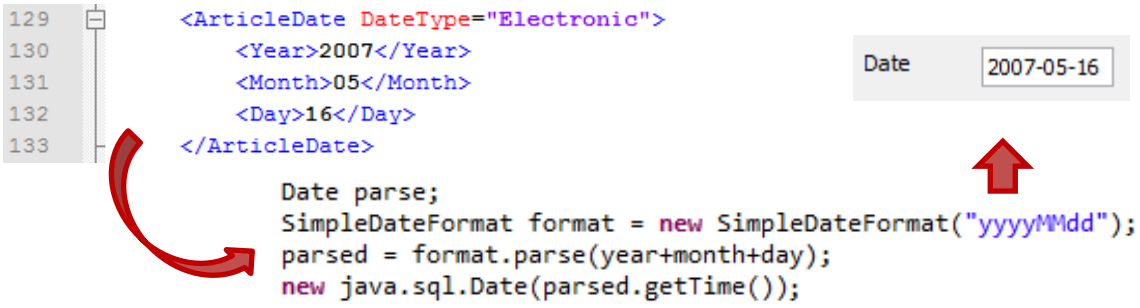

Figura 69. Detalle obtención atributo "*Date*"

5.2.4.4. CARGA DE DATOS

La carga de datos es la última etapa del proceso ETL y de la etapa Load de SILE. Cabe destacar que la etapa de carga introduce todos los datos extraídos y

transformados en el sistema de destino, es decir, la base de datos del genoma humano.

La información para introducir es única y no permite la existencia de duplicidades, motivo por el cual antes de realizar cualquier carga se debe comprobar la versión del genoma y la existencia de nuevos estudios respecto a la variación. En caso de que la información existente en el sistema sea la misma que la extraída de los repositorios, la carga no se llevará a cabo.

A continuación se describe el proceso de carga con el prototipo desarrollado (Figura 52), y el detalle de la información almacenada en la base de datos (HGDB).

PROTOTIPO SOFTWARE ETL

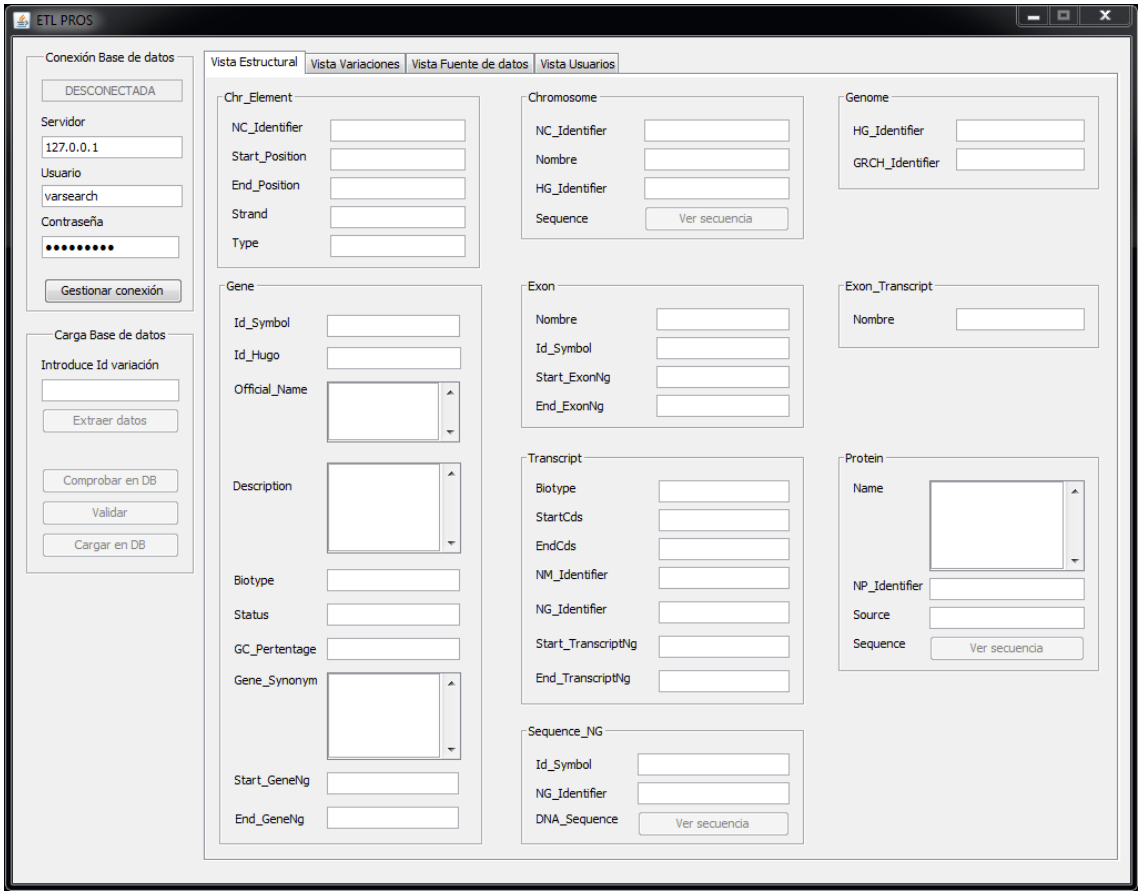

Figura 70. Ventana principal del prototipo software ETL

La aplicación se inicia con el establecimiento de la conexión a la base de datos (Figura 70). Para ello, el software dispone de una pequeña gestión de conexiones con la que se comprueba la comunicación directa con el servidor (Figura 71). Una vez establecida y validada, la aplicación informa del estado de la conexión y

permite el proceso de *extracción* de carga de información desde los repositorios biológicos *Clinvar*, *dbSNP*, *Gene*, *Nucleotide* y *Pudmeb,* y la gestión de usuarios desde la Vista Usuarios, activando el botón de Extracción de datos e informando de los usuarios que hay en la base de datos actualmente.

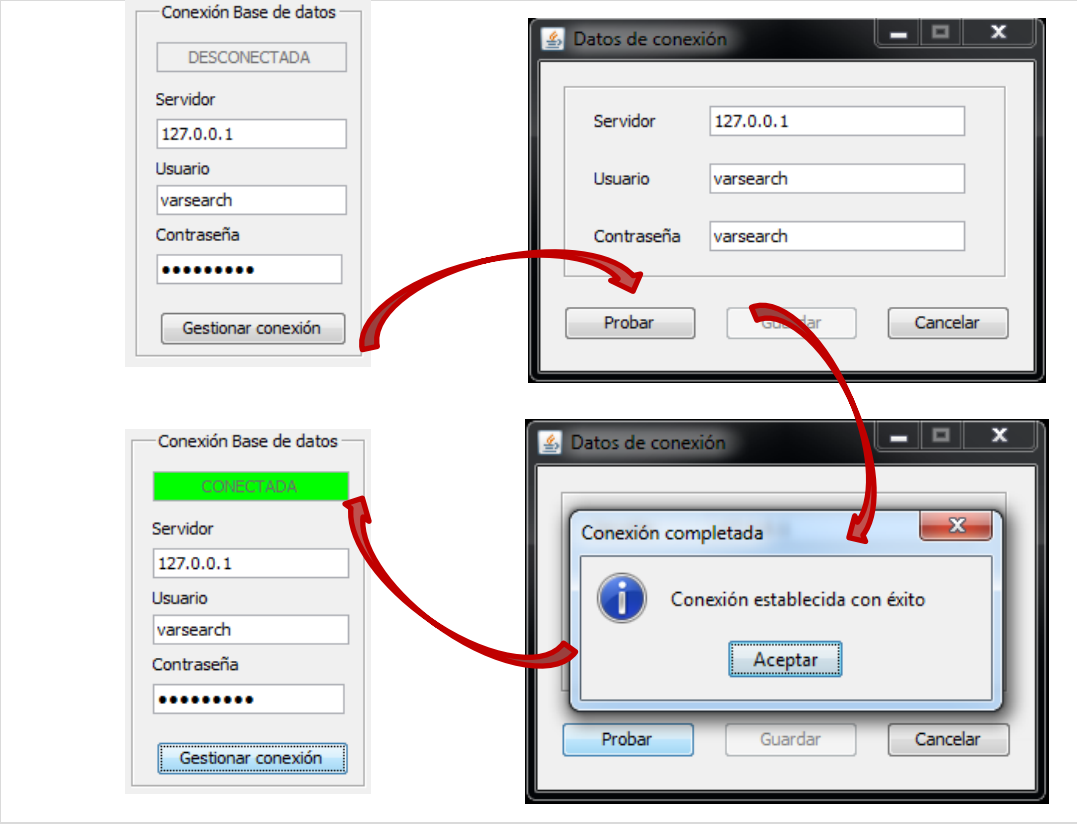

Figura 71. Detalle gestión de conexión

En caso de que la conexión no pueda establecerse, la aplicación informa de ello (Figura 72) e indica al usuario que debe comprobar sus credenciales y la dirección del servidor.

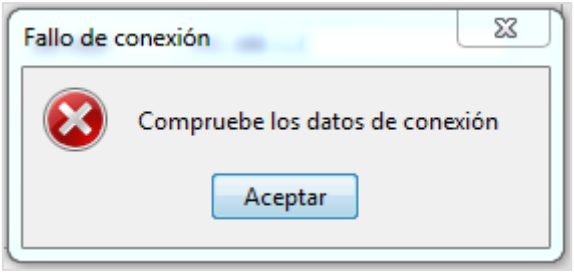

Figura 72. Detalle ventana Fallo de conexión

La Vista de Usuarios tiene como objetivo dar cabida al personal de validación de variaciones, cuyo conocimiento médico-científico certifican que las variaciones de estudio tienen una repercusión biológica suficiente como para ser introducidos en

la base de datos. Esta vista dispone (Figura 73) de una pequeña gestión de usuarios en la que se permite la edición, eliminación y creación de nuevo personal (Figura 74).

A diferencia del resto de vistas, la información proviene directamente de la base de datos, de ahí la necesidad de establecer conexión con ella antes de extraer la información de los repositorios científicos.

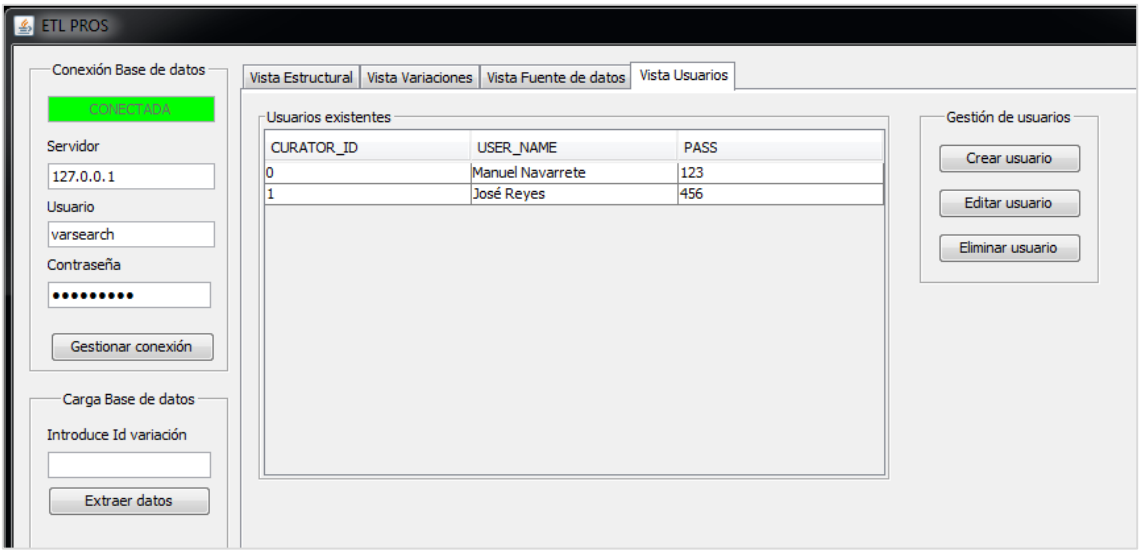

Figura 73. Detalle Vista usuarios de la aplicación ETL

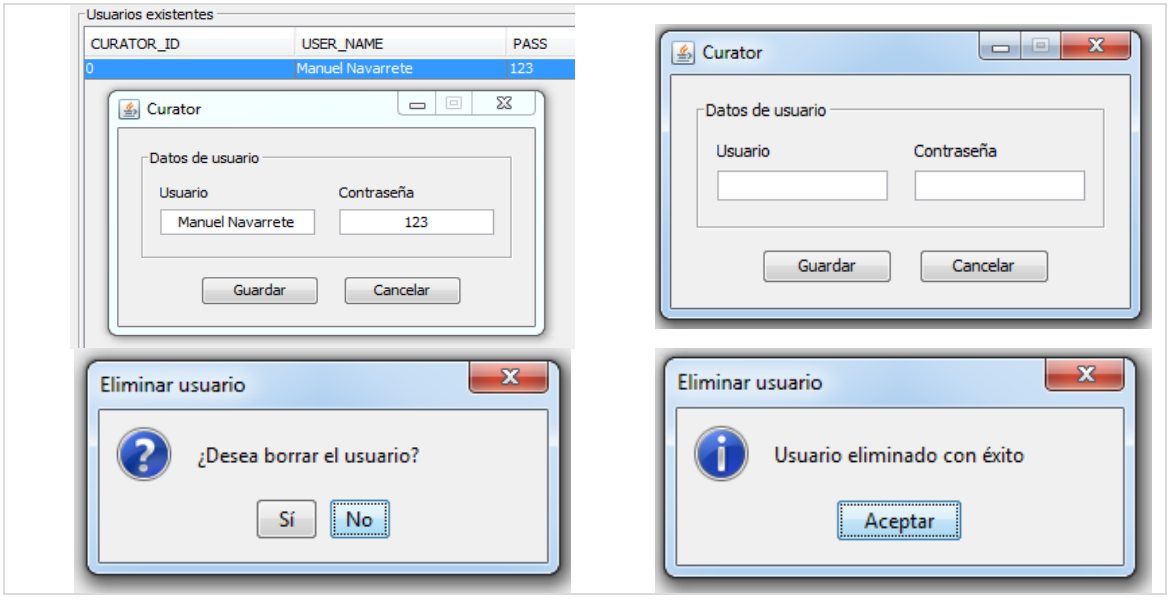

Figura 74. Detalle ventanas de edición, creación y eliminación de usuarios

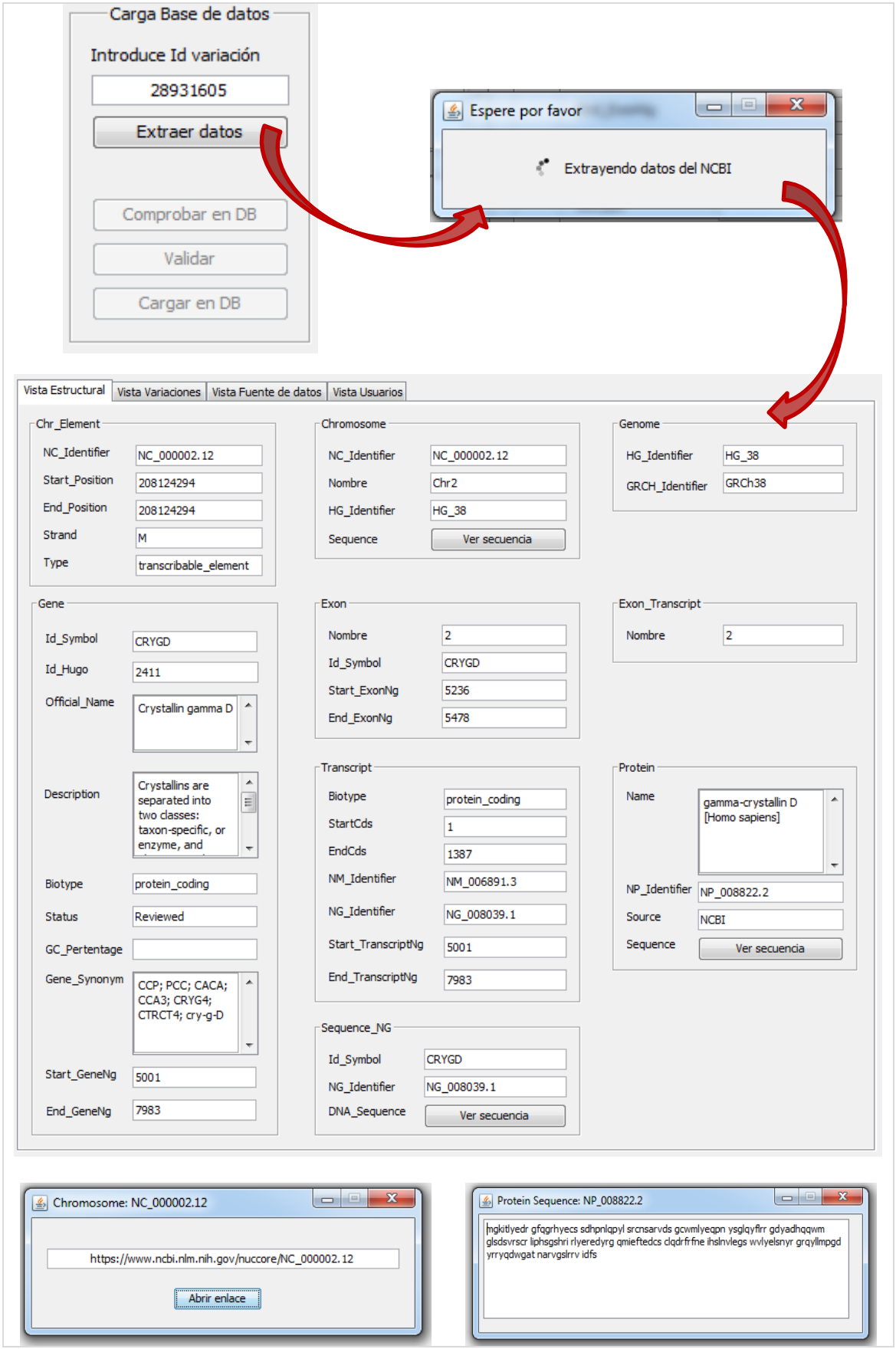

Figura 75. Detalle Extracción información y Vista estructural

La Vista Variaciones (Figura 76) muestra toda la información de la variación modelizada por el esquema conceptual del genoma humano. Dado que pueden producirse dos cambios de nucleótidos en la misma variación, toda la información subyacente se diversifica mediante pestañas.

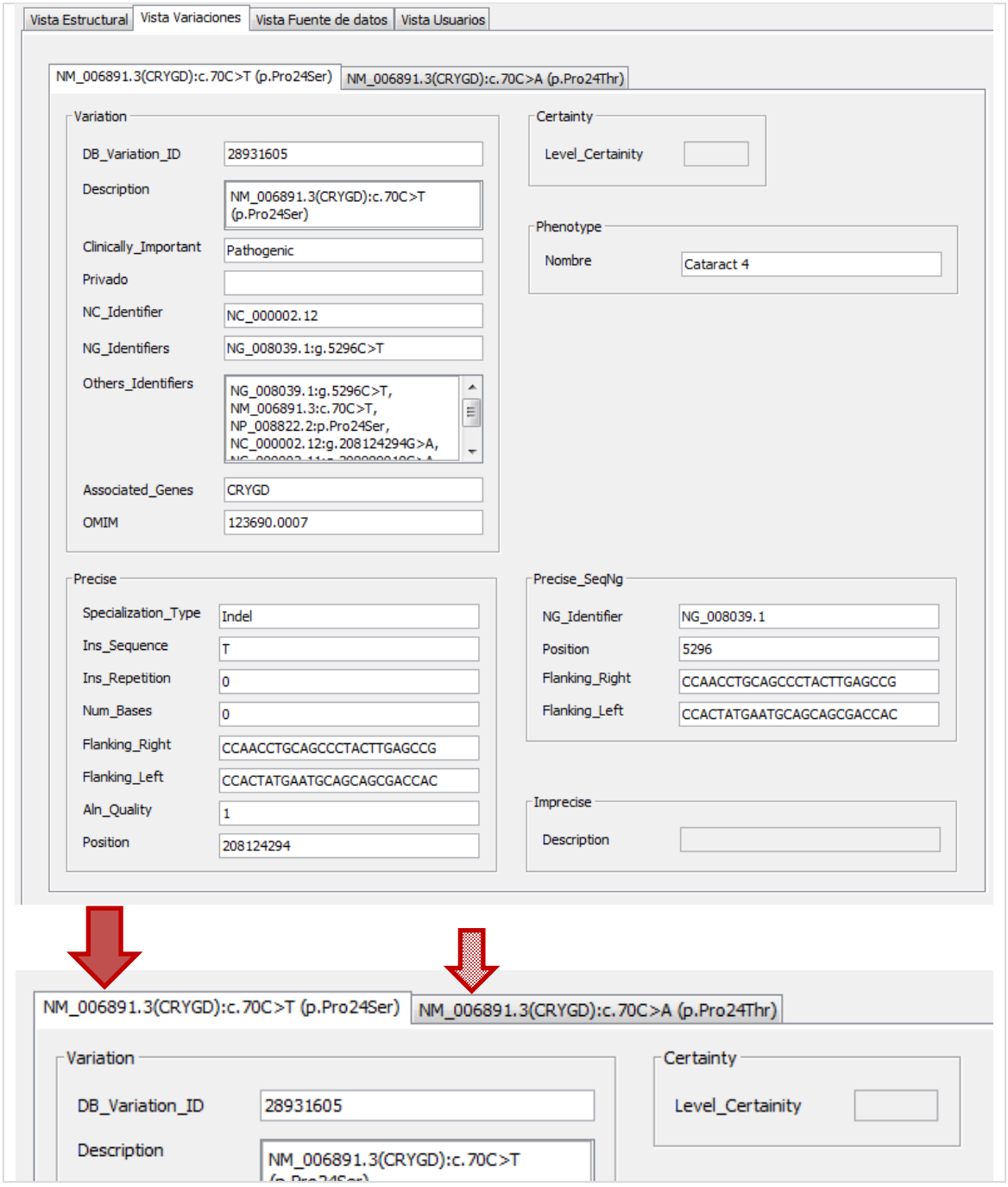

Figura 76. Detalle Vista variaciones y pestañas de selección de variación

La vista de Fuente de datos (Figura 77) muestra toda la información de las citas bibliográficas que documentan la variación de estudio, y las fuentes de datos de la

que se extrajo toda la información. De forma similar a la Vista variaciones, la existencia de múltiples citas bibliográficas se organiza mediante pestañas.

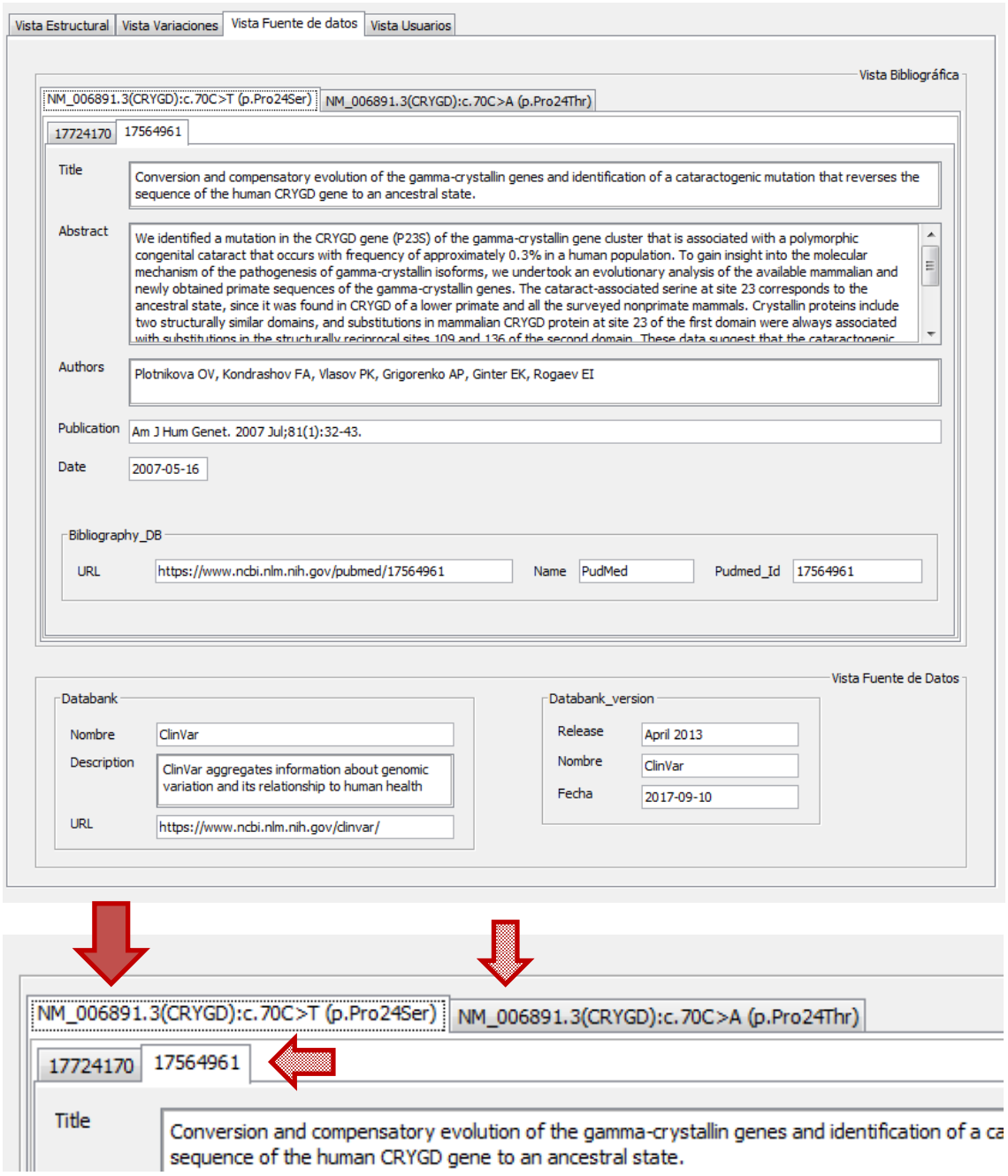

Figura 77. Detalle Vista Fuente de datos y pestañas de selección de bibliografía

Si la información extraída de los repositorios es útil y se desea introducir en la base de datos para su posterior explotación, previo a la inserción, la aplicación realiza una comprobación de duplicidades, y posteriormente, independientemente de si lleva a cabo la validación, induce toda la información biológica obtenida en la base de datos HGDB (Figura 78).

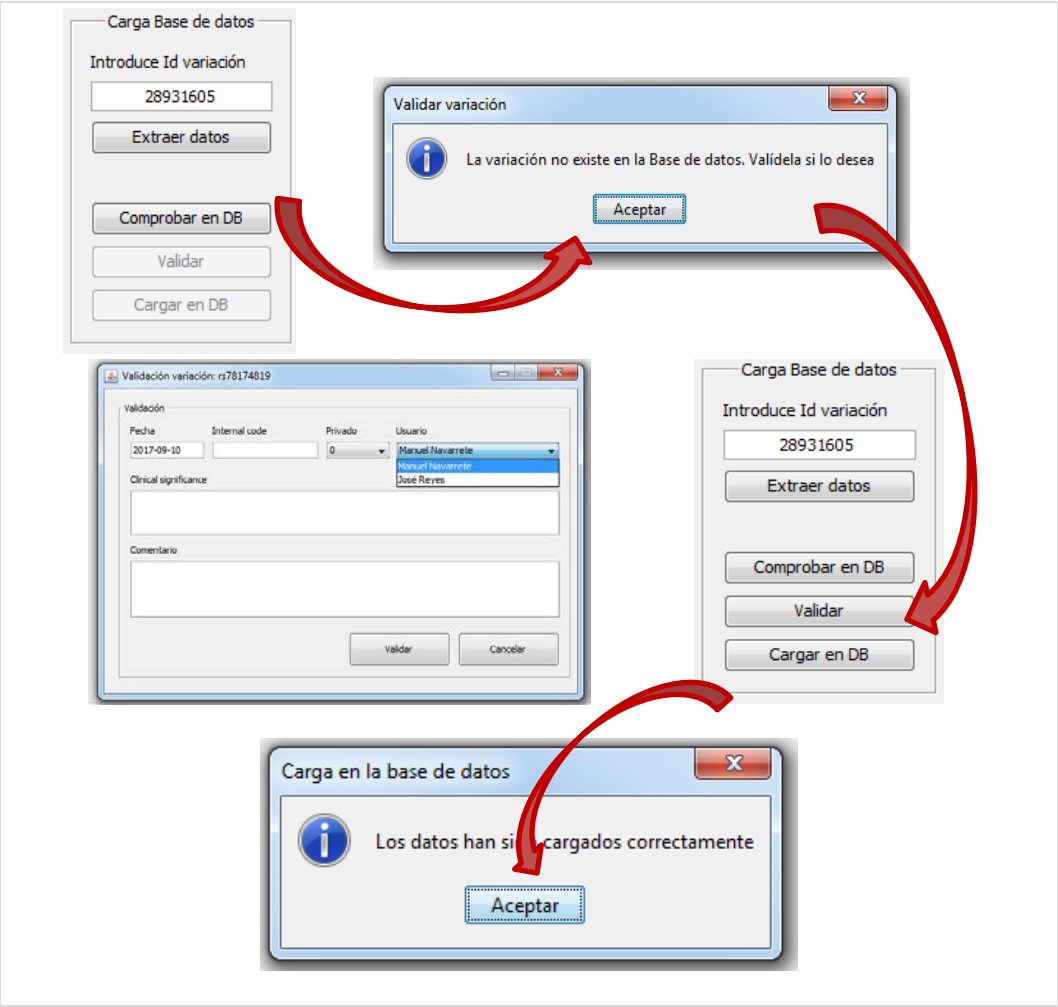

Figura 78. Detalle proceso de validación y carga

A continuación, se muestra unas capturas de la información extraída con la aplicación y cargada en la base de datos HGDB (Figuras 79-81)

VARIATION_ID	CHR_GENE_ID	DB_VERSION_ID	DESCRIPTION	DB_VARIATION_ID	CL
32	32	0	NM_006891.3(CRYGD):c.70C>T (p.Pro24Ser)	28931605	Pa
33	32	0	NM_006891.3(CRYGD):c.70C>A (p.Pro24Thr)	28931605	Pa

Figura 79. Detalle de la información cargada en la tabla "*Variation*"

VARIATION_ID	SPECIALIZATION_TYPE	INS_SEQUENCE	INS_REPETITION	NUM_BASES	FLANKING_RIGHT
32	Indel	T	0	0	CCAACCTGCAGCCCTA
33	Indel	A	0	0	CCAACCTGCAGCCCTA

Figura 80. Detalle de la información cargada en tabla "*Precise*"

ID_SYMBOL	ID_HUGO	OFFICIAL_NAME	DESCRIPTION	BIOTYPE	STATUS	GC_PERCENTAGE	GENE_SYNONYM	START_
CRYGD	2411	Crystallin gamma D	Crystallins are separated into two classes:	protein_coding	Reviewed	0	CCP; PCC; CACA; CCA3; CRYG4; CTRCT4; cry-g-D	5001

Figura 81. Detalla de la información cargada en la tabla "*Gene*"

5.2.5. EXPLOITATION

La última etapa de la metodología SILE es la explotación de datos mediante el framework genómico "*VarSearch*", cuya función es la obtención de conocimiento mediante el procesamiento de la información contenida (Figura 82) en las muestras de los pacientes (facilitadas mediante archivos VCF), los cuales representan y almacenan variaciones; y la existente en la base de datos del genoma humano.

Figura 82. Funcionamiento *VarSearch*

El framework genómico "*VarSearch*", basado en la web, se inicia con la selección y procesamiento del archivo VCF de estudio, el cual contendrá la información de las variaciones organizadas en bloques, representando la posición en la que se produce la variación, el cromosoma, el identificador único de la variación, así como los alelos de referencia y otros datos biológicos.

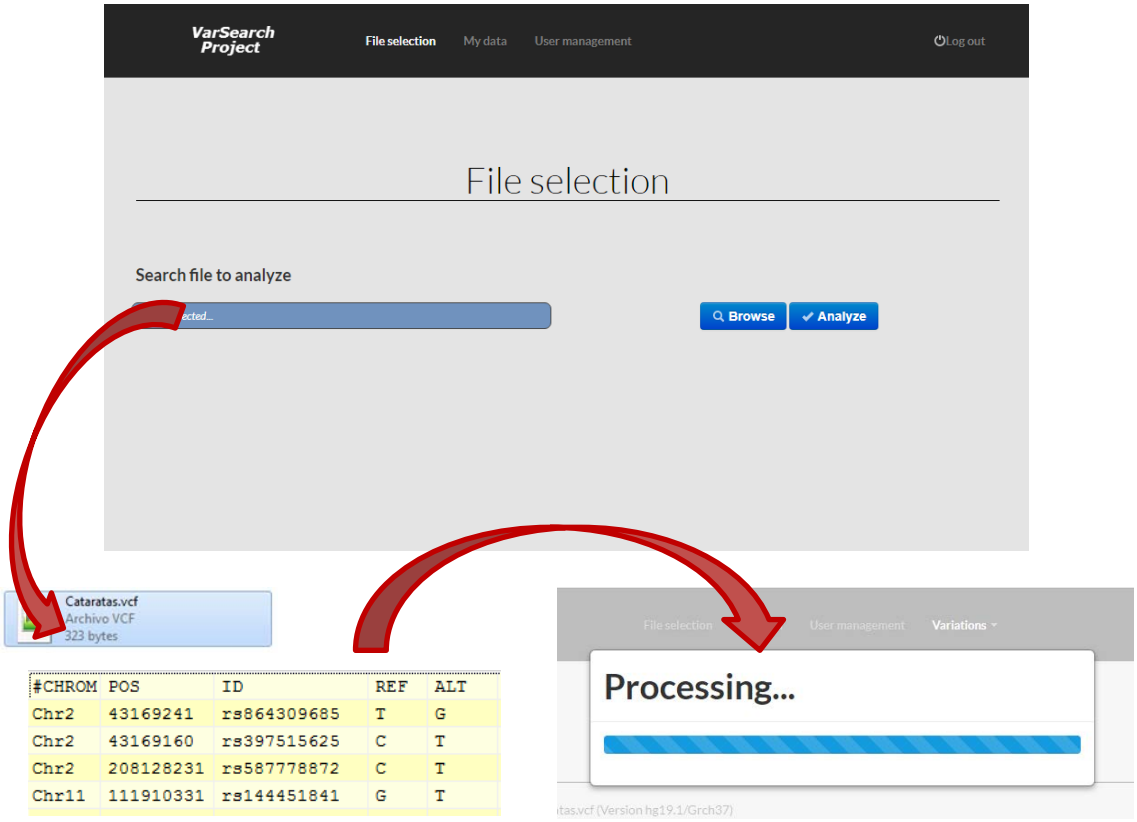

Figura 83. Detalle del proceso de carga y procesamiento

Una vez finalizado el análisis del fichero VCF, la aplicación muestra dos tipos de resultados, “variaciones encontradas” (Figura 84), las cuales indican la relación existente con las cataratas; y “variaciones no encontradas” (Figura 85), es decir, aquellas que se encuentran en el fichero VCF de estudio (muestra), pero cuya información no está respaldada/almacenada por el Sistema de Información Genómico, y por ende, no se encuentran relacionadas con las cataratas.

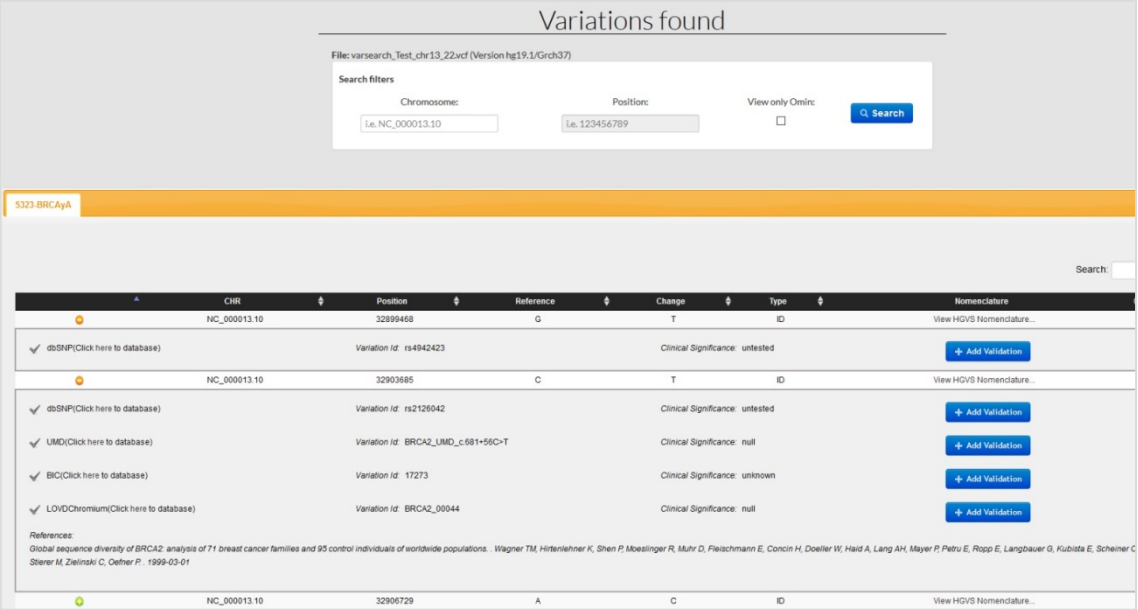

Figura 84. Detalle variaciones encontradas por *VarSearch*

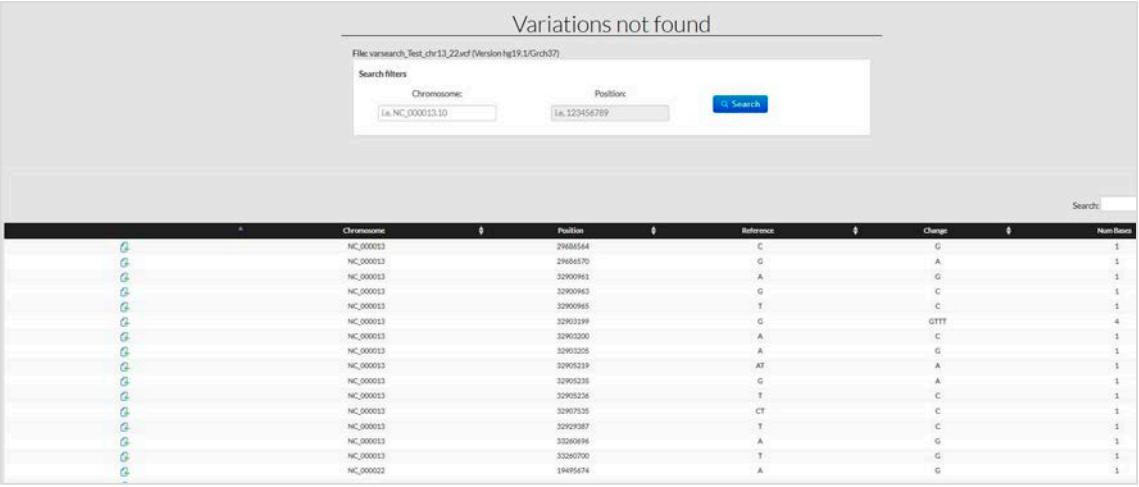

Figura 85. Detalle variaciones no encontradas por *VarSearch*

Todos estos datos tienen utilidad desde el punto de vista del diagnóstico genético temprano, que se pueden detectar utilizando marcadores genéticos que pueden ser ofrecidos como producto mediante los conocidos como *Tests Genéticos Directos al Consumidor (TGDC).* Un ejemplo de estos tests son los que se pueden encontrar en la plataforma *GenesLove.Me* (71) (Figura 86), que ofrece tests para condiciones con base genética como la alopecia androgénica, la intolerancia a la lactosa, la sensibilidad al alcohol o la enfermedad de Dupuytren.

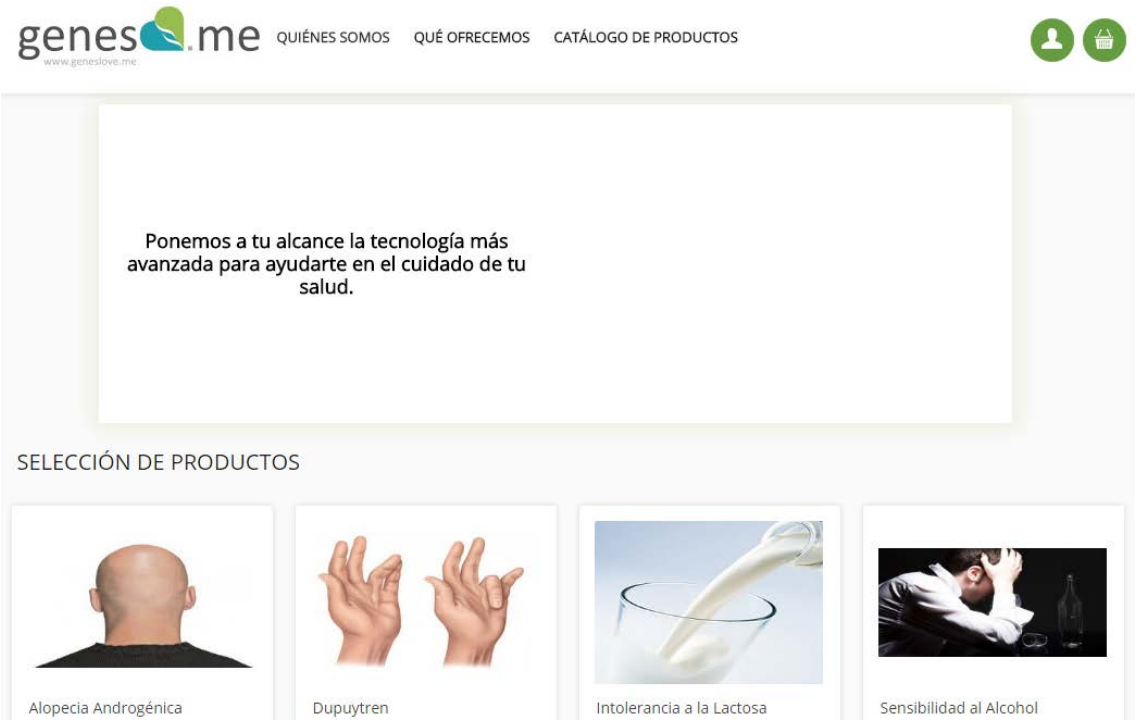

Figura 86. Página inicial de la plataforma GensLove.me

CONCLUSIONES Y TRABAJO FUTURO

La presente tesis tenía como objetivo el diseño e implementación de un sistema de información genómico para el diagnóstico preventivo de la catarata congénita, empleando para ello la metodología SILE. La aplicación de esta metodología, de carácter manual, se ha automatizado parcialmente mediante el desarrollo de un software de Extracción, Transformación y Carga, cuyo prototipo ha funcionado correctamente para los datos de estudio.

El desarrollo de esta aplicación se ha basado en el Modelo Conceptual del Genoma Humano (MCGH), el cual representa y unifica todo el conocimiento existente de una forma ágil y consistente, con el objetivo de potenciar la Medicina Personalizada o de precisión. Su estructura, basada en vistas, ha permitido dar cabida a las variaciones genéticas que pueden contener un gen y su expresión fenotípica. En este sentido, se ha definido y modelado una serie de clases que permiten la gestión informática de la información, tanto para su obtención como para su carga en la base de datos del genoma humano (HGDB), la cual es una proyección del MCGH descrito.

Previo a este desarrollo, se ha estudiado toda la estructura del genoma, sus procesos, elementos biológicos y las unidades de información existentes en cada etapa, todo ello con el fin de conocer el dominio de trabajo y la obtención de los datos de estudio, que son las variaciones relacionadas con las cataratas congénitas.

Para la obtención de la información biológica que documentan las variaciones, se han estudiado los distintos repositorios científicos del NCBI, como *Gene*, *ClinVar*, *dbSNP*, *Nucleotide*, *Protein* y *Pubmed*; los métodos de acceso, la información ofrecida y el procesamiento más idóneo para su posterior extracción, transformación y carga. Además, se ha analizado la relación entre repositorios y se han definido métodos de correspondencia entre ellos, ya que el principal problema al que se hace frente en este ámbito es la falta de estandarización y orden de la información, dando lugar a problemas de inconsistencia y heterogeneidad, situación a la cual se ha querido hacer frente en este Trabajo Final de Máster.

El prototipo de aplicación ETL desarrollado funciona como una herramienta de obtención de información biológica desde múltiples repositorios a partir de un

único parámetro de entrada, mostrando visualmente y de forma estructura la información para su posterior carga y explotación, en esta primera fase se han utilizado las fuentes de datos del NCBI, sin embargo, este prototipo presenta ciertas limitaciones ligadas a la obtención de información, ya que el programa no obtiene en algunos casos toda la necesaria porque los repositorios biológicos no disponen de una descripción completa. Adicionalmente, el programa solo accede a datos del NCBI, por lo que habría una línea de mejora en cuanto a la exploración de nuevas fuentes de datos y estandarizarla.

Por otro lado, el trabajo fututo más destacable sería dotar a la aplicación de mecanismos de actualización automática de la información almacenada. Esto permitiría la comparación directa entre la información existente y la obtenida de los repositorios, los cuales se actualizan continuamente con nuevos datos biológicos, así como la carga selectiva de información dependiendo de su versión, veracidad y utilidad biomédica. También, la aplicación da cabida a una posible conversión de herramienta de escritorio a herramienta web, permitiendo así su utilización desde cualquier lugar, dispositivo y plataforma.

REFERENCIAS BIBLIOGRÁFICAS

1. **Ding, X.-X., Zhu, Q.-G., Zhang, S.-M., Guan, L., Li, T., Zhang, L., … Zhang, H.-Q.** *(2017). Precision medicine for hepatocellular carcinoma: driver mutations and targeted therapy. Oncotarget, 8(33), 55715–55730. http://doi.org/10.18632/oncotarget.18382.*

2. **Ma, A. S., Grigg, J. R., Ho, G., Prokudin, I., Farnsworth, E., Holman, K., … Jamieson, R. V.** *(2016). Sporadic and Familial Congenital Cataracts: Mutational Spectrum and New Diagnoses Using Next-Generation Sequencing. Human Mutation, 37(4), 371–384. https://doi.org/10.1002/humu.22948.*

3. **Joly, Y., Feze, I., & Simard, J.** *(2013). Genetic discrimination and life insurance: a systematic review of the evidence. BMC Medicine, 11(1), 25. https://doi.org/10.1186/1741-7015-11-25.*

4. **Joly, Y., Burton, H., Knoppers, B. M., Feze, I. N., Dent, T., Pashayan, N., … Van Hoyweghen, I.** *(2014). Life insurance: genomic stratification and risk classification. European Journal of Human Genetics, 22(5), 575–579. https://doi.org/10.1038/ejhg.2013.228.* págs. 575–579.

5. **Staden, R.** *(1979). A strategy of DNA sequencing employing computer programs. Nucleic Acids Research, 6(7), 2601–2610. https://doi.org/10.1093/nar/6.7.2601.*

6. **Solomon, B. D.** *(2014). Obstacles and opportunities for the future of genomic medicine. Molecular Genetics & Genomic Medicine, 2(3), 205–209. http://doi.org/10.1002/mgg3.78.*

7. **Boyd, Kierstan.** (2016) What Are Cataracts? American Academy of Ophthalmology. [En línea] https://www.aao.org/eye-health/diseases/what-are-cataracts.

8. **The National Eye Institute (NEI).** (2015) Facts About Cataract. [En línea] https://nei.nih.gov/health/cataract/cataract_facts.

9. **Arrazola-Vázquez J.C., Morfín-Salido I.L., Moya-Romero J.O.** *(2008) Cirugía de extracción extracapsular de catarata con incisión pequeña versus convencional realizadas por residentes.*

10. **Reyes R., J. F.** *(2013). Integración de Haplotipos al Modelo Conceptual del Genoma Humano utilizando la Metodología SILE.*

11. **Unknow.** (2015). The Variant Call Format (VCF) Version 4 . 2 Specification. Online Resource, 1–28. https://doi.org/10.1016/j.ymeth.2012.07.021. [En línea]

12. **EMBL-EBI, Ensembl.** Ensemble. [En línea] https://www.ensembl.org/index.html.

13. **University of California, Santa Cruz.** UCSC. [En línea] https://genome.ucsc.edu/.

14. **NCBI, U.S.** GenBank. [En línea] https://www.ncbi.nlm.nih.gov/genbank/.

15. **National Institute of Genetics, (NIG).** DDBJ | DNA Data Bank of Japan. [En línea] www.ddbj.nig.ac.jp.

16. **Olivé, A.** *(2007). Conceptual modeling of information systems. Conceptual Modeling of Information Systems (pp. 1–455). Universitat Politècnica de Catalunya, Department of Software (LSI), Jordi Girona 1-3, E-08034 Barcelona Catalonia, Spain: Springer Berlin Heidelber.*

17. **N. W. Paton, S. A. Khan, A. Hayes, F. Moussouni, A. Brass, K. Eilbeck, C. A. Goble, S. J. Hubbard, and S. G. Oliver.** *(2000) "Conceptual modelling of genomic information," Bioinformatics, vol. 16, p. 548.*

18. **Gray, P. M., Paton, N. W., Kemp, G. J., & Fothergill, J. E.** *(1990). An object-oriented database for protein structure analysis. Protein Eng, 3(4), 235–243. https://doi.org/10.1093/protein/3.4.235.*

19. **Ram, S., and Wei, W. (2004). Modeling the semantics of 3D protein structures. ER2004, pages 696-708.**

20. **Pastor, O.** *(2008). Conceptual modeling meets the human genome. In Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) (Vol. 5231 LNCS, pp. 1–11). https://doi.org/10.1007/978-3-540-87877.*

21. **Reyes R., J. F., Pastor, O., Casamayor, J. C., Valverde, F.** *(2016). Applying Conceptual Modeling to Better Understand the Human Genome. ER2016 (Gifu, Japan), pages 404-412.*

22. **Pastor, O., Reyes J. F., Valverde, F.** *(2016). Conceptual Schema of the Human Genome (CSHG). Tech. Rep. from http://hdl.handle.net/10251/67297.*

23. **GemBiosoft.** *(2013) Base de datos v3. Resumen: Presentación de la base de Datos v3 (Estructura).*

24. **Garrison, E.** vcflib - A C++ library for parsing and manipulating VCF files. [En línea] https://github.com/vcflib/vcflib.

25. **Danecek, P., Auton, A., Abecasis, G., Albers, C. A., Banks, E., DePristo, M. A., ... Durbin, R.** *(2011). The variant call format and VCFtools. Bioinformatics, 27(15), 2156–2158. https://doi.org/10.1093/bioinformatics/btr330.*

26. **Project, 1000 Genomes.** VCFtools projec. [En línea] http://vcftools.sourceforge.net/.

27. **Biostars.** Tool: Vcflib Documentation. [En línea] https://www.biostars.org/p/67390/.

28. **Browning, Brian L.** BEAGLE Utilities. University of Washington. [En línea] https://faculty.washington.edu/browning/beagle_utilities/utilities.html.

29. **MGH, The Analytic and Translational Genetics Unit - Department of Medicine at.** Plink/Seq - A library of the analysis of genetic variation data. [En línea] https://atgu.mgh.harvard.edu/plinkseq/.

30. **Birney, E., Andrews, T. D., Bevan, P., Caccamo, M., Chen, Y., Clarke, L., ... Clamp, M.** *(2004, May). An overview of Ensembl. Genome Research. https://doi.org/10.1101/gr.1860604.*

31. **Herrero, J., Muffato, M., Beal, K., Fitzgerald, S., Gordon, L., Pignatelli, M., ... Flicek, P.** *(2016). Ensembl comparative genomics resources. Database: The Journal of Biological Databases and Curation, 2016, bav096. http://doi.org/10.1093/database/bav096.*

32. **Tyner, C., Barber, G. P., Casper, J., Clawson, H., Diekhans, M., Eisenhart, C., … Kent, W. J.** *(2017). The UCSC Genome Browser database: 2017 update. Nucleic Acids Research, 45(Database issue), D626–D634. http://doi.org/10.1093/nar/gkw1134.*

33. **Benson, D. A., Cavanaugh, M., Clark, K., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J., & Sayers, E. W.** *(2013). GenBank. Nucleic Acids Research, 41(Database issue), D36–D42. http://doi.org/10.1093/nar/gks1195.*

34. **Kulikova, T., Aldebert, P., Althorpe, N., Baker, W., Bates, K., Browne, P., … Apweiler, R.** *(2004). The EMBL Nucleotide Sequence Database. Nucleic Acids Research, 32(Database issue), D27-30. https://doi.org/10.1093/nar/gkh120.*

35. **Mashima J., Kodama Y., Kosuge T., Fujisawa T., Katayama T., Nagasaki H., Okuda Y., Kaminuma E., Ogasawara O., Okubo K., Nakamura Y., Takagi T.** *(2016) DNA data bank of Japan (DDBJ) progress report. Nucleic Acids Research, 44(Database issue), D51–D57. http://doi.org/10.1093/nar/gkv1105.*

36. **Schaafsma, G. C.P. and Vihinen, M.** *(2015), VariSNP, A Benchmark Database for Variations From dbSNP. Human Mutation, 36: 161–166. doi:10.1002/humu.22727.*

37. **González-Izarzugaza, J.** (2008) Introducción a las bases de datos en biología molecular. [En línea] http://www.bbm1.ucm.es/masterbioinfo08/images/materialmaster/secuencias/teoria1.pdf.

38. **wwPDB.** (2008) Protein Data Bank Contents Guide. [En línea] ftp://ftp.wwpdb.org/pub/pdb/doc/format_descriptions/Format_v32_letter.pdf.

39. **Johns Hopkins University.** OMIM - Online Mendelian Inheritance in Man. [En línea] https://www.omim.org/.

40. **M. García, J. Peretó, F. Sapiña, D. Ramón, F.J. Morales, J. Fabregat, P.J. García, M. Edwards, F. González.** *Ciencias para el mundo contemporáneo (pp. 102 - 104).* s.l. : ECIR Editorial.

41. **Alberts, B., Bray, D., Hopkin, K., Johnson, A., Lewis, J., Raff, M., … Walter, P.** *(2009). Essential Cell Biology. Essential Cell Biology (p. 731). https://doi.org/citeulike-article-id:4505949.*

42. **Pierce, Benjamin A.** *(2007) Genética, un enfoque conceptual.*

43. **B. Alberts, D. Bray, A. Johnson, J. Lews, M. Raff, K. Roberts, P. Walter.** *(2005) Introducción a la biología molecular (pp. 226 - 237).* s.l. : Omega.

44. **M. García, M.A. García, J. Furió.** *Biología. Las leyes de la herencia (pp.262 - 268).*

45. —. *Biología. Las leyes de la herencia (pp. 236 - 237).*

46. **(ITU), International Telecommunication Union.** Introduction to ASN.1. [En línea] http://www.itu.int/en/ITU-T/asn1/Pages/introduction.aspx.

47. **NCBI, U.S.** Query Input and database selection. [En línea] https://blast.ncbi.nlm.nih.gov/Blast.cgi?CMD=Web&PAGE_TYPE=BlastDocs&DOC_TYPE=BlastHelp.

48. **W3Schools.** Introduction to XML. [En línea] https://www.w3schools.com/xml/xml_whatis.asp.

49. **Hejtmancik, J. F.** *(2008, April). Congenital cataracts and their molecular genetics. Seminars in Cell and Developmental Biology. https://doi.org/10.1016/j.semcdb.2007.10.003.*

50. **Shiels, A., & Hejtmancik, J. F.** *(2013, August). Genetics of human cataract. Clinical Genetics. https://doi.org/10.1111/cge.12182.*

51. **Beby F, Commeaux C, Bozon M, Denis P, Edery P, Morlé L.** *(2007) New phenotype associated with an Arg116Cys mutation in the CRYAA gene: nuclear cataract, iris coloboma, and microphthalmia. Archives of Ophthalmology, 125(2), 213–6. https://doi.org/10.1001/archopht.125.2.213.*

52. **Cobb, B. A., & Petrash, J. M.** *(2000). Structural and functional changes in the ??A-crystallin R116C mutant in hereditary cataracts. Biochemistry, 39(51), 15791–15798. https://doi.org/10.1021/bi001453j.*

53. **Fu, L., & Liang, J. J. N.** *(2002). Detection of protein-protein interactions among lens crystallins in a mammalian two-hybrid system assay. Journal of Biological Chemistry, 277(6), 4255–4260. https://doi.org/10.1074/jbc.M110027200.*

54.

55. **Faiyaz-Ul-Haque, M., Zaidi, S. H. E., Al-Mureikhi, M. S., Peltekova, I., Tsui, L. C., & Teebi, A. S.** *(2007, August). Mutations in the CHX10 gene in non-syndromic microphthalmia/anophthalmia patients from Qatar [5]. Clinical Genetics. https://doi.org/10.1111/j.1399-0004.2007.00846.x.*

56. **Richter, L., Flodman, P., Von-Bischhoffshausen, F. B., Burch, D., Brown, S., Nguyen, L., … Bateman, J. B.** *(2008). Clinical variability of autosomal dominant cataract, microcornea and corneal opacity and novel mutation in the alpha A crystallin gene (CRYAA). American Journal of Medical Genetics, Part A, 146(7), 833–842. https://doi.org/10.1002/ajmg.a.32236.*

57. **Javadiyan, S., Craig, J. E., Souzeau, E., Sharma, S., Lower, K. M., Pater, J., … Burdon, K. P.** *(2016). Recurrent mutation in the crystallin alpha A gene associated with inherited paediatric cataract. BMC Research Notes, 9, 83. http://doi.org/10.1186/s13104-016-1890-0.*

58. **Litt, M., Kramer, P., LaMorticella, D. M., Murphey, W., Lovrien, E. W., & Weleber, R. G.** *(1998). Autosomal dominant congenital cataract associated with a missense mutation in the human alpha crystallin gene CRYAA. Human Molecular Genetics, 7(3), 471–474. https://doi.org/10.1093/hmg/7.3.471.*

59. **Sagona AP, Nezis IP, Stenmark H. Association of CHMP4B and Autophagy with Micronuclei: Implications for Cataract Formation. BioMed Research International. 2014 y doi:10.1155/2014/974393., 2014:974393.**

60. **Santhiya, Sathiyavedu T. et al.** *Identification of a novel, putative cataract-causing allele in CRYAA (G98R) in an Indian family.* 2006.

61. **Eye Genetics Research Group, Children's Medical Research Institute.** Eye Genetics Research Group. [En línea] http://www.cmri.org.au/Research/Research-Units/Eye-Genetics.

62. **Muñoz, L., Mazón, J. N., Trujillo, J., Munoz, L., & Mazon, J.-N.** (2011). ETL Process Modeling Conceptual for Data Warehouses: A Systematic Mapping Study. IEEE Latin America Transactions, 9(3), 360–365. https://doi.org/10.1109/TLA.2011.5893784.

63. **NCBI, U.S.** National Center for Biotechnology Information. [En línea] https://www.ncbi.nlm.nih.gov/.

64. —. Gene. [En línea] https://www.ncbi.nlm.nih.gov/gene.

65. —. SNP. [En línea] https://www.ncbi.nlm.nih.gov/projects/SNP/.

66. —. ClinVar. [En línea] https://www.ncbi.nlm.nih.gov/clinvar/.

67. —. Nucleotide. [En línea] https://www.ncbi.nlm.nih.gov/nucleotide.

68. —. PubMed. [En línea] https://www.ncbi.nlm.nih.gov/pubmed.

69. —. NCBI Data in XML. [En línea] https://www.ncbi.nlm.nih.gov/data_specs/NCBI_data_in_XML.html.

70. **Corporation, Oracle.** Package org.xml.sax Description. [En línea] https://docs.oracle.com/javase/7/docs/api/org/xml/sax/package-summary.html#package_description.

71. **Reyes R., J.F., Iñiguez, C. & Pastor, O.** *(2017). GenesLove.Me: A Model-based Web-application for Direct-to-consumer Genetic Tests. 133-143. 10.5220/0006340201330143.*

72. **Conley, Y. P., Erturk, D., Keverline, A., Mah, T. S., Keravala, A., Barnes, L. R., … Gorin, M. B.** *(2000). A juvenile-onset, progressive cataract locus on chromosome 3q21-q22 is associated with a missense mutation in the beaded filament structural protein-2. American Journal of Human Genetics, 66(4), 1426–31. https://doi.org/10.1086/302871.*

73. **PLINK/SEQ.** A library for the analysis of genetic variation. *PLINK/SEQ - Downloads.* [En línea] https://atgu.mgh.harvard.edu/plinkseq/download.shtml.

74. **Hansen, L., Yao, W., Eiberg, H., Kjaer, K. W., Baggesen, K., Hejtmancik, J. F., & Rosenberg, T.** *(2007). Genetic heterogeneity in microcornea-cataract: Five novel*

mutations in CRYAA, CRYGD, and GJA8. Investigative Ophthalmology and Visual Science, 48(9), 3937–3944. https://doi.org/10.1167/iovs.07-0013.

75. **Mackay, D. S., Andley, U. P., & Shiels, A.** *(2003). Cell death triggered by a novel mutation in the alphaA-crystallin gene underlies autosomal dominant cataract linked to chromosome 21q. European Journal of Human Genetics, 11(10), 784–793. https://doi.org/10.1038/sj.ejhg.5201046.*

76. **Ferda Percin, E., Ploder, L. a, Yu, J. J., Arici, K., Horsford, D. J., Rutherford, a, … McInnes, R. R.** *(2000). Human microphthalmia associated with mutations in the retinal homeobox gene CHX10. Nature Genetics, 25(4), 397–401. https://doi.org/10.1038/78071.*

77. **Pras, E., Frydman, M., Levy-Nissenbaum, E., Bakhan, T., Raz, J., Assia, E. I., … Pras, E.** *(2000). A nonsense mutation (W9X) in CRYAA causes autosomal recessive cataract in an inbred Jewish Persian family. Investigative Ophthalmology and Visual Science, 41(11), 3511–3515.*

78. **Vanita, V., Singh, J. R., Hejtmancik, J. F., Nuernberg, P., Hennies, H. C., Singh, D., & Sperling, K.** *(2006). A novel fan-shaped cataract-microcornea syndrome caused by a mutation of CRYAA in an Indian family. Molecular Vision, 12(May), 518–522.*

79. **Yamada, K., Tomita, H. A., Kanazawa, S., Mera, A., Amemiya, T., & Niikawa, N.** *(2000). Genetically distinct autosomal dominant posterior polar cataract in a four-generation Japanese family. American Journal of Ophthalmology, 129(2), 159–165. https://doi.org/10.1016/S0002-9394(99)00313-X.*

80. **Shiels, A., Bennett, T. M., Knopf, H. L. S., Yamada, K., Yoshiura, K., Niikawa, N., … Hanson, P. I.** *(2007). CHMP4B, a novel gene for autosomal dominant cataracts linked to chromosome 20q. American Journal of Human Genetics, 81(3), 596–606. https://doi.org/10.1086/519980.*

ANEXO I: GLOSARIO

ADN polimerasa: Enzima que sintetiza el ADN.

Alelo: Una de dos o más formas alternativas de un gen.

Aminoácido: Unidad repetitiva de las proteínas; consiste en un grupo amino, un grupo carboxilo, un átomo de hidrógeno y un grupo R variables.

Anticodón: Secuencia de tres nucleótidos del ARNt que se aparea con el codón correspondiente del ARNm durante la traducción.

Antígeno: Sustancia que es reconocida por el sistema inmune y que provoca una respuesta inmune.

ARN mensajero: Molécula de ARN que contiene información genética sobre la secuencia de aminoácidos de una proteína.

ARN polimerasa: Enzima que sintetiza ARN a partir de un molde de ADN durante la transcripción.

ARN ribosómico: Molécula de ARN que constituye un componente estructural del ribosoma.

ARN transferencia: Molécula de ARN que lleva un aminoácido al ribosoma y lo transfiere a una cadena polipeptídica creciente durante la traducción.

Base nitrogenada: Base que contiene nitrógeno y constituye una de tres partes del nucleótido.

Cadena molde: Cadena de ADN que se utiliza como molde durante la transcripción. El ARN sintetizado durante la transcripción es complementario y antiparalelo respecto a la cadena molde.

Cariotipo: Gráfico de todos los cromosomas en metafases de un individuo.

Citosina (C): Pirimidina del ADN y ARN.

Codón: Secuencia de tres nucleótidos que codifica una aminoácido de una proteína.

Codón de iniciación: Codón del ARNm que especifica el primer aminácido.

Codón de terminación: Codón del ARMm que marca el final de la traducción.

Deleción: Mutación en la que se elimina nucleótidos de una secuencia de ADN.

Deleción cromosómica: Pérdida de un segmento cromosómico.

Desnaturalización: Proceso de separación de las cadenas de ADN por calentamiento.

Dexoxirribonucleótido: Componente estructural básico del ADN, que consiste en un azúcar desoxirribosa, un fosfato y una base nitrogenada.

Dominante: Se refiere a un alelo o a un fenotipo que se expresa en los homocigotos (AA) y en los heterocigotos (Aa); solo el alelo dominante se expresa en un fenotipo heterocigótico.

Entrecruzamiento: Intercambio de material genético entre cromátidas homólogas pero no hermanas.

Exón: Región codificadora de un gen fragmentado.

Fenotipo: Apariencia o manifestación de una característica genética.

Gen: Factor genético que contribuye a determinar una características.

Genoma: Totalidad de instrucciones genéticas presentes en un organismo.

Genotipo: Conjunto de genes que posee un individuo.

Guanina (G): Purina del ADN y ARN.

Haplotipo: Conjunto específico de variantes genéticas o alelos en un cromosoma individual o en parte de un cromosoma.

Homocigoto: Individuo que tiene dos alelos idénticos en un locus.

Inserción: Mutación por la cual se agregan nucleótidos a una secuencia de ADN.

Intrón: Secuencia interpuesta de un gen dividido; es eliminado del ARN después de la transcripción.

Locus: Posición en la que un determinado gen se ubica dentro un cromosoma.

Mutación: Cambio heredable en la información genética.

Mutación génica: La que afecta a un solo gen o alelo.

Pirimidina: Tipo de base nitrogenada del ADN y ARN. La citosina, la timina y el uracilo son pirimidinas.

Purina: Tipo de base nitrogenada del ADN y ARN: La adenina y la guanina son purinas.

Recesivo: Se refiere a un alelo o fenotipo que se expresa tan solo cuando se da en homocigoto.

Replicación: Proceso por el cual el ADN se sintetiza a partir de un molde de nucleótidos de cadena simple.

Represor: Proteína reguladora que se une a una secuencia de ADN e inhibe la transcripción.

Ribonucleótido: Nucleótido que contiene un ribosoma; se encuentra en el ARN.

Ribosoma: Azúcar de cinco carbonos de ARN.

Subunidad ribosómica grande: La más grande de las dos subunidades de un ribosoma funciona.

Subunidad ribosómica pequeña: La más pequeña de las dos subunidades de un ribosoma funcional.

Timina (T): Pirimidina presente en ADN pero no en ARN:

Traducción: Proceso por el cual se sintetiza una proteína a partir de la información contenida en el ARN mensajero.

Uracilo (U): Pirimidina presente en al ARN pero casi nunca en el ADN.

ANEXO II: CÓDIGO FUENTE DESTACADO

Main.java

```
//Proceso de espera de extracción de información
private void extractProcess(ActionEvent arg0){

		SwingWorker<Boolean, Void> worker = new SwingWorker<Boolean, Void>() {

			@Override
			protected Boolean doInBackground() throws Exception {
				btn_ChargeData.setEnabled(false);
				completaEstructura(txt_IdVariation.getText());

				if(connectionMySQL.checkConnection()){
					btn_CheckDB.setEnabled(true);
					btn_ChargeData.setEnabled(true);
				}
				else{
					txt_EstadoConexion.setText("DESCONECTADA");
					txt_EstadoConexion.setBackground(Color.red);
					btn_CheckDB.setEnabled(false);
				}
				return true;
			}
			@Override
			protected void done() {
				frameLoading.dispose();
			}
		};

		frameLoading = new FrameLoading();
		frameLoading.setLocationRelativeTo(this);
		frameLoading.setVisible(true);
		frameLoading.setDefaultCloseOperation(javax.swing.JFrame.EXIT_ON_CLOSE);

		worker.execute();
}

//Proceso creación de ChromosomeElement
private void completaEstructura(String rsVariationId){

		int variationId = 0;
		int continueCharge = 0;

		//Comprobación tipo de Id introducido
		if(rsVariationId.equals("") || rsVariationId.equals("0")){
			JOptionPane.showMessageDialog(this, "Introduzca Id variación válido", "Falló la carga",
JOptionPane.ERROR_MESSAGE);
		}else{
			//Comprobación variación a documentar
			variationId = Integer.parseInt(rsVariationId);
			if(chromosomeElement != null){
				if (chromosomeElement.getGene().getVariations().get(0).getDbVariationId()
== variationId){
```

```
                                        JOptionPane.showMessageDialog(this, "La variacion ya está cargada", "No es necesario cargar", JOptionPane.WARNING_MESSAGE);
                                    }
                                    else
                                        continueCharge = 1;
                                }
                                else
                                    continueCharge = 1;
                            }

                            //Asociación de información y creación de elementos del cromosoma
                            if(continueCharge == 1){

                                chromosomeElement = new ChromosomeElement();

                                Genome genome = new Genome();
                                Chromosome chromosome = new Chromosome();

                                Gene gene = new Gene();
                                Variation variation = new Variation();
                                ArrayList<Variation> variations;

                                Transcript transcript = new Transcript();
                                SequenceNg sequenceNg = new SequenceNg();
                                Protein protein = new Protein();
                                Exon exon = new Exon();
                                ExonTranscript exonTranscript = new ExonTranscript();

                                Precise precise = new Precise();
                                PreciseSeqNg preciseSeqNg = new PreciseSeqNg();

                                Databank databank = new Databank();
                                DatabankVersion databankVersion = new DatabankVersion();
                                ElementDatabank elementDatabank = new ElementDatabank();

                                BibliographyReference bibliographyReference = new BibliographyReference();
                                ArrayList<BibliographyReference> bibliographyReferences = new ArrayList<BibliographyReference>();

                                /************** GENE & VARIATIONS *************/
                                variations = readVariations(variationId);
                                gene = readGene(variations.get(0).getGeneId());
                                gene.setVariations(variations);

                                /****************** CHR_ELEM ******************/
                                chromosomeElement.setNcIdentifier(variations.get(0).getNcIdentifier());
                                chromosomeElement.setStartPosition(variations.get(0).getChrStartPosition());
                                chromosomeElement.setEndPosition(variations.get(0).getChrEndPosition());
                                chromosomeElement.setStrand(variations.get(0).getStrand());
                                chromosomeElement.setSpecializationType("transcribable_element");
                                chromosomeElement.setGene(gene);
                                gene.setChromosomeElement(chromosomeElement);

                                /****************** GENOME ******************/
                                genome.setGrchIdentifier(variations.get(0).getGrchIdentifier());
                                if (genome.getGrchIdentifier().equals("GRCh38"))
```

```
                        genome.setHgIdentifer("HG_38");
                    else if(genome.getGrchIdentifier().equals("GRCh37"))
                        genome.setHgIdentifer("HG_19");
                    else
                        genome.setHgIdentifer("HG_");
                    chromosome.setGenome(genome);

                    /**************** CHROMOSOME ****************/
                    chromosome.setHgIdentifier(genome.getHgIdentifer());
                    chromosome.setNcIdentifier(chromosomeElement.getNcIdentifier());
                    if(chromosome.getNcIdentifier().contains("NC_0000"))
                        chromosome.setName("Chr"+chromosome.getNcIdentifier().substring(7,
chromosome.getNcIdentifier().length()-3));
                    if(chromosome.getNcIdentifier().contains("NC_00000"))
                        chromosome.setName("Chr"+chromosome.getNcIdentifier().substring(8,
chromosome.getNcIdentifier().length()-3));
                    chromosomeElement.setChromosome(chromosome);

                    /************* DATABANK_VERSION **************/
                    databank.setName("ClinVar");
                    databank.setUrl("https://www.ncbi.nlm.nih.gov/clinvar/");
                    databank.setDescription("ClinVar aggregates information about genomic variation and its
relationship to human health");

                    databankVersion.setDatank(databank);
                    databankVersion.setRelease("April 2013");
                    databankVersion.setDate(new java.sql.Date(calendar.getTime().getTime()));
                    databank.setDatabankVersion(databankVersion);

                    /***************** VARIATIONS *****************/
                    for(int i=0; i<variations.size(); i++){

                        variation = variations.get(i);
                        variation.setGene(gene);
                        variation.setDatabankVersion(databankVersion);
                        databankVersion.addVariation(variation);

                        preciseSeqNg = variation.getPrecise().getPreciseSeqNg();

                        precise = new Precise();
                        precise.setPreciseSeqNg(preciseSeqNg);
                        precise.setVariation(variation);
                        precise.setSpecializationType(variation.getPrecise().getSpecializationType());
                        precise.setInsSequence(variation.getPrecise().getInsSequence());
                        precise.setInsRepetition(variation.getPrecise().getInsRepetition());
                        precise.setNumBases(variation.getPrecise().getNumBases());
                        precise.setFlankingLeft(readPrecise(variation).getFlankingLeft());
                        precise.setFlankingRight(readPrecise(variation).getFlankingRight());

                        variation.setPrecise(precise);
                        preciseSeqNg.setPrecise(precise);
                        preciseSeqNg.setSequenceNg(sequenceNg);

                        transcript = readTranscript(preciseSeqNg.getNgIdentifier());
                        transcript.setChromosomeElement(chromosomeElement);
                        transcript.setBiotype(gene.getBiotype());
```

```
                                        transcript.setNmIdentifier(variation.getNmIdentifier());
                                        transcript.setNgIdentifier(preciseSeqNg.getNgIdentifier());
                                        protein = readProtein(variation.getNpIdentifier());
                                        protein.setTranscript(transcript);
                                        transcript.setProtein(protein);
                                        transcript.setGene(gene);

                                        bibliographyReferences = new ArrayList<BibliographyReference>();
                                        for (int j=0; j<variation.getBibliographyIds().size(); j++){
                                                bibliographyReference =
readBibliographyReference(variation.getBibliographyIds().get(j));
                                                bibliographyReferences.add(bibliographyReference);
                                        }
                                        variation.setBibliographyReferences(bibliographyReferences);
                                }
                                chromosomeElement.setTranscript(transcript);


                                /*************** ELEMENT DATABANK ***************/
                                elementDatabank.setChromosomeElement(chromosomeElement);
                                elementDatabank.setDatabankVersion(databankVersion);
                                elementDatabank.setSourceIdentifier(transcript.getSourceIdentifier());
                                chromosomeElement.setElementDatabank(elementDatabank);

                                /***************** SEQUENCE NG ******************/
                                sequenceNg = transcript.getSequenceNg();
                                sequenceNg.setGene(gene);
                                sequenceNg.addVariation(variation);
                                sequenceNg.setPreciseSeqNg(preciseSeqNg);
                                sequenceNg.setTranscript(transcript);
                                sequenceNg.setNgIdentifier(preciseSeqNg.getNgIdentifier());

                                /******************** EXONS ***********************/
                                exon.setIdSymbol(gene.getIdSymbol());
                                exon.setChromosomeElement(chromosomeElement);
                                ArrayList<Exon> listExon = transcript.getExons();
                                for(int i=0;i<listExon.size();i++){
                                        if(listExon.get(i).getStartExonNg() <= preciseSeqNg.getPosition() &&
                                                        listExon.get(i).getEndExonNg()>=
preciseSeqNg.getPosition()){
                                                exon.setName(listExon.get(i).getName());
                                                exon.setStartExonNg(listExon.get(i).getStartExonNg());
                                                exon.setEndExonNg(listExon.get(i).getEndExonNg());
                                        }
                                }

                                /*************** EXONS TRANSCRIPT ****************/
                                exonTranscript.setExon(exon);
                                exonTranscript.setTranscript(transcript);
                                if(exon.getName().matches("[0-9]+"))
                                        exonTranscript.setName(exon.getName());
                                else
                                        exonTranscript.setName(exon.getName().substring(0,
exon.getName().length()-1));
                                exon.setExonTranscript(exonTranscript);
                                transcript.setExonTranscript(exonTranscript);
```

```
                        //LLamada a la actualización de datos de la interfaz
                        printData(chromosomeElement);
                }
}

//Método creación de variaciones a partir de Id
private ArrayList<Variation> readVariations(int rsVariation) {

                ArrayList<Variation> listVariations = new ArrayList<Variation>();
                ArrayList<Integer> clinVariations = new ArrayList<Integer>();
                Variation variationAux;

                clinVariations = snpToClinvarId(rsVariation);
                for(int i=0; i< clinVariations.size(); i++){
                        variationAux = readClinvar(clinVariations.get(i));
                        variationAux.setDbVariationId(rsVariation);
                        listVariations.add(variationAux);
                }
                return listVariations;
}

//Método extracción identificador ClinVar
private ArrayList<Integer> snpToClinvarId(int rsVariationId) {
                String url;
                ArrayList<Integer> clinVariationId = new ArrayList<Integer>();
                SAXParserFactory saxParserFactory = SAXParserFactory.newInstance();
                SAXParser saxParser;
                try {
                        saxParser = saxParserFactory.newSAXParser();
                        XMLSnpToClinvar handlerXML = new XMLSnpToClinvar();
                        url =
"https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=snp&db=clinvar&id="+rsVariationId;
                        saxParser.parse(new InputSource(new URL(url).openStream()), handlerXML);
                        clinVariationId = handlerXML.getVariationsId();
                } catch (ParserConfigurationException | SAXException | IOException | NumberFormatException e) {
                        JOptionPane.showMessageDialog(this, "Falló el cambio de identificador", "Error en
snpToClinvarId()", JOptionPane.ERROR_MESSAGE);
                }
                return clinVariationId;
}

//Método extracción de información de ClinVar
private Variation readClinvar(int variationId) {
                String url;
                Variation variation = null;
                SAXParserFactory saxParserFactory = SAXParserFactory.newInstance();
                SAXParser saxParser;
                try {
                        saxParser = saxParserFactory.newSAXParser();
                        XMLClinvarHandler handlerXML = new XMLClinvarHandler();
                        url =
"https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=clinvar&rettype=variation&id="+variationId;
                        saxParser.parse(new InputSource(new URL(url).openStream()), handlerXML);
                        variation = handlerXML.getVariation();
                } catch (ParserConfigurationException | SAXException | IOException | NumberFormatException e) {
```

```
                        JOptionPane.showMessageDialog(this, "Error en la extracción de variaciones", "Fallo en
readClinvar()", JOptionPane.ERROR_MESSAGE);
                }
                return variation;
}

//Método creación/asignación Precise
private Precise readPrecise(Variation variation) {
                Variation snpVariation = readSnp(variation.getDbVariationId());
                Precise preFinal = snpVariation.getPrecise();
                return preFinal;
}

//Método extracción de información de dbSNP (Precise, Flankings)
private Variation readSnp(int rsVariationId) {
                String url;
                Variation variation = null;
                SAXParserFactory saxParserFactory = SAXParserFactory.newInstance();
                SAXParser saxParser;
                try {
                        saxParser = saxParserFactory.newSAXParser();
                        XMLSnpHandler handlerXML = new XMLSnpHandler();
                        url =
"https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=snp&id="+rsVariationId+"&retmode=xml";
                        saxParser.parse(new InputSource(new URL(url).openStream()), handlerXML);
                        variation = handlerXML.getVariation();
                } catch (ParserConfigurationException | SAXException | IOException | NumberFormatException e) {
                        JOptionPane.showMessageDialog(this, "Error en la estracción de Snp", "Fallo en
readSnp()", JOptionPane.ERROR_MESSAGE);
                }
                return variation;
}

//Método extracción de información de Gene
private Gene readGene(int genId) {
                String url;
                Gene gene = null;
                SAXParserFactory saxParserFactory = SAXParserFactory.newInstance();
                SAXParser saxParser;
                try {
                        saxParser = saxParserFactory.newSAXParser();
                        XMLGeneHandler handlerXML = new XMLGeneHandler();
                        url =
"https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=gene&id="+genId+"&retmode=xml";
                        saxParser.parse(new InputSource(new URL(url).openStream()), handlerXML);
                        gene = handlerXML.getGene();
                } catch (ParserConfigurationException | SAXException | IOException | NumberFormatException e) {
                        JOptionPane.showMessageDialog(this, "Error en la extracción de genes", "Fallo en
readGene()", JOptionPane.ERROR_MESSAGE);
                }
                return gene;
}

//Método extracción de información de Pubmed
private BibliographyReference readBibliographyReference(int bibRefID) {
                String url;
```

```
                BibliographyReference bibliographyRef = null;
                SAXParserFactory saxParserFactory = SAXParserFactory.newInstance();
                SAXParser saxParser;
                try {
                                saxParser = saxParserFactory.newSAXParser();
                                XMLPubmedHandler handlerXML = new XMLPubmedHandler();
                                url =
"https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=pubmed&id="+bibRefID+"&retmode=xml";
                                saxParser.parse(new InputSource(new URL(url).openStream()), handlerXML);
                                bibliographyRef = handlerXML.getbibRef();
                } catch (ParserConfigurationException | SAXException | IOException | NumberFormatException e) {
                                JOptionPane.showMessageDialog(this, "Error en la extracción de referencias
bibliográficas", "Fallo en readBibliography()", JOptionPane.ERROR_MESSAGE);
                }
                BibliographyDb bibliographyDb = new BibliographyDb();
                bibliographyDb.setNameDb("PudMed");
                bibliographyDb.setPubmedId(bibRefID);
                bibliographyDb.setUrl("https://www.ncbi.nlm.nih.gov/pubmed/"+bibRefID);
                bibliographyDb.setBibliographyReference(bibliographyRef);
                bibliographyRef.setBibliographyDb(bibliographyDb);

                return bibliographyRef;
}

//Método creación frame de gestión de conexión
private void openConnectionFrame(ActionEvent evt){
                connectionFrame = new FrameConnection(databaseConnection, connectionMySQL);
                connectionFrame.setLocationRelativeTo(this);
                connectionFrame.setVisible(true);
                connectionFrame.setDefaultCloseOperation(javax.swing.JFrame.DISPOSE_ON_CLOSE);

                //Cuando se cierra ese frame se actualiza la ventana principal
                connectionFrame.addWindowListener(new WindowAdapter() {
                                @Override
                                public void windowClosed(final WindowEvent event) {
                                                txt_ServerDatabase.setText(databaseConnection.getHost());
                                                txt_UserDatabase.setText(databaseConnection.getUser());
                                                txt_PassDatabase.setText(databaseConnection.getPass());

                                                if(connectionMySQL.getConexion()!=null){
                                                                btn_ChargeData.setEnabled(true);
                                                                tableModelCurator.loadCurators();
                                                                txt_EstadoConexion.setText("CONECTADA");
                                                                txt_EstadoConexion.setBackground(Color.green);
                                                }
                                                if(connectionMySQL.getConexion()==null){
                                                                btn_CheckDB.setEnabled(false);
                                                }
                                }
                });
}

//Método creación frame para la visualización de la secuencia de Chromosome
private void openFrameChromosome(){
                frameChromosome = new FrameChromosome(chromosomeElement.getNcIdentifier());
                frameChromosome.setLocationRelativeTo(this);
```

```
                frameChromosome.setVisible(true);
                frameChromosome.setDefaultCloseOperation(javax.swing.JFrame.DISPOSE_ON_CLOSE);

}

//Método creación frame para la visualización de la secuencia de Protein
private void openFrameProtein(ActionEvent evt){
                frameProtein = new FrameProtein(chromosomeElement.getTranscript().getProtein());
                frameProtein.setLocationRelativeTo(this);
                frameProtein.setVisible(true);
                frameProtein.setDefaultCloseOperation(javax.swing.JFrame.DISPOSE_ON_CLOSE);
}

//Método creación frame para la visualización de la secuencia de SequenceNg
private void openFrameSequenceNg(ActionEvent evt){
                frameSequenceNg = new FrameSequenceNg(chromosomeElement.getTranscript().getSequenceNg());
                frameSequenceNg.setLocationRelativeTo(this);
                frameSequenceNg.setVisible(true);
                frameSequenceNg.setDefaultCloseOperation(javax.swing.JFrame.DISPOSE_ON_CLOSE);
}

//Método creación frame para la validación de variaciones
private void openFrameValidation(){
                frameValidation = new FrameValidation(connectionMySQL, chromosomeElement);
                frameValidation.setLocationRelativeTo(this);
                frameValidation.setVisible(true);
                frameValidation.setDefaultCloseOperation(javax.swing.JFrame.DISPOSE_ON_CLOSE);


                frameValidation.addWindowListener(new WindowAdapter() {
                                @Override
                                public void windowClosed(final WindowEvent event) {

                                                if(connectionMySQL.getConexion()!=null ||

                chromosomeElement.getGene().getVariations().get(0).getValidations().get(0) != null){
                                                }
                                }
                });
}

//Método para borrar usuarios
private void deleteCurator(){
                if(JOptionPane.showOptionDialog(this, "¿Desea borrar el usuario?", "Eliminar usuario",
JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE, null, new Object[] { "Sí", "No"}, "No")== 0){
                                if(tableModelCurator.deleteCurator(tableCurator.getSelectedRow()))
                                                JOptionPane.showMessageDialog(this, "Usuario eliminado con éxito",
"Eliminar usuario", JOptionPane.INFORMATION_MESSAGE);
                                else
                                                JOptionPane.showMessageDialog(this, "El usuario no puede ser eliminado",
"Eliminar usuario", JOptionPane.ERROR_MESSAGE);
                }
                tableCurator.clearSelection();
}
```

```
//Métodos para crear usuarios
private void createCurator(){
		frameCurator = new FrameCurator(1, tableModelCurator);
		frameCurator.setLocationRelativeTo(this);
		frameCurator.setVisible(true);
		frameCurator.setDefaultCloseOperation(javax.swing.JFrame.DISPOSE_ON_CLOSE);
		tableCurator.clearSelection();
}

//Métodos para actualizar datos usuarios
private void updateCurator(){
		Curator curator = tableModelCurator.getCurator(tableCurator.getSelectedRow());
		frameCurator = new FrameCurator(0, tableModelCurator, curator);
		frameCurator.setLocationRelativeTo(this);
		frameCurator.setVisible(true);
		frameCurator.setDefaultCloseOperation(javax.swing.JFrame.DISPOSE_ON_CLOSE);
		tableCurator.clearSelection();
}

//Método carga en DB
private void insertData(ChromosomeElement chromosomeElement){

		btn_InsertDb.setEnabled(false);
		btn_Validation.setEnabled(false);

		int nextIdVariation = -1;
		int nextIdDatabankversion = -1;
		int nextIdBibliographyReference = -1;

		Variation variation = new Variation();
		BibliographyReference bibliographyReference = new BibliographyReference();
		ReferenceChromosomeElement referenceChromosomeElement = new
ReferenceChromosomeElement();
		ReferenceVariation referenceVariation = new ReferenceVariation();

		if(connectionMySQL.getConexion() == null)
			System.out.println("Conexión no establecida");
		else{
			System.out.println("**********************");

		if(connectionMySQL.existsVariation(String.valueOf(chromosomeElement.getGene().getVariations().get(
0).getDbVariationId())))
				System.out.println("Exists variation");
			else{
				System.out.println("Don't exists variation");

				System.out.println("**********************");

		chromosomeElement.setChromosomeElementId(connectionMySQL.getNextIdentifier("CHR_ELEM"));
				System.out.println("CHR_ELEM_ID:
"+chromosomeElement.getChromosomeElementId());

				System.out.println("**********************");

		if(connectionMySQL.insertGenome(chromosomeElement.getChromosome().getGenome()))
					System.out.println("Insert genome: OK");
```

```
                    else
                            System.out.println("Insert genome: ERROR");

                    System.out.println("************************");

        if(connectionMySQL.insertChromosome(chromosomeElement.getChromosome()))
                            System.out.println("Insert chromosome: OK");
                    else
                            System.out.println("Insert chromosome: ERROR");

                    System.out.println("************************");

                    if(connectionMySQL.insertChromosomeElement(chromosomeElement))
                            System.out.println("Insert cromosoma_element: OK");
                    else
                            System.out.println("Insert cromosoma_element: ERROR");

                    System.out.println("************************");
                    if(connectionMySQL.insertGene(chromosomeElement.getGene()))
                            System.out.println("Insert gene: OK");
                    else
                            System.out.println("Insert gene: ERROR");

                    System.out.println("************************");

        if(connectionMySQL.insertSequenceNg(chromosomeElement.getTranscript().getSequenceNg()))
                            System.out.println("Insert sequenceNg: OK");
                    else
                            System.out.println("Insert sequenceNg: ERROR");

                    System.out.println("************************");
                    if(connectionMySQL.existsTranscript(chromosomeElement.getTranscript()))
                            System.out.println("Exists transcript");
                    else{
                            System.out.println("Don't exists transcrit");

        if(connectionMySQL.insertTranscript(chromosomeElement.getTranscript()))
                                    System.out.println("Insert transcript: OK");
                            else
                                    System.out.println("Insert transcript: ERROR");
                    }

                    System.out.println("************************");

        if(connectionMySQL.insertProtein(chromosomeElement.getTranscript().getProtein()))
                            System.out.println("Insert protein: OK");
                    else
                            System.out.println("Insert protein: ERROR");

                    System.out.println("************************");

        if(connectionMySQL.insertExon(chromosomeElement.getTranscript().getExonTranscript().getExon()))
                            System.out.println("Insert exon: OK");
                    else
                            System.out.println("Insert exon: ERROR");
```

```
                                        System.out.println("************************");

                if(connectionMySQL.insertExonTranscript(chromosomeElement.getTranscript().getExonTranscript()))
                                                System.out.println("Insert exon_transcript: OK");
                                        else
                                                System.out.println("Insert exon_transcript: ERROR");

                                        System.out.println("************************");

                if(connectionMySQL.insertDatabank(chromosomeElement.getElementDatabank().getDatabankVersion(
).getDatank()))
                                                System.out.println("Insert databank: OK");
                                        else
                                                System.out.println("Insert databank: ERROR");


                if(connectionMySQL.existsDatabankVersion(chromosomeElement.getElementDatabank().getDatabank
Version())){
                                                System.out.println("************************");
                                                System.out.println("Exists databankVersion");
                                        }
                                        else{
                                                System.out.println("************************");
                                                System.out.println("Don't exists databankVersion");
                                                nextIdDatabankversion =
connectionMySQL.getNextIdentifier("DATABANK_VERSION");
                                                System.out.println("DB_VERSION_ID: "+nextIdDatabankversion);

                chromosomeElement.getElementDatabank().getDatabankVersion().setDatabankVersionId(nextIdDatab
ankversion);

                connectionMySQL.insertDatabankVersion(chromosomeElement.getElementDatabank().getDatabankVe
rsion());
                                        }

                                        System.out.println("************************");

                if(connectionMySQL.insertElementDatabank(chromosomeElement.getElementDatabank()))
                                                System.out.println("Insert elementDatabank: OK");
                                        else
                                                System.out.println("Insert elementDatabank: ERROR");

                                        for(int i=0;i<chromosomeElement.getGene().getVariations().size();i++){

                                                variation = chromosomeElement.getGene().getVariations().get(i);

                                                System.out.println("************************");
                                                nextIdVariation =
connectionMySQL.getNextIdentifier("VARIATION");
                                                System.out.println("VARIATION_ID: "+nextIdVariation);

                                                variation.setVariationId(nextIdVariation);

                                                System.out.println("************************");
                                                if(connectionMySQL.insertVariation(variation))
                                                        System.out.println("Insert variation: OK");
```

```
                                        else
                                                System.out.println("Insert variation: ERROR");

                                        System.out.println("************************");
                                        if(connectionMySQL.insertPrecise(variation.getPrecise()))
                                                System.out.println("Insert precise: OK");
                                        else
                                                System.out.println("Insert precise: ERROR");

                                        System.out.println("************************");

        if(connectionMySQL.insertPreciseSeqNg(variation.getPrecise().getPreciseSeqNg()))
                                                System.out.println("Insert preciseSeqNg: OK");
                                        else
                                                System.out.println("Insert preciseSeqNg: ERROR");

                                        System.out.println("************************");
                                        if(!variation.getValidations().isEmpty()){

        if(connectionMySQL.insertValidation(variation.getValidations().get(0)))
                                                        System.out.println("Insert validation:
OK");
                                                else
                                                        System.out.println("Insert validation:
ERROR");
                                        }


                                        for(int j=0;j<variation.getBibliographyReferences().size();j++){

                                                bibliographyReference =
variation.getBibliographyReferences().get(j);
                                                nextIdBibliographyReference =
connectionMySQL.getNextIdentifier("BIB_REF");
                                                System.out.println("************************");
                                                System.out.println("Next BIB_REF_ID:
"+nextIdBibliographyReference);

        bibliographyReference.setBibliographyReferenceId(nextIdBibliographyReference);

        if(connectionMySQL.insertBibliographyReference(bibliographyReference))
                                                        System.out.println("Insert
bibliographyReference: OK");
                                                else
                                                        System.out.println("Insert
bibliographyReference: ERROR");

                                                System.out.println("************************");

        if(connectionMySQL.insertBibliographyDb(bibliographyReference.getBibliographyDb()))
                                                        System.out.println("Insert
bibliographyDb: OK");
                                                else
                                                        System.out.println("Insert
bibliographyDb: ERROR");
```

```
                                                referenceVariation = new ReferenceVariation();
                                                referenceVariation.setVariation(variation);

            referenceVariation.setBibliographyReference(bibliographyReference);

                                                System.out.println("**********************");

            if(connectionMySQL.insertReferenceVariation(referenceVariation))
                                                        System.out.println("Insert
referenceVariation: OK");
                                                else
                                                        System.out.println("Insert
referenceVariation: ERROR");

                                                referenceChromosomeElement = new
ReferenceChromosomeElement();

            referenceChromosomeElement.setChromosomeElement(chromosomeElement);

            referenceChromosomeElement.setBibliographyReference(bibliographyReference);

                                                System.out.println("**********************");

            if(connectionMySQL.insertReferenceChromosomeElement(referenceChromosomeElement))
                                                        System.out.println("Insert
referenceChromosomeElement: OK");
                                                else
                                                        System.out.println("Insert
referenceChromosomeElement: ERROR");

                                                }

                                        }

                        }
                        JOptionPane.showMessageDialog(this, "Los datos han sido cargados correctamente",
"Carga en la base de datos", JOptionPane.INFORMATION_MESSAGE);

            }
}
```

MySqlConnecion.java

```
// Método comprobación conexión en DB
public boolean checkConnection(){
            try {
                        sentencia = "SELECT 1 FROM varsearch.GENOME";
                        ps = con.prepareStatement(sentencia);
                        ps.executeQuery();
                        ps.close();
            } catch (SQLException e) {
                        e.printStackTrace();
                        return false;
            }
            return true;
}
```

```
// Método gerenador IDs libres para la inserción
public int getNextIdentifier(String database){
		int nextId = 0;
		switch (database){
		case "CHR_ELEM" :
			sentencia = "SELECT MAX(CHR_ELEM_ID)+1 FROM varsearch.CHR_ELEM";
			break;
		case "BIB_REF":
			sentencia = "SELECT MAX(BIB_REF_ID)+1 FROM varsearch.BIB_REF";
			break;
		case "VARIATION":
			sentencia = "SELECT MAX(VARIATION_ID)+1 FROM varsearch.VARIATION";
			break;
		case "DATABANK_VERSION":
			sentencia = "SELECT MAX(DB_VERSION_ID)+1 FROM varsearch.DATABANK_VERSION";
			break;
		case "CURATOR":
			sentencia = "SELECT MAX(CURATOR_ID)+1 FROM varsearch.CURATOR";
			break;
		default:
			break;
		}
		try {
			ps = con.prepareStatement(sentencia);
			rs = ps.executeQuery();
			while(rs.next()){
				nextId = rs.getInt(1);
			}
			rs.close();
			ps.close();
		} catch (SQLException e) {
			e.printStackTrace();
		}
		return nextId;
}

// Método comprobación duplicados Transcript
public boolean existsTranscript(Transcript tanscript){
		Boolean res = null;
		try {
			sentencia = "SELECT * FROM varsearch.TRANSCRIPT WHERE `NM_IDENTIFIER`  = ?";
			ps = con.prepareStatement(sentencia);
			ps.setString(1, tanscript.getNmIdentifier());

			rs = ps.executeQuery();

			if(rs.last())
				res = true;
			else
				res = false;
			rs.close();
			ps.close();
		} catch (SQLException e) {
			e.printStackTrace();
		}
```

```
		return res;
}

//Método de comprobación de DatabankVersion
public boolean existsDatabankVersion(DatabankVersion databankVersion){
		Boolean res = null;
		try {
			sentencia = "SELECT * FROM varsearch.DATABANK_VERSION WHERE `NOMBRE` = ? AND
`FECHA` = ? AND `RELEASE` = ?";
			ps = con.prepareStatement(sentencia);
			ps.setString(1, databankVersion.getDatank().getName());
			ps.setDate(2, (Date) databankVersion.getDate());
			ps.setString(3, databankVersion.getRelease());
			rs = ps.executeQuery();
			if(rs.last())
				res = true;
			else
				res = false;
			rs.close();
			ps.close();
		} catch (SQLException e) {
			e.printStackTrace();
		}
		return res;
}

//Método de comprobación de Variation
public boolean existsVariation(String DB_VARIATION_ID){
		Boolean res = null;
		try {
			sentencia = "SELECT * FROM varsearch.VARIATION WHERE DB_VARIATION_ID = ?";
			ps = con.prepareStatement(sentencia);
			ps.setString(1, DB_VARIATION_ID);
			rs = ps.executeQuery();
			if(rs.last())
				res = true;
			else
				res = false;
			rs.close();
			ps.close();

		} catch (SQLException e) {
			e.printStackTrace();
		}
		return res;
}

//Método de comprobación de BibliographyReference
public boolean existsBibliographyReference(BibliographyReference bibliographyReference){
		Boolean res = null;
		try {
			sentencia = "SELECT * FROM varsearch.BIB_REF WHERE `PUBLICATION` = ? ";
			ps = con.prepareStatement(sentencia);
			ps.setString(1, bibliographyReference.getPublication());
			rs = ps.executeQuery();
```

```
                        if(rs.last())
                                res = true;
                        else
                                res = false;
                        rs.close();
                        ps.close();
                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return res;
}

//Método de comprobación de DatabankVersion
public boolean existsBibliographyDb(String URL){
                Boolean res = null;
                try {
                        sentencia = "SELECT * FROM varsearch.BIBLIOGRAPHY_DB WHERE URL = ?";
                        ps = con.prepareStatement(sentencia);
                        ps.setString(1, URL);
                        rs = ps.executeQuery();
                        if(rs.last())
                                res = true;
                        else
                                res = false;
                        rs.close();
                        ps.close();
                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return res;
}

//Método de comprobación de Curator
public boolean existsCurator(Curator curator){
                Boolean res = null;
                try {
                        sentencia = "SELECT * FROM varsearch.CURATOR WHERE USER_NAME = ?";
                        ps = con.prepareStatement(sentencia);
                        ps.setString(1, curator.getUserName());
                        rs = ps.executeQuery();
                        if(rs.last())
                                res = true;
                        else
                                res = false;
                        rs.close();
                        ps.close();
                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return res;
}

//Método inserción Genome
public boolean insertGenome(Genome genome) {
                Boolean res = null;
                try {
```

```
                        sentencia = "INSERT INTO varsearch.GENOME (`HG_IDENTIFIER`, `GRCH_IDENTIFIER`)
VALUES (?, ?)";
                        ps = con.prepareStatement(sentencia);
                        ps.setString(1, genome.getHgIdentifer());
                        ps.setString(2, genome.getGrchIdentifier());
                        res = true;
                        ps.executeUpdate();
                        ps.close();
                } catch (SQLException e) {
                        res = false;
                }
                return res;
}

//Método inserción Chromosome
public boolean insertChromosome(Chromosome chromosome) {
                Boolean res = null;
                try {
                        sentencia = "INSERT INTO varsearch.CHROMOSOME (`NC_IDENTIFIER`, `NOMBRE`,
`HG_IDENTIFIER`, `SEQUENCE`) VALUES (?, ?, ?, ?)";
                        ps = con.prepareStatement(sentencia);
                        ps.setString(1, chromosome.getNcIdentifier());
                        ps.setString(2, chromosome.getName());
                        ps.setString(3, chromosome.getHgIdentifier());
                        ps.setString(4, null); // <-- Sequence
                        res = true;
                        ps.executeUpdate();
                        ps.close();
                } catch (SQLException e) {
                        res = false;
                }
                return res;
}

//Método inserción ChromosomeElement
public boolean insertChromosomeElement(ChromosomeElement chromosomeElement) {
                Boolean res = null;
                try {

                        sentencia = "INSERT INTO varsearch.CHR_ELEM (`CHR_ELEM_ID`, `NC_IDENTIFIER`,
`START_POSITION`, `END_POSITION`, "
                                        + "`STRAND`, `SPECIALIZATION_TYPE`) VALUES (?, ?, ?, ?, ?, ?)";
                        ps = con.prepareStatement(sentencia);
                        ps.setInt(1, chromosomeElement.getChromosomeElementId()); // <--- CHR_ELEM_ID
                        ps.setString(2, chromosomeElement.getNcIdentifier());
                        ps.setInt(3, chromosomeElement.getStartPosition());
                        ps.setInt(4, chromosomeElement.getEndPosition());
                        ps.setString(5, chromosomeElement.getStrand());
                        ps.setString(6, chromosomeElement.getSpecializationType());
                        res = true;
                        ps.executeUpdate();
                        ps.close();
                } catch (SQLException e) {
                        res = false;
                }
                return res;
```

```
}
//Método inserción Gene
public boolean insertGene(Gene gene) {
		Boolean res = null;
		try {
			sentencia = "INSERT INTO varsearch.GENE (`CHR_ELEM_ID`, `ID_SYMBOL`, `ID_HUGO`, `OFFICIAL_NAME`, `DESCRIPTION`, "
					+ "`BIOTYPE`, `STATUS`, `GC_PERCENTAGE`, `GENE_SYNONYM`, `START_GENENG`, `END_GENENG`) "
					+ "VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)";
			ps = con.prepareStatement(sentencia);
			ps.setInt(1, gene.getChromosomeElement().getChromosomeElementId()); // <--- CHR_ELEM_ID
			ps.setString(2, gene.getIdSymbol());
			ps.setString(3, gene.getIdHugo());
			ps.setString(4, gene.getOfficialName());
			ps.setString(5, gene.getDescription());
			ps.setString(6, gene.getBiotype());
			ps.setString(7, gene.getStatus());
			ps.setDouble(8, gene.getGcPercentaje());
			ps.setString(9, gene.getGeneSynonym());
			ps.setInt(10, gene.getStartGeneNg());
			ps.setInt(11, gene.getEndGeneNg());
			res = true;
			ps.executeUpdate();
			ps.close();
		} catch (SQLException e) {
			res = false;
		}
		return res;
}

//Método inserción SequenceNg
public boolean insertSequenceNg(SequenceNg sequenceNg) {
		Boolean res = null;
		try {
			sentencia = "INSERT INTO varsearch.SEQUENCE_NG (`DNA_SEQUENCE`, `ID_SYMBOL`, `NG_IDENTIFIER`) VALUES (?, ?, ?)";
			ps = con.prepareStatement(sentencia);
			ps.setString(1, sequenceNg.getSequence());
			ps.setString(2, sequenceNg.getGene().getIdSymbol());
			ps.setString(3, sequenceNg.getNgIdentifier());
			res = true;
			ps.executeUpdate();
			ps.close();
		} catch (SQLException e) {
			res = false;
		}
		return res;
}

//Método inserción Transcript
public boolean insertTranscript(Transcript transcript) {
		Boolean res = null;
		try {
			sentencia = "INSERT INTO varsearch.TRANSCRIPT (`CHR_ELEM_ID`, `BIOTYPE`,
```

```
`STARTCDS`, `ENDCDS`, `NM_IDENTIFIER`, "
                                        + "`NG_IDENTIFIER`, `START_TRANSCRIPTNG`,
`END_TRANSCRIPTNG`) VALUES (?, ?, ?, ?, ?, ?, ?, ?)";
                        ps = con.prepareStatement(sentencia);
                        ps.setInt(1, transcript.getChromosomeElement().getChromosomeElementId()); // <---
CHR_ELEM_ID
                        ps.setString(2, transcript.getBiotype());
                        ps.setInt(3, transcript.getStartCds());
                        ps.setInt(4, transcript.getEndCds());
                        ps.setString(5, transcript.getNmIdentifier());
                        ps.setString(6, transcript.getNgIdentifier());
                        ps.setInt(7, transcript.getStartTranscriptNg());
                        ps.setInt(8, transcript.getEndTranscriptNg());
                        res = true;
                        ps.executeUpdate();
                        ps.close();
                } catch (SQLException e) {
                        res = false;
                }
                return res;
}

//Método inserción Protein
public boolean insertProtein(Protein protein) {
                Boolean res = null;
                try {
                        sentencia = "INSERT INTO varsearch.PROTEIN (`CHR_TRANSCRIPT_ID`, `NAME`, "
                                        + "`SEQUENCE`, `SOURCE`, `NP_IDENTIFIER`) VALUES (?, ?, ?, ?,
?)";
                        ps = con.prepareStatement(sentencia);
                        ps.setInt(1,
protein.getTranscript().getChromosomeElement().getChromosomeElementId()); // <--- CHR_TRANSCRIPT_ID
                        ps.setString(2, protein.getName());
                        ps.setString(3, protein.getSequence());
                        ps.setString(4, protein.getSource());
                        ps.setString(5, protein.getNpIdentifier());
                        res = true;
                        ps.executeUpdate();
                        ps.close();
                } catch (SQLException e) {
                        res = false;
                }
                return res;
}

//Método inserción Exon
public boolean insertExon(Exon exon) {
                Boolean res = null;
                try {
                        sentencia = "INSERT INTO varsearch.EXON (`CHR_ELEM_ID`, `NOMBRE`, `ID_SYMBOL`, "
                                        + "`START_EXONNG`, `END_EXONNG`) VALUES (?, ?, ?, ?, ?)";
                        ps = con.prepareStatement(sentencia);
                        ps.setInt(1, exon.getChromosomeElement().getChromosomeElementId()); // <---
CHR_ELEM_ID
                        ps.setString(2, exon.getName());
                        ps.setString(3, exon.getIdSymbol());
```

```
                        ps.setInt(4, exon.getStartExonNg());
                        ps.setInt(5, exon.getEndExonNg());
                        res = true;
                        ps.executeUpdate();
                        ps.close();
                } catch (SQLException e) {
                        System.out.println(ps.toString());
                        res = false;
                }
                return res;
}

//Método inserción ExonTranscript
public boolean insertExonTranscript(ExonTranscript exonTranscript) {
                Boolean res = null;
                try {
                        sentencia = "INSERT INTO varsearch.EXON_TRANSCRIPT (`CHR_EXON_ID`,
`CHR_TRANSCRIPT_ID`, `NOMBRE`) VALUES (?, ?, ?)";
                        ps = con.prepareStatement(sentencia);
                        ps.setInt(1,
exonTranscript.getExon().getChromosomeElement().getChromosomeElementId());
                        ps.setInt(2,
exonTranscript.getTranscript().getChromosomeElement().getChromosomeElementId());
                        ps.setString(3, exonTranscript.getName());
                        res = true;
                        ps.executeUpdate();
                        ps.close();
                } catch (SQLException e) {
                        res = false;
                }
                return res;
}

//Método inserción Databank
public boolean insertDatabank(Databank databank) {
                Boolean res = null;
                try {
                        sentencia = "INSERT INTO varsearch.DATABANK (`NOMBRE`, `DESCRIPTION`, `URL`)
VALUES (?, ?, ?)";
                        ps = con.prepareStatement(sentencia);
                        ps.setString(1, databank.getName());
                        ps.setString(2, databank.getDescription());
                        ps.setString(3, databank.getUrl());
                        res = true;
                        ps.executeUpdate();
                        ps.close();
                } catch (SQLException e) {
                        res = false;
                }
                return res;
}

//Método inserción DatabankVersion
public boolean insertDatabankVersion(DatabankVersion databankVersion) {
                Boolean res = null;
                try {
```

```
                        sentencia = "INSERT INTO varsearch.DATABANK_VERSION (`RELEASE`, `NOMBRE`,
`FECHA`, `DB_VERSION_ID`) VALUES (?, ?, ?, ?)";
                        ps = con.prepareStatement(sentencia);
                        ps.setString(1, databankVersion.getRelease());
                        ps.setString(2, databankVersion.getDatank().getName());
                        ps.setDate(3, (Date) databankVersion.getDate());
                        ps.setInt(4, databankVersion.getDatabankVersionId()); // <--- DB_VERSION_ID
                        res = true;
                        ps.executeUpdate();
                        ps.close();
                } catch (SQLException e) {
                        res = false;
                        System.out.println(ps.toString());
                        e.printStackTrace();
                }
                return res;
}

//Método inserción ElementDatabank
public boolean insertElementDatabank(ElementDatabank elementDatabank){
                Boolean res = null;
                try {
                        sentencia = "INSERT INTO varsearch.ELEMENT_DATABANK (`DB_VERSION_ID`,
`SOURCE_IDENTIFIER`, `CHR_ELEM_ID`) VALUES (?, ?, ?)";
                        ps = con.prepareStatement(sentencia);
                        ps.setInt(1, elementDatabank.getDatabankVersion().getDatabankVersionId()); // <---
DB_VERSION_ID
                        ps.setString(2, elementDatabank.getSourceIdentifier());
                        ps.setInt(3, elementDatabank.getChromosomeElement().getChromosomeElementId()); //
<--- CHR_ELEM_ID
                        res = true;
                        ps.executeUpdate();
                        ps.close();
                } catch (SQLException e) {
                        res = false;
                        System.out.println(ps.toString());
                        e.printStackTrace();
                }
                return res;
}

//Método inserción Variation
public boolean insertVariation(Variation variation) {
                Boolean res = null;
                try {
                        sentencia = "INSERT INTO varsearch.VARIATION (`VARIATION_ID`, `CHR_GENE_ID`,
`DB_VERSION_ID`, `DESCRIPTION`, `DB_VARIATION_ID`, "
                                        + "`CLINICALLY_IMPORTANT`, `PRIVADO`, `NC_IDENTIFIER`,
`NG_IDENTIFIERS`, `OTHER_IDENTIFIERS`, `ASSOCIATED_GENES`, `OMIM`, "
                                        + "`CREATION_VERSION`) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
?)";
                        ps = con.prepareStatement(sentencia);
                        ps.setInt(1, variation.getVariationId()); // <--- VARIATION_ID
                        ps.setInt(2, variation.getGene().getChromosomeElement().getChromosomeElementId());
// <--- CHR_GENE_ID
                        ps.setInt(3, variation.getDatabankVersion().getDatabankVersionId()); // <---
```

```
DB_VERSION_ID
                        ps.setString(4, variation.getDescription());
                        ps.setString(5, String.valueOf(variation.getDbVariationId()));
                        ps.setString(6, variation.getClinicallyImportant());
                        ps.setInt(7, variation.getPrivado()); // <-- PRIVATE
                        ps.setString(8, variation.getNcIdentifier());
                        ps.setString(9, variation.getNgIdentifiers());
                        ps.setString(10, variation.getOtherIdentifiers());
                        ps.setString(11, variation.getGene().getIdSymbol());
                        ps.setString(12, variation.getOminReference());
                        ps.setString(13, null); // <--- CREATION_VERSION
                        res = true;
                        ps.executeUpdate();
                        ps.close();
                } catch (SQLException e) {
                        res = false;
                        System.out.println(ps.toString());
                        e.printStackTrace();
                }
                return res;
}

//Método inserción Precise
public boolean insertPrecise(Precise precise) {
                Boolean res = null;
                try {
                        sentencia = "INSERT INTO varsearch.PRECISE (`VARIATION_ID`, `SPECIALIZATION_TYPE`,
`INS_SEQUENCE`, `INS_REPETITION`, `NUM_BASES`, "
                                        + "`FLANKING_RIGHT`, `FLANKING_LEFT`, `ALN_QUALITY`,
`POSITION`) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)";
                        ps = con.prepareStatement(sentencia);
                        ps.setInt(1, precise.getVariation().getVariationId()); // <--- VARIATION_ID
                        ps.setString(2, precise.getSpecializationType());
                        ps.setString(3, precise.getInsSequence());
                        ps.setInt(4, precise.getInsRepetition());
                        ps.setInt(5, precise.getNumBases());
                        ps.setString(6, precise.getFlankingRight());
                        ps.setString(7, precise.getFlankingLeft());
                        ps.setInt(8, precise.getAlnQuality());
                        ps.setInt(9, precise.getPosition());
                        res = true;
                        ps.executeUpdate();
                        ps.close();
                } catch (SQLException e) {
                        res = false;
                        e.printStackTrace();
                }
                return res;
}

//Método inserción PreciseSeqNg
public boolean insertPreciseSeqNg(PreciseSeqNg preciseSeqNg) {
                Boolean res = null;
                try {
                        sentencia = "INSERT INTO varsearch.PRECISE_SEQNG (`VARIATION_ID`, `NG_IDENTIFIER`,
`POSITION`, `FLANKING_LEFT`, `FLANKING_RIGHT`) "
```

```
                                        + "VALUES (?, ?, ?, ?, ?)";
                        ps = con.prepareStatement(sentencia);
                        ps.setInt(1, preciseSeqNg.getPrecise().getVariation().getVariationId()); // <---
VARIATION_ID
                        ps.setString(2, preciseSeqNg.getNgIdentifier());
                        ps.setInt(3, preciseSeqNg.getPosition());
                        ps.setString(4, preciseSeqNg.getPrecise().getFlankingLeft());
                        ps.setString(5,preciseSeqNg.getPrecise().getFlankingRight())
                        res = true;
                        ps.executeUpdate();
                        ps.close();
                } catch (SQLException e) {
                        res = false;
                        System.out.println(ps.toString());
                }
                return res;
}
```

//Método inserción BibliographyReference

```
public boolean insertBibliographyReference(BibliographyReference bibliographyReference) {
                Boolean res = null;
                try {
                        sentencia = "INSERT INTO varsearch.BIB_REF (`BIB_REF_ID`, `TITLE`, `ABSTRACT`,
`PUBLICATION`, "
                                        + "`AUTHORS`, `DATE_PUB`) VALUES (?, ?, ?, ?, ?, ?)";
                        ps = con.prepareStatement(sentencia);
                        ps.setInt(1, bibliographyReference.getBibliographyReferenceId()); // <--- BIB_REF_ID
                        ps.setString(2, bibliographyReference.getTitle());
                        ps.setString(3, bibliographyReference.getAbstract());
                        ps.setString(4, bibliographyReference.getPublication());
                        ps.setString(5, bibliographyReference.getAuthors());
                        ps.setDate(6, (Date) bibliographyReference.getDatePub());
                        res = true;
                        ps.executeUpdate();
                        ps.close();
                } catch (SQLException e) {
                        res = false;
                }
                return res;
}
```

//Método inserción BibliographyDb

```
public boolean insertBibliographyDb(BibliographyDb bibliographyDb) {
                Boolean res = null;
                try {
                        sentencia = "INSERT INTO varsearch.BIBLIOGRAPHY_DB (`URL`, `BIB_REF_ID`, `NAME_DB`,
`PUBMED_ID`) VALUES (?, ?, ?, ?)";
                        ps = con.prepareStatement(sentencia);
                        ps.setString(1, bibliographyDb.getUrl());
                        ps.setInt(2, bibliographyDb.getBibliographyReference().getBibliographyReferenceId()); //
<--- BIB_REF_ID
                        ps.setString(3, bibliographyDb.getNameDb());
                        ps.setInt(4, bibliographyDb.getPubmedId());
                        res = true;
                        ps.executeUpdate();
                        ps.close();
```

```
		} catch (SQLException e) {
			res = false;
		}
		return res;
}

//Método inserción ReferenceVariation
public boolean insertReferenceVariation(ReferenceVariation referenceVariation) {
		Boolean res = null;
		try {
			sentencia = "INSERT INTO varsearch.REFERENCE_VARIATION (`VARIATION_ID`,
`BIB_REF_ID`) VALUES (?, ?)";
			ps = con.prepareStatement(sentencia);
			ps.setInt(1, referenceVariation.getVariation().getVariationId()); // <--- VARIATION_ID
			ps.setInt(2, referenceVariation.getBibliographyReference().getBibliographyReferenceId());
// <--- BIB_REF_ID
			res = true;
			ps.executeUpdate();
			ps.close();
		} catch (SQLException e) {
			e.printStackTrace();
			res = false;
		}
		return res;
}


//Método inserción ReferenceChromosomeElement
public boolean insertReferenceChromosomeElement(ReferenceChromosomeElement
referenceChromosomeElement) {
		Boolean res = null;
		try {
			sentencia = "INSERT INTO varsearch.REF_CHR_ELEM (`CHR_ELEM_ID`, `BIB_REF_ID`)
VALUES (?, ?)";
			ps = con.prepareStatement(sentencia);
			ps.setInt(1,
referenceChromosomeElement.getChromosomeElement().getChromosomeElementId()); // <--- CHR_ELEM_ID
			ps.setInt(2,
referenceChromosomeElement.getBibliographyReference().getBibliographyReferenceId()); // <--- BIB_REF_ID
			res = true;
			ps.executeUpdate();
			ps.close();
		} catch (SQLException e) {
			res = false;
		}
		return res;
}

//Método inserción Validation
public boolean insertValidation(Validation validation){
		Boolean res = null;
		try {
			sentencia = "INSERT INTO varsearch.VALIDATION (`VARIATION_ID`, `CURATOR_ID`,
`COMENTARIO`, `PRIVADO`, `VALIDATION_DATE`, "
					+ "`INTERNAL_CODE`, `CLINICAL_SIGNIFICANCE`) VALUES (?, ?, ?,
?, ?, ?, ?)";
```

```
                        ps = con.prepareStatement(sentencia);
                        ps.setInt(1, validation.getVariation().getVariationId()); // <--- VARIATION_ID
                        ps.setInt(2, validation.getCurator().getCuratorId()); // <--- CURATOR_ID
                        ps.setString(3, validation.getComentario());
                        ps.setInt(4, validation.getPrivado());
                        ps.setDate(5, (Date) validation.getValidationDate());
                        ps.setString(6, validation.getInternalCode());
                        ps.setString(7, validation.getClinicalSignificance());
                        res = true;
                        ps.executeUpdate();
                        ps.close();
                } catch (SQLException e) {
                        res = false;
                }
                return res;
}

//Método inserción Curator
public boolean insertCurator(Curator curator) {
                Boolean res = null;
                try {
                        sentencia = "INSERT INTO varsearch.CURATOR (`CURATOR_ID`, `USER_NAME`, `PASS`)
VALUES (?, ?, ?)";
                        ps = con.prepareStatement(sentencia);
                        ps.setInt(1, curator.getCuratorId());
                        ps.setString(2, curator.getUserName());
                        ps.setString(3, curator.getPassword());
                        res = true;
                        ps.executeUpdate();
                        ps.close();
                } catch (SQLException e) {
                        res = false;
                }
                return res;
}

//Método consulta Curator
public Curator getCurator(int id){
                Curator curator = new Curator();
                try {
                        sentencia = "SELECT * FROM varsearch.CURATOR WHERE `CURATOR_ID` = ?";
                        ps = con.prepareStatement(sentencia);
                        ps.setInt(1, id);
                        rs = ps.executeQuery();
                        while(rs.next()){
                                curator.setCuratorId(rs.getInt("CURATOR_ID"));
                                curator.setUserName(rs.getString("USER_NAME"));
                                curator.setPassword(rs.getString("PASS"));
                        }
                        rs.close();
                        ps.close();

                } catch (SQLException e) {
                        e.printStackTrace();
                }
                return curator;
```

```
}

//Método consulta Curators
public ArrayList<Curator> getCurators(){
		Curator curator = new Curator();
		ArrayList<Curator> curators = new ArrayList<Curator>();
		try {
			sentencia = "SELECT * FROM varsearch.CURATOR";
			ps = con.prepareStatement(sentencia);
			rs = ps.executeQuery();

			while(rs.next()){
				curator = new Curator();
				curator.setCuratorId(rs.getInt("CURATOR_ID"));
				curator.setUserName(rs.getString("USER_NAME"));
				curator.setPassword(rs.getString("PASS"));

				curators.add(curator);
			}
			rs.close();
			ps.close();
		} catch (SQLException e) {
			e.printStackTrace();
		}
		return curators;
}

//Método inserción Curator
public boolean deleteCurator(int id){
		try {
			sentencia = "DELETE FROM varsearch.CURATOR WHERE `CURATOR_ID` = ?";
			ps = con.prepareStatement(sentencia);
			ps.setInt(1, id);
			ps.executeUpdate();
			ps.close();
			return true;
		} catch (SQLException e) {
			return false;
		}
}

//Método actualización Curator
public void updateCurator(Curator curator){
		try {
			sentencia = "UPDATE varsearch.CURATOR SET `USER_NAME` = ?, `PASS`= ? WHERE
`CURATOR_ID` = ?";
			ps = con.prepareStatement(sentencia);
			ps.setString(1, curator.getUserName());
			ps.setString(2, curator.getPassword());
			ps.setInt(3, curator.getCuratorId());
			ps.executeUpdate();
			ps.close();
		} catch (SQLException e) {
			e.printStackTrace();
		}
}
```

XMLClinvarHandler.java

```
public Variation getVariation(){
		precise.setPreciseSeqNg(preciseSeqNg);
		variation.setPrecise(precise);
		variation.setBibliographyIds(bibliographyIdList);
		return variation;
}

@Override
public void characters(char[] ch, int start, int length) throws SAXException {
		buffer.append(ch, start, length);
}

@Override
public void endElement(String uri, String localName, String qName) throws SAXException {
		switch(qName.toString()){
		//Creación lista de Pubmed_Ids para su posterior extracción
		case "ID":
			if(!idSource.isEmpty() && idSource.equals("PubMed"))
				if(!bibliographyIdList.contains(Integer.parseInt(buffer.toString())))
					bibliographyIdList.add(Integer.parseInt(buffer.toString()));
			break;

		//Obtención y definición SpecializationType
		case "VariantType":
			specialization = buffer.toString();
			if(specialization.equals("Deletion"))
				precise.setSpecializationType("Deletion");
			if(specialization.equals("single nucleotide variant"))
				precise.setSpecializationType("Indel");
			if(specialization.equals("Copy Number Loss"))
				precise.setSpecializationType("Deletion");
			if(specialization.equals("Copy Number Chain") || specialization.equals("Duplication") ||
					specialization.equals("Tandem Duplication") ||
specialization.equals("Microsatellite"))
				precise.setSpecializationType("Insertion");
			break;
		case "ClinicalSignificance":
			aux++;
			break;
		//Obtención ClinicallyImportant
		case "Description":
			if(aux == 0){
				variation.setClinicallyImportant(buffer.toString());
				aux++;
			}
			break;

		//Obtención OtherIdentifier
		case "HGVS":
			variation.setOtherIdentifiers(buffer.toString());
			break;

		//Obtención OminReference
		case "OMIM":
			variation.setOminReference(buffer.toString());
```

```
                        break;
                default:
                        break;
                }
}

@Override
public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {
                switch(qName.toString()){
                //Obtención Pubmed_Id
                case "ID":
                        idSource = attributes.getValue("Source");
                        buffer.setLength(0);
                        break;
                //Obtención Phenotype
                case "Phenotype":
                        if(posPheno==0){
                                variation.setPhenotypeName(attributes.getValue("Name"));
                                posPheno++;
                        }
                        buffer.setLength(0);
                        break;
                //Obtención OMIM
                case "XRef":
                        posDB = attributes.getValue("DB");
                        if(posOMIM == 0 && posDB.equals("OMIM")){
                                variation.setOminReference(attributes.getValue("ID"));
                                posOMIM++;
                        }
                        buffer.setLength(0);
                        break;
                //Obtención GrchIdentifier, Position, ChrStartPosition, ChrEndPosition y valores para el
cálculo de InsSequence, InsRepetition y NumBases
                case "SequenceLocation":
                        status = attributes.getValue("AssemblyStatus");
                        if(status.equals("current")){
                                variantLength = Integer.parseInt(attributes.getValue("variantLength"));
                                variation.setGrchIdentifier(attributes.getValue("Assembly"));
                                precise.setPosition(Integer.parseInt(attributes.getValue("start")));
                                variation.setChrStartPosition(Integer.parseInt(attributes.getValue("start")));
                                variation.setChrEndPosition(Integer.parseInt(attributes.getValue("stop")));
                                alternateAllele = attributes.getValue("alternateAllele");
                                referenceAllele = attributes.getValue("referenceAllele");
                        }
                        buffer.setLength(0);
                        break;

                //Obtención Description
                case "VariationReport":
                        String variationName = attributes.getValue("VariationName");
                        variation.setDescription(variationName);
                        String nmReference[] = variationName.split("\\(");
                        variation.setNmIdentifier(nmReference[0]);
                        buffer.setLength(0);
                        break;
```

```
//Obtención Gene_Id y Strand
case "Gene":
		if(posGen == 0){
			variation.setGeneId(Integer.parseInt(attributes.getValue("GeneID")));
			posGen++;
		}
		if(attributes.getValue("strand").equals("+"))
			variation.setStrand("P");
		else if(attributes.getValue("strand").equals("-"))
			variation.setStrand("M");

		buffer.setLength(0);
		break;

case "HGVS":
		accessionVersion = attributes.getValue("AccessionVersion");
		//Obtención NpIdentifier
		if(posNP == 0){
			if(accessionVersion.contains("NP_")){
				variation.setNpIdentifier(accessionVersion);
				posNP++;
			}
		}
		if(posNC == 0){
			//Obtención NcIdentifier
			if(attributes.getValue("AccessionVersion").contains("NC_")){
				variation.setNcIdentifier(accessionVersion);
				posNC++;
			}
		}
		if(posNG == 0){
			if(accessionVersion.contains("NG_")){
				change = attributes.getValue("Change");
				//Obtención NgIdentifiers
				ngIdentifiers =
attributes.getValue("AccessionVersion")+":"+change;
				//Obtención NgIdentifier
				variation.setNgIdentifiers(ngIdentifiers);

preciseSeqNg.setNgIdentifier(attributes.getValue("AccessionVersion"));

				//Obtención InsSequence, InsRepetition y NumBases en
caso de SNPs
				if(change.contains(">") && specialization.equals("single
nucleotide variant")){

preciseSeqNg.setPosition(Integer.parseInt(change.substring(2,change.length()-3)));
					precise.setInsSequence(alternateAllele);
					precise.setInsRepetition(0);
					precise.setNumBases(0);
				}
				//Obtención InsSequence, InsRepetition y NumBases en
caso de Duplicaciones
				if(change.contains("dup") &&
specialization.equals("Duplication")){
					if(change.contains("_")){
```

```
                                                String parts[] = change.split("_");

            preciseSeqNg.setPosition(Integer.parseInt(parts[0].substring(2)));
                                                }
                                                if(!change.contains("_")){
                                                String split1[] = change.split(":");
                                                String split2[] = split1[0].split("dup");

            preciseSeqNg.setPosition(Integer.parseInt(split2[0].substring(2,split2[0].length())));
                                                }
                                                precise.setInsSequence(referenceAllele);

            precise.setInsRepetition(alternateAllele.length()/variantLength);
                                                precise.setNumBases(alternateAllele.length()-
variantLength);
                                    }
                                    //Obtención InsSequence, InsRepetition y NumBases en
caso de Deleciones
                                    if(change.contains("del") && specialization.equals("Deletion")){

                                                if(change.contains("_")){
                                                String parts[] = change.split("_");

            preciseSeqNg.setPosition(Integer.parseInt(parts[0].substring(2)));
                                                }
                                                if(!change.contains("_")){
                                                String split1[] = change.split(":");
                                                String split2[] = split1[0].split("del");

            preciseSeqNg.setPosition(Integer.parseInt(split2[0].substring(2,split2[0].length())));
                                                }
                                                precise.setInsSequence("-");
                                                precise.setInsRepetition(0);
                                                precise.setNumBases((alternateAllele.length()-1)-
variantLength);
                                    }
                        }
                        posNG++;
            }
            buffer.setLength(0);
            break;

      default:
            buffer.setLength(0);
            break;
      }
{
```

XMLGeneHandler.java

```
public Gene getGene(){
		return gene;
}

@Override
public void characters(char[] ch, int start, int length) throws SAXException {
		buffer.append(ch, start, length);
}

@Override
public void endElement(String uri, String localName, String qName) throws SAXException {
		switch(qName.toString()){
		case "Entrezgene_type":
			switch(buffer.toString()){
			//Obtención Biotype
			case "0":
				gene.setBiotype("unknow");
				break;
			case "1":
				gene.setBiotype("tRNA");
				break;
			case "2":
				gene.setBiotype("rRNA");
				break;
			case "3":
				gene.setBiotype("snRNA");
				break;
			case "4":
				gene.setBiotype("scRNA");
				break;
			case "5":
				gene.setBiotype("snRNA");
				break;
			case "6":
				gene.setBiotype("protein_coding");
				break;
			case "7":
				gene.setBiotype("pseudo");
				break;
			case "8":
				gene.setBiotype("transposon");
				break;
			case "9":
				gene.setBiotype("miscRNA");
				break;
			case "10":
				gene.setBiotype("ncRNA");
				break;
			case "11":
				gene.setBiotype("biological_region");
				break;
			case "12":
				gene.setBiotype("other");
				break;
			default:
```

```
                    break;
                }
            //Obtención Id_Symbol
            case "Gene-ref_locus":
                    gene.setIdSymbol(buffer.toString());
                    break;
            //Obtención OffcialName
            case "Gene-ref_desc":
                    gene.setOfficialName(buffer.toString());
                    break;
            //Obtención Id_Hugo
            case "Object-id_str":
                    gene.setIdHugo(buffer.toString());
                    break;
            //Obtención Gene Synonym
            case "Gene-ref_syn_E":
                    gene.setGeneSynonym(buffer.toString());
                    break;
            //Obtención Decription
            case "Entrezgene_summary":
                    gene.setDescription(buffer.toString());
                    break;
            //Obtención Id_Symbol
            case "Gene-commentary_accession":
                    if(buffer.toString().contains("NG_"))
                            posNG++;
                    break;
            //Obtención Start/end_GeneNg
            case "Seq-interval_from":
                    if(posNG == 1)
                            gene.setStartGeneNg(Integer.parseInt(buffer.toString())+1);
                    break;

            case "Seq-interval_to":
                    if(posNG == 1){
                            gene.setEndGeneNg(Integer.parseInt(buffer.toString())+1);
                            posNG++;
                    }
                    break;

            //Obtención Status
            case "Gene-commentary_label":
                    switch(buffer.toString()){
                    case "MODEL":
                            gene.setStatus("Model");
                            break;
                    case "INFERRED":
                            gene.setStatus("Inferred");
                            break;
                    case "PREDICTED":
                            gene.setStatus("Predicted");
                            break;
                    case "PROVISIONAL":
                            gene.setStatus("Provisional");
                            break;
```

```
                        case "REVIEWED":
                                gene.setStatus("Reviewed");
                                break;
                        case "VALIDATED":
                                gene.setStatus("Validated");
                                break;
                        case "WGS":
                                gene.setStatus("WGS");
                                break;
                        default:
                                break;
                        }
                }
}
```

XMLNucNgHandler.java

```
public Transcript getTranscript(){

        //Transformación de datos para la obtención de Start/end_Cds
        String cdsString = "";
        String arrayCDS[] = new String[]{""};
        if(locationCDS.contains("complement") && locationCDS.contains("join"))
                cdsString = locationCDS.substring(16, locationCDS.length()-2);
        else if(locationCDS.contains("complement"))
                // <--- complement(join(      ))
                cdsString = locationCDS.substring(12, locationCDS.length()-1);
        else if(locationCDS.contains("join"))
                // <--- "join(5001..5323")
                cdsString = locationCDS.substring(5, locationCDS.length()-1);
        else
                cdsString = locationCDS;
        // <--- "<919..2830, "
        if(!cdsString.contains(",")){
                arrayCDS[0] = cdsString;
        }
        else
                arrayCDS = cdsString.split(",");
        for(int i=0; i<arrayCDS.length;i++){
                if(i==0){
                        String startEndCDS[] = arrayCDS[i].split("\\..");
                        if(startEndCDS[0].matches("[0-9]+"))
                                transcript.setStartCds(Integer.parseInt(startEndCDS[0]));
                        else

        transcript.setStartCds(Integer.parseInt(startEndCDS[0].substring(1)));
                }
                if(i == arrayCDS.length-1){
                        String startEndCDS[] = arrayCDS[i].split("\\..");
                        transcript.setEndCds(Integer.parseInt(startEndCDS[1]));
                }
        }
        transcript.setStartTranscriptNg(listExon.get(0).getStartExonNg());
        transcript.setEndTranscriptNg(listExon.get(listExon.size()-1).getEndExonNg());
        transcript.setExons(listExon);
```

```
                sequenceNG.setTranscript(transcript);
                transcript.setSequenceNg(sequenceNG);
                transcript.setProteinsId(proteinListID);
                return transcript;
}


@Override
public void characters(char[] ch, int start, int length) throws SAXException {
                buffer.append(ch, start, length);
}

@Override
public void endElement(String uri, String localName, String qName) throws SAXException {
                switch(qName.toString()){
                //Obtención SourceIdentifier
                case "GBSeqid":
                                if(buffer.toString().contains("gi"))
                                                transcript.setSourceIdentifier(buffer.toString().substring(3));
                                break;

                //Obtención secuencia sequenceNG
                case "GBSeq_sequence":
                                sequenceNG.setSequence(buffer.toString());
                                break;

                //Creación de exones
                case "GBFeature_key":
                                if(buffer.toString().equals("exon")){
                                                isExon = true;
                                                exon = new Exon();
                                }
                                if(buffer.toString().equals("CDS"))
                                                isCDS++;
                                break;

                //ObtenciónStartExonNg
                case "GBInterval_from" :
                                if(isExon){
                                                exon.setStartExonNg(Integer.parseInt(buffer.toString()));
                                }
                                break;

                //ObtenciónsetEndExonNg
                case "GBInterval_to" :
                                if(isExon){
                                                exon.setEndExonNg(Integer.parseInt(buffer.toString()));
                                }
                                break;

                //Captura información CDS para su posterior transformación
                case "GBFeature_location":
                                if(isCDS == 1){
                                                locationCDS = buffer.toString();
                                                isCDS++;
                                }
```

```
                        break;

                case "GBQualifier_name":
                        if(buffer.toString().equals("protein_id"))
                                isProtein = true;
                        break;

                case "GBQualifier_value":
                        if(isExon && countGBQualifier != 4)
                                countGBQualifier++;

                        if(isExon && countGBQualifier == 4){
                                exon.setName(buffer.toString());
                                listExon.add(exon);
                                countGBQualifier = 0;
                                isExon = false;
                        }

                        //ObtenciónProtein_Ids
                        if(isProtein){
                                proteinListID.add(buffer.toString());
                                isProtein = false;
                        }
                        break;

                default:
                        break;
                }
}
```

XMLProteinHandler.java

```
//Creación proteína
public Protein getProtein(){
                protein.setName(name);
                protein.setNpIdentifier(npIdentifier);
                protein.setSource("NCBI");
                protein.setSequence(sequence);
                return protein;
}

@Override
public void characters(char[] ch, int start, int length) throws SAXException {
                buffer.append(ch, start, length);
}

@Override
public void endElement(String uri, String localName, String qName) throws SAXException {
                switch(qName.toString()){
                //Obtención del nombre
                case "GBSeq_definition":
                        name = buffer.toString();
                        break;
                        //Obtención de npIdentifier
```

```
                case "GBSeq_accession-version":
                        npIdentifier = buffer.toString();
                        break;
                        //Obtención secuencia proteina
                case "GBSeq_sequence":
                        sequence = buffer.toString();
                        break;

                default:
                        break;
                }
}

@Override
public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {

                buffer.setLength(0);

}
```

XMLSnpHandler.java

```
//Creación variación parcial
public Variation getVariation(){
                precise.setPreciseSeqNg(preciseSeqNg);
                variation.setPrecise(precise);
                return variation;
}

@Override
public void characters(char[] ch, int start, int length) throws SAXException {
                buffer.append(ch, start, length);
}

@Override
public void endElement(String uri, String localName, String qName) throws SAXException {

                switch(qName.toString()){
                //Obtención ClinicallyImportant
                case "ClinicalSignificance":
                        variation.setClinicallyImportant(buffer.toString());
                        break;
                //Obtención FlankingLeft
                case "Seq5":
                        if(flankingLeft == 0){
                                precise.setFlankingLeft(buffer.toString().substring(buffer.toString().length()-
25, buffer.toString().length()));
                                flankingLeft++;
                        }
                        break;
                //Obtención FlankingRight
                case "Seq3":
                        if(flankingRight == 0){
                                precise.setFlankingRight(buffer.toString().substring(0, 25));
                                flankingRight++;
                        }
```

```
                        break;
                default:
                        break;
                }
}

@Override
public void startElement(String uri, String localName, String qName, Attributes attributes) throws SAXException {

                switch(qName.toString()){

                //Obtención DB_Variation_Id
                case "Rs":
                        precise.setVariationId(Integer.parseInt(attributes.getValue("rsId")));
                        buffer.setLength(0);
                        break;

                //Obtención geneId
                case "FxnSet":
                        variation.setGeneId(Integer.parseInt(attributes.getValue("geneId")));
                        buffer.setLength(0);
                        break;

                case "PrimarySequence":
                        if(attributes.getValue("accession").contains("NG_")){
                                primaryPos = attributes.getValue("accession");
                        }
                        buffer.setLength(0);
                        break;
                //Obtención alnQuality
                case "MapLoc":
                        if(primaryPos.contains("NG_")){
                                precise.setAlnQuality(Integer.parseInt(attributes.getValue("alnQuality")));
                        }
                        primaryPos = "";
                        buffer.setLength(0);
                        break;

                default:
                        buffer.setLength(0);
                        break;
                }
}
```