Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Departamento de Computación

Recuperación de Información en Bases de Datos de tipo Bioinformático

Tesis que presenta

Elizabeth Luna Luz

para obtener el grado de

Maestra en Ciencias en

la Especialidad de Ingeniería Eléctrica

Director de Tesis

Dr. Guillermo Morales Luna

México, D.F. Febrero 2008

Agradecimientos

A mi madre por ser tan cariñosa y comprensiva en todo momento, a mi padre por darme la fortaleza que siempre necesité, a mis hermanos por darme muchos ánimos, pero sobre todo a Dios por darme una familia tan maravillosa que siempre me ha apoyado en todo momento. Gracias porque una vez más lo hemos logrado juntos.

Agradezco al Dr. Guillermo Morales Luna por ser mi director de tesis, su gran paciencia y por todo el apoyo que me ha brindado, a los doctores Sergio Víctor Chapa Vergara y Julio Isael Pérez Carreón por apoyarme y ser mis sinodales. A Sofy quien siempre estuvo en todo momento para ayudarme.

También quiero agradecer a Ricardo por apoyarme en esas arduas desveladas, a Agustín quien ha sido mi amigo durante muchos años y a esos momentos de distracción juntos, a Amilcar por sus buenos consejos, a mis amigos de la maestría con quienes he compartido momentos inolvidables: Juan, Fabiola, Carlos, Victor y Eduardo.

Agradezco al Departamento de Biología Celular del Cinvestav-IPN, al laboratorio del Dr.Saúl Villa Treviño, en especial al Dr.Julio Isael Pérez Carreón por su gran participación en el planteamiento del problema y su apoyo.

Por otra parte, agradezco a todos mis profesores del Departamento de Computación por transmitirme sus valiosos conocimientos.

Finalmente, agradezco al Departamento de Computación del Cinvestav-IPN, que ha sido mi segunda casa durante mi maestría, al personal que en él labora, y al CONACyT y COMECyT por su financiamiento en mis estudios.

Resúmen

En la actualidad diversas instituciones tienen la necesidad de realizar la gestión de su información. La gran cantidad de datos generados provoca la dificultad en su manipulación, por lo que es relavante la formulación de aplicaciones que permitan la recuperación de datos. El Departamento de Bíologia Celular del Cinvestav-IPN no es ajeno a esta problemática. Por ejemplo, el laboratorio del Dr. Saúl Villa Treviño ha generado abundante información concerniente a la genómica del cáncer.

Este documento aborda la problemática que enfrenta el Departamento de Bíologia Celular al tratar de manipular la gran cantidad de información recabada para sus investigaciones, la cual proviene de diversas bases biológicas. Por lo que se plantea como solución la creación de un sistema de información capaz de analizar, manipular y administrar dicha información con la finalidad de agilizar las investigaciones.

Este proyecto trata entonces de un sistema recuperador de información dotado de reglas heurísticas de búsqueda de intereses y aplicaciones propios para datos generados en Biología Celular, mediante el uso de formas de almacenamiento y lenguajes de consulta estándares, así como de herramientas de sofware libre para la elaboración de dicho recuperador. Así la presente tesis constituye el esfuerzo interdiciplinario generado entre la interacción del Departamento de Computación y el de Biología Celular del Cinvestav. Aporta una aplicación en beneficio de la investigación.

Abstract

Nowadays, several institutions have needs for the management of their information. The huge volume of data produced has resulted difficult to be handling. The Departamento de Biología Celular del Cinvestav-IPN is no stranger to this problem.

This work addresses that problem focused in the Departamento de Biología Celular where it is trying to manipulate a vast amount of information collected in its research, which comes from various biological sources. This thesis sets as a solution the creation of an information system which can analyze, manipulate and manage information in order to help and improve research activity.

This project involves an Information Retrieval System (IRS) based on heuristics rules and searching of own interest and applications for Biología Celular data, using standards types of storage and query languages, as well as open software tools for the development of this IRS.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Con los avances de la tecnología se ha dado un gran crecimiento en el descubrimiento de nueva información en los diferentes campos de la ciencia, dicha generación de información, automatizada o no, nos ha encaminado a encontrar a la par formas para almacenarla de manera organizada. Día a día miles de personas alrededor del mundo generan información, la almacenan, la administran, la analizan y la manipulan para nuevamente generar más información.

El Departamento de Biologia Celular del Cinvestav-IPN al que nos referiremos simplemente como "Biología Celular", está conformado por un núcleo de investigadores, los cuales tienen la función de producir material humano de alto nivel para la investigacón, la educación y la industria.

Este documento nos muestra como caso de estudio el laboratorio del Dr. Saúl Villa Treviño del Departamento de Biologia Celular de Cinvestav-IPN; el cual está generando un gran volúmen de información y requiere herramientas para manipularla. Este laboratorio no es el único que enfrenta éste problema. La presente tesis plantea como solución la creación de un sistema de información capaz de analizar, manipular y administrar toda la información recabada con el objetivo de agilizar las investigaciones.

1.1. Motivación y antecedentes

Este proyecto surge a partir del interés en la búsqueda de sistemas de integración de información biológica de manera automatizada, en particular para el Departamento de Biología Celular del CINVESTAV-IPN, el cual maneja grandes cantidades de información para el desarrollo de sus investigaciones.

Ahora bien, las bases de datos bioinformáticas, como por ejemplo v. gr. `GenBank`, `SwissProt` y `Medline` las cuales son algunas de las bases consultadas por el Departamento, contienen cada una de ellas una inmensa cantidad de información generalmente ajustada a estándares de la comunidad donde cada base contiene enlaces entre ellas y entre los diversos tópicos contenidos en ellas. Sin embargo aún es vigente el problema de buscar información eficientemente. Si bien cada una de las bases cuenta con sus propios manejadores y existe una comunicación muy variada entre ellas, el

analista que las consulta ha de conocer en profundidad los alcances, los recursos y los lenguajes de consulta utilizados por las bases de datos.

Es importante entonces disponer de mecanismos de recuperación de la información, tanto de tipo descriptiva como predictiva, con capacidades para manipular diversos tipos de datos. Se ha entonces de tratar el problema de localizar patrones de interés, que aparezcan con frecuencias determinadas por el usuario en las bases de datos. Entre los algoritmos de búsqueda están *primero a lo profundo*, o *búsqueda aleatoria*. Inclusive, puede buscarse patrones que aparezcan un cierto mínimo de veces en un contexto y uno máximo en otro (por ejemplo cuando se quiere distinguir genes que se expresan de manera diferenciada) o bien que aparezcan en contextos determinados. Bases de datos en las que se tiene un recuento de los contextos se dicen ser *inductivas* y en ellas el problema de recuparción de la información es más especializado. En el caso relacional, la *subsunción-θ* es muy importante. En las estructuras donde vale, las consultas pueden jerarquizarse precisamente por la noción de *subsunción* (la respuesta buscada está en la de otra consulta ya formulada). Para esto se ha de considerar operadores de *espacialización* y *generalización*. La información puede ser provista de manera *estructurada* (en tablas relacionales, por ejemplo), *semiestructurada* (en XML, por ejemplo) o *no-estructurada* (en `Medline`, por ejemplo). Ha habido varios sistemas de éstos, por ejemplo `DiscoveryNet` [1] implementa métodos de integración de datos y de minería de textos utilizando diversas bases de datos bioinformáticas. Por otro lado, `WordNet` es una base de datos de tipo lingüístico sobre la cual se ha definido diversas funciones para estimar similitudes conceptuales [2].

Veamos algunos métodos específicos para la recolección de información. CAC (*Correlate the Annotations Components*) [3] evalúa la relevancia de la información de parejas (productos-de-genes, propiedades-biológicas) en base a la correlación entre estructuras y funciones biológicas. Sea $\mathcal{G}$ un conjunto de *productos de genes* y sea $\mathcal{P}$ uno de *productos de propiedades biológicas*, dotados de sendas distancias $D_{\mathcal{G}}$ y $D_{\mathcal{P}}$, determinadas, por ejemplo, de acuerdo con BLAST (*Basic Local Alignment Search Tool*) [4]. Una *anotación* es una pareja $(g,p) \in \mathcal{A} \subset \mathcal{G}\times\mathcal{P}$, y una *medida de semejanza* en el conjunto de anotaciones $\mathcal{A}$ es

$$A : \mathcal{A}\times\mathcal{A} \to \mathbb{R} \quad , \quad A : ((g_0,p_0),(g_1,p_1)) \mapsto A\left((g_0,p_0),(g_1,p_1)\right) = \frac{D_{\mathcal{G}}(g_0,g_1)}{D_{\mathcal{P}}(p_0,p_1)},$$

donde se ha de interpretar $\frac{0}{x} = 0$, cualquiera que sea x, y $\frac{x}{0} = \infty$, siempre que $x \neq 0$.

Dado un umbral $y > 0$, dos anotaciones a_0, a_1 se dicen ser *y-semejantes* si $A(a_0,a_1) \geq y$. Y para una anotación a_0, su *índice de correlación* es $I_y(a_0) = \mathrm{card}\{a_1 \in \mathcal{A} |\, A(a_0,a_1) \geq y\}$. Estas son nociones presentadas en [5].

La manera en la que se utilizan estos conceptos es como sigue: Se tiene una base de datos *Carbohydrate Active enZYmes* (`CAZy`) que relaciona cierto tipo de enzimas con sus estructuras y otra *Gene Ontology* (`GO`) que relaciona términos lingüísticos de enzimas con las propiedades funcionales de ellas. Ambas bases provenientes de diversas bases de datos bioinformáticas. Se trata entonces de relacionar los registros de `CAZy` con los de `GO`. Hay que formar las anotaciones y luego comparar semejanzas.

Luego de distinguir conceptos semejantes, hay que verificar que en efecto lo son. Para esto se utiliza frecuencias de ocurrencia: Si un término aparece con cierta frecuencia, sus semejantes han de mantener esa frecuencia. Los detalles están en [3].

1.2. Planteamiento del problema

El Departamento de Biología Celular del Cinvestav-IPN requiere una aplicación para el análisis de grandes cantidades de información de importancia para sus investigaciones. No se cuenta con un sistema que realice de manera automatizada dichas consultas, la consulta de la información en las diversas bases se realiza de manera manual, ingresando a las interfaces de cada una de ellas, obtenidos los resultados dichos datos son enviados a hojas de cálculo de Microsoft Excel manualmente. Una vez que se tiene toda la información necesaria, ésta es manipulada con Microsoft Access, el manejador de Microsoft Access no está diseñado para almacenar grandes cantidades de información por lo que presenta irregularidades al mostrar su contenido sumándole el tiempo invertido en la elaboración de la base de datos local.

1.3. Principales objetivos

Construir un sistema de recuperación de información haciendo uso de la minería de datos y diversas bases de datos bioinformáticas.

1.3.1. Objetivos específicos

1. Contar con un propio recuperador de información dotado de reglas heurísticas de búsqueda de intereses y aplicaciones propios del Departamento de Biología Celular del Cinvestav-IPN, en particular, del laboratorio del Dr. Saúl Villa Treviño, así como experimentar con diversas funciones de semejanza.

2. Hacer uso de de las operaciones relaciones convencionales SQL.

3. Manejar operaciones considerando funciones de similitud.

4. Creación del diseño de una base bioinformática tal que permita la búsqueda eficiente de la información.

5. Manejo de estándares técnicos para bases de datos.

6. Utilización de formatos de almacenamiento de tipo estándar.

1.4. Organización de la tesis

El documento de tesis está organizado de la forma siguiente: el capítulo 2 es una breve explicación sobre los conceptos básicos de bases de datos, lenguajes de consulta, manejador de bases de datos y una introducción a la minería de datos como herramienta para la explotación de la información. También se da un explicación sobre los que son las dependencias funcionales. En el capítulo 3 se aborda la temática de la bioinformática, la importancia de las bases de datos bioinformáticas, una reseña de los servidores existentes para bases de datos bioinformáticas y la manera en la que se pueden accesar a algunas de ellas (`Medline, GenBank` y `SwissProt`). El capítulo 4 se refiere al diseño del sistema de recuperación de información desarrollado, muestra los diagramas de modelado hecho en Lenguaje Unificado de Modelado (UML). En el capítulo 5 se describen los algoritmos aplicados para la implementación de las funciones durante el desarrollo del sistema. El capítulo 6 hace referencia a los resultados de este proyecto, y finalmente el capítulo 7 contiene las conclusiones y trabajo a futuro.

Capítulo 2

Bases de datos y recuperación de la información

Un *sistema manejador de bases de datos* (DBMS) es el software que permite a una o varias personas usar y/o modificar datos [6]; consiste de una colección de datos interrelacionados además de un conjuto de programas para accesar a los datos. A esta colección de datos se le conoce como *base de datos* [7]. Existen diferentes tipos de bases de datos como por ejemplo, las bases de datos relacionales, bases de datos orientadas a objetos, bases de datos de objetos relacionales, entre otras. Vamos a enfocarnos a las bases de datos relacionales debido a que son las que nos interesan para este proyecto.

2.1. Bases de datos relacionales

El objetivo del diseño de las bases de datos ralacionales es generar un conjunto de esquemas relacionados entre sí que permite almacenar información sin redundancia, así como recuperar información fácilmente. Un problema es diseñar esquemas que estén en una *forma normal* apropiada. Para determinar si un esquema relacional está en una de las formas normales, necesitamos información adicional sobre el mundo real que modelaremos con la base de datos [7]. Algunos conceptos necesarios para entender mejor el tema se describen a continuación:

tupla sea *T1,T2,...,Tn* ($n \geq 0$) tipos de nombres, no necesariamente todos distintos. Asociar con cada *Ti* un nombre de atributo distinto *Ai*; cada una de las n combinaciones atributo-nombre:tipo-nombre que resulta es un *atributo*. Al asociar cada atributo a un valor *vi* del tipo *Ti*; cada uno de los n valores de las combinaciones atributo:valor que resulta se le denomina *componente*.

Ahora bien, t es el *valor de una tupla*(o bien*tupla*) sobre los atributos *A1,A2,...,An*. El valor n es el *grado* de t; una tupla de grado uno se le llama *unaria*, una tupla de grado dos es *binaria*, una tupla de grado tres es *ternaria*,..., y generalmente

una tupla de grado n es n-aria. El conjunto de todos los n atributos nos da el *encabezado o heading* de t [8].

relación sea {H} el encabezado y sean *t1,t2, ... , tm*($m \geq 0$) tuplas distintas con encabezado {H}. La combinación, r de {H} y el conjunto de tuplas {*t1,t2, ..., tm*} es un *valor relacional* (o *relación*) sobre los atributos *A1,A2, ..., An*, donde *A1,A2, ..., An* son los atributos en {H}. El *encabezado* de r es {H}; r tiene los mismos atributos (y por lo tanto los mismos nombres y tipos) y el mismo grado como encabezado que es. El *cuerpo* de r es el conjunto de tuplas {*t1,t2, ..., tm*}. El valor m es la cardinalidad de r [8].

2.1.1. Formas normales y optimización de bases de datos

La *normalización* es el proceso reversible de reemplazamiento paso a paso apartir de un conjunto de relaciones en la cual las relaciones obtienen de manera gradual estructuras más simples. La reversibilidad garantiza que la colección original de relaciones pueda ser recuperada y por lo tanto que la información no se pierda.
Los objetivos de la normalización se enumeran a continuación:

1. Crear una representación viable de cualquier relación en la base de datos
2. Obtener algoritmos de recuperación de información poderosos basados en una simple colección de operaciones relacionales
3. Obtención de relaciones libres de inserciones no deseadas, actualizaciones, y eliminación de dependecias
4. Reducir la necesidad de reestructuración de relaciones cuando un nuevo tipo de dato es introducido
5. Crear la colección de relaciones neutrales para consultas estadísticas, donde dichas estadísticas deben actualizarse conforme pasa el tiempo.

Los primeros dos objetivos son aplicables al primer paso de normalización (conversión a la primera forma normal). Los últimos tres se aplican a todo el proceso de normalización.

Primera forma normal o 1NF trata de la estructura de la relación. Esencialmente requiere que cada atributo de la relación esté basada en un dominio simple. Cualquier relación puede estar en *1NF* algorítmicamente reemplazando un dominio no simple por sus dominios simples que lo consitituyen. El problema reside en elegir relaciones fuertemente unidas al hecho de que los valores de algunos atributos determinen completamente los valores de otros atributos en una relación.

Segunda forma normal o 2NF. Una relación R está en *2NF* si R está en *1NF*, y cada atributo en R es totalmente dependiente de cada llave.

Tercera forma normal o 3NF. Una relación R está en *2NF* si R está en *2NF*, y ningún atributo de R es transitivamente dependiente de cualquier llave de R.

Cualquier relación en *2NF* tiene la propiedad de que cada atributo dependiente de la relación tampoco es parcialmente dependiente ni transitivamente dependiente de cualquier llave. Esto significa que atributos no llave son independientes unos de otros [9].

2.1.2. Beneficios de la normalización de datos

El mayor beneficio de normalizar una base de datos desde la perspectiva del Manejo de Sistemas de Información (MIS) incluye:

- Desarrollo de una estrategia para la construcción de relaciones y selección de llaves
- Mejora de interfaces con usuarios finales en actividades computacionales (por ejemplo, habilidad para acomodar peticiones no planeadas)
- Reducción de problemas asociados con inserción y eliminación de datos
- Identificación de problemas potenciales que probablemente requieran análisis adicional y documentación

Desde la perspectiva del usuario final, una base de datos normalizada se convertirá en una mejora de tiempo de respuesta desde la organización del MIS. La información es el conocimiento derivado de los datos y la normalización de datos es el componente principal usado para transformar datos en información [10].

Una vez que a los datos se les ha dado el tratamiento adecuado para su mejor manipulación debemos conocer la manera en la que deben ser consultados. En la siguiente sección mencionaremos algunos lenguajes de consulta en bases de datos y el que se utilizó a lo largo de este proyecto.

2.2. Lenguajes de Consulta

Los sistemas de bases de datos requieren de un lenguaje de consulta que sea amigable para los usuarios. Existen diversos tipos de lenguajes de consulta como por ejemplo: *Consulta Mediante Ejemplos(QBE, Query-by-Example), Lenguaje Lógico de Base de Datos(Datalog, Database Logic)* y el *Lenguaje de Consulta Estructurado(SQL, Structured Query Language)*. Cada uno con distintos estilos. *QBE* está basado en el cálculo relacional de dominios, *Datalog* está basada en la lógica de programación del lenguaje *Prolog* y *SQL* que usa una combinación de álgebra relacional y constructores de cálculo-relacional.

Aparte de hacer consultas a bases de datos, estos lenguajes contienen muchas otras características para definición de estructuras de datos, modificación de datos en la base, y restricciones específicas de seguridad [7].

Enfaticemos cada uno de estos lenguajes para conocer las diferencias entre ellos.

préstamo	nombre-sucursal	número-préstamo	importe
	Navacerrada	P._p	

Tabla 2.1: Expresión de consultas en QBE por conjunción.

2.2.1. *QBE. Query By Example*

Query-By-Example (QBE, Consulta mediante ejemplos) se refiere a una familia de lenguajes que implementan las ideas del cálculo relacional de dominios.

Es el nombre de un lenguaje de manipulación de datos como el de un sistema de base de datos que incluyó a este lenguaje. El sistema QBE se desarrolló en el Centro de Desarrollo "T.J . Watson", de IBM, a principios de los setenta y el lenguaje de manipulación de datos QBE se usó mas tarde en *Query Management Facility*(QMF, mecanismo de gestión de consulta) como opción de interfaz para DB2.

Sistema QBE de IBM

Características

QBE tiene una sintaxis bidimensional. Las consultas se presentan en forma de tablas.

Una consulta en un lenguaje unidimensional (como SQL) se puede formular en una línea (posiblemente larga). Un lenguaje bidimensional necesita dos dimensiones para la formuación de consultas.

Las consultas en QBE se expresan "mediante un ejemplo".

Expresión de consultas

Esqueletos de tablas presentan el esquema de relación que se rellena con *filas ejemplo*. Una fila ejemplo está formada por constantes y *elementos de ejemplo*, que son variables de dominio.

Las variables de dominio van a ser precedidas por _ . Ejemplo: _x

Las constantes aparecen sin ninguna indicación particular.

Condiciones de selección sobre tuplas:

- Conjunción: por filas
- Disyunción: por columnas

El ejemplo para el caso de conjunción puede ser visto en la tabla 2.1.

P. significa Print(mostrar). En cálculo relacional de dominios:

prestatario	nombre-cliente	número-préstamo
	Santos	P._x
	Gómez	P._y

Tabla 2.2: Expresión de consultas en QBE por disyunción.

préstamo	número-préstamo	nombre-sucursal	importe
	_x	Navacerrada	

Tabla 2.3: Expresión de consultas en QBE forzando datos (A).

$\{ \langle p \rangle | \exists s, j(\langle s, p, i \rangle \in \textit{préstamo} \land s =$ "Navacerrada"$)\}$

El ejemplo para el caso de disyunción puede ser visto en la tabla 2.2

Consultas sobre varias relaciones

En el producto cartesiano o la reunión(`join`). Las conexiones entre varias relaciones se hace con variables, que obligan a algunas tuplas a tomar el mismo valor en ciertos atributos.

Ejemplo: Supóngase que se desea encontrar los nombres de todos los clientes que tienen un préstamo en la sucursal Navacerrada. Ver tablas 2.3 y 2.4.

El sistema localiza las tuplas en la relación *préstamo* que tienen el atributo *nombre-sucursal* igual a "Navacerrada".

Para cada una de esas tuplas, el sistema busca las tuplas de la relación *prestatario* con el mismo valor para el atributo *número-préstamo* que el mismo atributo de la tupla de la relación *préstamo* [11].

Para leer más sobre el tema consultar la página:
`http://www.fdi.ucm.es/profesor/milanjm/bdsi0304/Tema03-QBE.pdf`.

2.2.2. *Datalog. Database Logic*

Datalog es un lenguaje lógico que es la forma más simple de lógica desarrollada para el modelo relacional. Datalog sin recursión tiene el mismo poder expresivo que el

prestatario	nombre-cliente	número-préstamo
	P._y	_x

Tabla 2.4: Expresión de consultas en QBE forzando datos (B).

álgebra relacional. Sin embargo, SQL:1999 ha usado la solución para la recursión en Datalog para el desarrollo de consultas recursivas. Datalog es similar a Prolog en su sintaxis, pero su semántica operacional es diferente. Una regla o cláusula en Datalog tiene la forma:

$$\text{cabeza} \longleftarrow \text{cuerpo}$$

donde *cabeza* es un átomo y *cuerpo* es una lista de átomos que puede ser vacía; en este caso se habla de un hecho. Los hechos se escriben:

$$\textit{cabeza}.$$

Un átomo es de la forma:

$$\mathbf{P}(t_1, ..., t_n)$$

Donde **P** es un símbolo de predicado y t_i son variables constantes. No se admiten símbolos de función en t_i, a diferencia de *Prolog*.

Significado de las reglas lógicas

Una regla se escribe:

$$P \longleftarrow Q_1, ..., Q_n.$$

y se lee: "Si $Q_1, Q_2, ...$ y Q_n son ciertos, entonces P es cierto". Si $n = 0$ "P es cierto", y se escribe:

$$P.$$

Hay formas alternativas de definir el significado de las reglas:

Interpretación de la teoría de pruebas. Es el conjunto de todos los hechos que se pueden probar a partir de las reglas del programa usándolas de todas las formas posibles.

Intepretación de la teoría de modelos. La interpretación de una colección de predicados asigna cierto o falso a cada posible instancia de los predicados, donde los argumentos se escogen de un conjunto infinito de constantes. Lainterpretación se representa habitualmete por el conjunto de instancias verdaderas.

Definición computacional. La última forma de definir el significado de las reglas lógicas es proporcionar una algoritmo para ejecutarlas para determinar si un hecho es cierto o falso. Prolog define su semántica de esta forma.

Inconveniente: hay hechos que no se pueden probar de esta forma (ramas infinitas).

2.2.3. *SQL. Structured Query Language*

El *Lenguaje de Consulta Estructurado* (SQL) se ha establecido como el lenguaje estándar de bases de datos relacionales. Existen varias versiones de SQL. La versión original fué desarrollada en *IBM*. Este lenguaje, originalmente llamado *Sequel*, fué implementado como parte del proyecto *System* ***R*** a principios de los años $70's$.

El lenguaje *SQL* [7] consta de varias partes:

Lenguaje de definición de datos (DDL). El DDL de SQL provee comandos para definición de esquemas relacionales, borrado de relaciones, creación de índices, y modificación de esquemas relacionales.

Lenguaje de Manipulación de datos interactivo (DML). El DML de SQL incluye un lenguaje de consulta basado tanto en álgebra relacional como cálculo de tuplas relacionales. Esto incluye comandos para inserción de tuplas, borrado de tuplas, y modificación de tuplas en la base de datos.

DML Embebido. La forma embebida de SQL está diseñada para su uso dentro de lenguajes de programación de propósito general, como *Cobol, Pascal, Fortran,* y *C*.

Definición de vista. El DDL de SQL incluye comandos para definición de vistas.

Autorización. El DDL de SQL incluye comandos para especificación de derechos de acceso a relaciones y vistas.

Integridad. El DDL de SQL incluye comandos para especificación restricciones de integridad para almacenamiento de datos en la base de datos mas satisfactoria. Actualiza la violación de restricciones de integridad que están deshabilitadas.

Control de transacción. SQL incluye comandos para especificación de principio y término de transacciones.

.

Estructura básica

Una base de datos relacional consiste de una colección de relaciones. A cada una de ellas se le asigna un único nombre. SQL permite el uso de *valores nulos* para indicar que el valor es desconocido o no existe. También permite que el usuario especifique a cuales atributos no se les puede asignar el valor nulo.

La estructura básica de una expresión SQL consiste de tres cláusulas: **select, from** y **where**.

- **select**: es la cláusula que corresponde a la operación de proyección del álgebra relacional. Es utilizada para listar atributos como resultado de una consulta.

SELECT	Recuperación de datos
INSERT UPDATE DELETE MERGE	Lenguaje de Manipulación de Datos (DML)
CREATE ALTER DROP RENAME TRUNCATE	Lenguaje de Defínción de Datos(DDL)
COMMIT ROLLBACK SAVEPOINT	Control de transacción
GRANT REVOKE	Lenguaje de Control de Datos(DCL)

Tabla 2.5: Declaraciones SQL.

- **from**: esta cláusula corresponde a la operación del producto-cartesiano del algebra relacional. Lista las relaciones para ser leidas en la evaluación de la expresión.
- **where**: esta cláusula corresponde a la selección de predicados del álgebra relacional. Consiste de un predicado con atributos de la relación que resulta de la cláusula **from**.

El término *select* tiene diferente significado en SQL y en el álgebra relacional. Una consulta típica SQL [7] tiene la forma:

$$\textbf{select } A_1, A_2, ..., A_n \textbf{ from } r_1, r_2, ..., r_m \textbf{ where } P$$

Cada A_i representa un atributo, y cada r_i una relación. P es un predicado. La consulta *select* es equivalente a la expresión del álgebra relacional $\prod_{A_1,...,A_n}(\sigma_P(r_1 \times r_2 \times ... \times r_m))$

Declaraciones SQL

Las declaraciones que maneja SQL junto con el tipo de lenguaje que ocúpan pueden ser vistas en la tabla 2.5.

Tanto el Instituto de Estándares Nacionales Americanos (*ANSI*) como la Organización de Estándares Internacionales (*OSI*) aceptaron a SQL como el lenguaje estándar para bases de datos relacionales [12].

Declaración	Descripción
`SELECT`	Recupera datos de la base de datos

`INSERT`
`UPDATE`
`DELETE`
`MERGE`
	Introduce nuevos renglones, cambia renglones existentes, remueve rengloes no deseados de las tablas en la base de datos respectiva.De manera conjunta se le conoce como *Lenguaje de Manipulación de Datos*(DML).

`CREATE`
`ALTER`
`DROP`
`RENAME`
`TRUNCATE`
	Levanta, cambia, y remueve estructuras de datos desde tablas.Conjuntamente a esto se le conoce como *Lenguaje de Definición de Datos*(DDL).

`COMMIT`
`ROLLBACK`
`SAVEPOINT`
	Maneja los cambios hechos por las declaraciones DML. Cambia los datos que pueden ser agrupados juntos dentro de una transacción lógica

`GRANT` `REVOKE`	Proporciona o remueve derechos de acceso. Conjuntamente a este se le conoco como *Lenguaje de Control de Datos*(DCL)

Tabla 2.6: Descripción de declaraciones SQL.

En la tabla 2.6 se hace una descripción de los lenguajes que utiliza SQL para la ejecución de sus declaraciones.

El tipo de operaciones que utiliza SQL en las relaciones son **union, intersect,** y **except** y corresponden a las operaciones del álgebra relacional $\cup, \cap, y-$. Como la *unión, intersección* y el conjunto *diferencia* en el álgebra relacional, las relaciones participantes en las operaciones deben ser *compatibles*; es decir, deben tener el mismo conjunto de atributos [7].

Relaciones de unión

Adicionalmente SQL provee mecanismos basicos de Producto-cartesiano para unón de tuplas, también provee otras mecanismos para unión de relaciones, incluyendo condiciones de `join`, `natutal join`, así como otras muchas formas de `outer joins`. Estas operaciones adicionales son usualmente utilizadas con *subqueries* (consultas anidadas) mediante la cláusula `from` [7].

Una *tabla unida* (en inglés, joined table) es una tabla derivada de otras dos tablas(reales o derivadas), actuamente existen distintos tipos de uniones particulares. `Inner, outer` and `cross`.

Para cada combinación de renglones de *T1* y *T2*, la tabla derivada contendrá un renglón compuesto por todas las columnas en *T1* seguido por todas las columnas en *T2*. Si la tabla tiene N y M renglones respectivamente, la tabla unida tendrá $N * M$ renglones.

`T1` { [`INNER`] | { `LEFT` | `RIGHT` | `FULL` } [`OUTER`] } `JOIN T2 ON` boolean_expresion

`T1 { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2 USING` (lista de columnas join)
`T1 NATURAL { [INNER] | { LEFT | RIGHT | FULL } [OUTER] } JOIN T2`

La *condición de unión* se especifica en las cláusulas `ON` o `USING`, o implícitamente por la palabra `NATURAL`. La condición de la unión determina cuales renglones de la tabla fuente son consideradas para hacer el "match".

2.3. Manejador de bases de datos (DBMS)

El DBMS incluye mecanismos para aplicación de programas para almacenar, recuperar y modificar datos, y también permite a las personas consultar interactivamente la respuesta a preguntas específicas.

Un DBMS es una pieza muy compleja de sowftware. Un solo DBMS puede controlar múltiples bases de datos, cada una usualmente esta consituída de varias tablas diferentes llenas de datos [13].

Un DBMS soporta aplicaciones individuales y sistemas de información completos con modelado y capacidades de almacenamiento de datos, así como recuperación y facilidad de manipulación de datos para usuarios concurrentes o transancciones. Muchas aplicaciones requieren del manejo de tipos de datos que no son bien manipulados por DBMSs convencionales. Algunos ejemplos de estos nuevos datos son datos multimedia, documentos, datos temporales, entre otros. SQL, la abstracción de la transacción, y el concepto de modelo de datos - el más notable es el modelo relacional, y el menos amplio, el modelo de datos orientado a objetos - son ingredientes cruciales para el manejo de datos. Para conocer todos los nuevos requerimientos, DBMS aparentemente tienen que ser ampliados por nuevas funcionalidades. Un solo sistema con todos los módulos implementando todas las nuevas funcionalidades no es viable por varias razones [14].

- El DBMS podría llegar a ser tan grande y, consecuentemente, complejo que no se le podría dar mantenimiento a un costo razonable.
- Los usuarios tendrían que pagar altos precios por la adición de funcionalidades, que incluso no ocuparán en su totalidad.
- Usuarios y aplicaciones también habrían que pagar una penalización de rendimiento por añadir funcionalidades que actualmente no usarían.
- El vendedor del DBMS probablemente no tenga expertos para la realización de cada extensión o los recursos para comenzar todas las extensiones en un periodo de tiempo rasonable.

2.3.1. Principales componentes de un DBMS

Como se mencionó en la sección anterior, las extensiones afectan a un DBMS su arquitectura y también requieren de ciertos requisitos. Por lo que se dará una breve explicación de la arquitectura y subsecuentemente de los compontenentes del DBMS (CDBMS).

Arquitectura del DBMS Existen varios tipos de "arquitecturas" que sirven para diferentes propósitos. Por ejemplo, la *arquitectura de tres niveles* (el cual distingue esquemas externos con los cuales trabajan los usuarios, los internos, esquemas integrados de la base de datos entera, y los esquemas físicos que determinan el almacenado y organización de la base de datos sobre un almacén secundario) refleja los diferentes niveles de abstracción de datos en un sistema de base de datos [14].

Ahora bien, la manera en la que se comunican los manejadores debase de datos con los lenguajes de programación para accesar a las bases es mediante *contectores*, en la siguiente subsección se dará una descripción de este tipo de software.

2.3.2. Conectores a bases de datos

Se refiere al software capaz de realizar la interección con sisitemas gestores de bases de datos y el lenguaje de programación. Cualquier sistema que requiera el uso de bases de datos necesita de al menos un conector a base de datos.

ODBC *(Open DataBase Connectivity)*

Interface de aplicaciones *API* para accceder a datos en sistemas gestores de bases de datos utilizando para ello *SQL*.

En general, el ODBC es un estándar de acceso a bases de datos, que permite mantener independencia entre los lenguajes de programación, los sistemas de bases de datos y los sistemas operativos. ODBC logra esto al insertar una capa intermedia llamada manejador de Bases de Datos, entre la aplicación y el DBMS, el propósito de esta capa es traducir las consultas de datos de la aplicación en comandos que el DBMS entienda. Para que esto funcione tanto la aplicación como el DBMS deben ser compatibles con ODBC. La aplicación debe ser capaz de producir comandos ODBC y el DBMS debe ser capaz de responder a ellos.

Un ejemplo es, si escribimos una aplicación para acceder a las tablas de una BD de Access, ¿qué ocurrirá si después queremos que la misma aplicación, y sin reescribir nada, utilice tablas de SQL Server u otra BD cualquiera? La respuesta es sencilla: no funcionará. Nuestra aplicación, diseñada para un motor concreto, no sabrá dialogar con el otro. Evidentemente, si todas las BD funcionaran igual, no tendríamos este problema.

Pero si hubiera un elemento que por un lado sea siempre igual, y por el otro sea capaz de dialogar con una BD concreta, solo tendríamos que ir cambiando este

elemento, y nuestra aplicación siempre funcionaría sin importar lo que hay al otro. A esas piezas intercambiables le llamamos ODBC [15].

JDBC. Java DataBase Connectivity

JDBC es el acrónimo de Java Database Connectivity, un API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede utilizando el dialecto SQL del modelo de base de datos que se utilice.

El API JDBC se presenta como una colección de interfaces Java y métodos de gestión de manejadores de conexión hacia cada modelo específico de base de datos.

JDBC es realmente similar a lo que es ODBC, pero esta diseñado específicamente para la ejecución de aplicaciones desarrolladas en Java, mientras que ODBC es independiente al lenguaje de programación [16].

2.4. Recuperación de Información

La *Recuperación de Información* consiste en dado un conjunto de datos jeraquizados o no, encontrar aquella información relevante para el usuario. Compara una consulta del usuario con una gran colección de documenos devolviendo una lista ordenada de acuerdo a los documentos que mejor se ajustan a la consulta [17].

2.4.1. Minería de datos

La gran cantidad de datos que se maneja en las investigaciones y sus grandes dimensiones sirve para crear datos de expresiones de genes aplicables en funciones de *minería de datos*. La *minería de datos* es el proceso de descubrimiento de conocimiento interesante a partir de grandes cantidades de datos alojados también en bases de datos u otros repoositorios de información [18]. Ahora bien, el *KDD* o *Knowledge Discovery and Data Mining*, se define como el proceso no trivial capaz de identificar datos válidos, novedosos y potencialmente útiles [17]. También se trata de las técnicas para encontrar y describir estructuras de patrones en datos así como de herramientas para ayudar a explicar datos y crear predicciones a partir de estos [19]. Un *KDD* es empleado para describir el proceso completo de extracción de conocimiento a partir de los datos. En este contexto, el conocimiento significa relacionar y encontrar patrones entre los datos. *KDD* abarca un campo multidiciplinario de investigación; *machine learning*, *estadística*, *visualización, Bases de datos y Sistemas expertos*. Por lo que *minería de datos* debería ser usado exclusivamente para la etapa de descubrimiento en el proceso del *KDD* [19].

Un KDD consiste de una secuencia iterativa de los siguientes pasos:

- Limpieza de datos

Figura 2.1: La Minería de datos es multidiciplanaria.

- Integración de datos
- Selección de datos
- Transformación de datos
- Minería de datos
- Evaluación de patrones
- Presentación de conocimiento

Ahora bien, nosotros nos enfocaremos únicamente en la parte de Minería de datos y Evaluación de patrones, en particular en el área de Base de datos como se puede ver en la figura 2.1.

El interés en la minería de datos en parte puede ser explicada de la siguiente forma. En los años 80, las mayoría de las organizaciones contruyeron bases de datos, con información de sus clientes, su competencia, y productos. Estas bases contienen gigabytes de datos de información que no se le puede dar seguimiento fácilmente usando *SQL*. Los algorítmos de minería de datos pueden encontrar peculiaridades interesantes en una base de datos. El Lenguaje de Consulta Estructurado,*SQL* por sus siglas en inglés; solo ayuda a encoontrar información bajo limitantes que uno conoce. Los algoritmos de minería de datos se encargan de encontrar información relevante en subpartes de la base de datos. En la mayoría de los casos repiten consultas SQL y almacenan resultados intermedios. Lo cual puede hacerse de manera manual pero es un trabajo tedioso [20].

Minería de datos vs herramientas de consulta

Primeramente es importante mencionar que las herramientas de consulta y la minería de datos son complementarias una de la otra. La minería de datos no reemplaza una herramienta de consulta, pero le brinda al usuario mayores beneficios. Suponiendo que se tiene información que consiste de millones de registros que describen las compras hechas por los clientes en los últimos diez años. Hay una riqueza de información potencial en cada archivo, la mayoría puede ser encontrada haciendo uso de consultas normales con SQL en la base de datos, tal como -¿quién compró que producto en que año?, ¿cuál es el promedio de ventas en una cierta región en el mes de julio?, y así sucesivamente. De cualquier forma existe conocimiento oculto en las bases de datos que es mucho más difícil de encontrar usando SQL. Uno podría definir criterios para perfiles de clientes y consultar las bases de datos para ver si funcionan o no. En un proceso de prueba y error, intuitívamente uno podría ir descubriendo que atributos son importantes. Siguiendo esta trayectoria uno podría tardar días o meses para encontrar una segmentación óptima de una larga base de datos, mientras que haciendo uso de algoritmos de minería de datos la respuesta se podría encontrar en un tiempo mucho más corto, algunas veces incluso en minutos o en un par de horas. Una vez que las herramientas de la minería de datos ha encontrado una segmentación, se puede hacer uso del ambiente de consulta nuevamente para consultar y analizar la información encontrada.

Se puede decir que si uno sabe exactamente que es lo que se esta buscando, hay que usar SQL; pero si solo se sabe vagamente lo que se está buscando, hay que hacer uso de la minería de datos. Generalmente es más fácil que uno comience a buscar información de manera vaga a las veces que cuando uno sabe precisamente lo que se está buscando, esto es lo que ha motivado el surgimiento de la Minería de datos [20].

2.4.2. Dependencia funcional

Las dependencias funcionales son limitaciones sobre un conjunto de relaciones. Estas nos permiten expresar hechos que estamos modelando con nuestra base de datos y juegan un papel importante en el diseño de una base de datos. Definimos el concepto de una *super-llave* como sigue. Sea R una relación. A un subconjunto K de R, es una *super llave* de R si, en cualquier relación r(R), para todas las parejas t_1 y t_2 de tuplas en r tales que $t_1 \neq t_2$, entonces $t_1[K] \neq t_2[K]$. Esto es, no hay dos tuplas en cualquier relación r(R) que tenga el mismo valor sobre atributos del conjunto K. La dependencia funcional es la generalización de lo que es una super-llave. Sea $\alpha \subseteq R$ and $\beta \subseteq R$. La *dependencia funcional*

$$\alpha \longrightarrow \beta$$

contenida en R es válida si en cualquier relación $r(R)$, para todas las parejas de tuplas t_1 y t_2 en r cumple que $t_1[\alpha] = t_2[\alpha]$, también el caso de $t_1[\beta] = t_2[\beta]$.

Decimos que K es una super-llave de R si $K \longrightarrow R$. Esto es, Kes super-llave si para cualquier $t_1[K] = t_2[K]$, al igual que en el caso de $t_1[R] = t_2[R]$(que es, $t_1 = t_2$).

A	**B**	**C**	**D**
a_1	b_1	c_1	d_1
a_1	b_2	c_1	d_2
a_2	b_2	c_2	d_2
a_2	b_3	c_2	d_3
a_3	b_3	c_2	d_4

Tabla 2.7: Relación r.

Las dependencias funcionales nos permiten expresar restricciones que no podemos expresar usando *superllaves*.

Consideremos la función r de la figura 2.7, hay que ver que dependencias funcionales se satisfacen. Se observa que $A \longrightarrow C$ se satisface. Hay una pareja de tuplas que tiene un valor en A de c_1. Estas tuplas tienen el mismo valor en C llamado c_1. De manera similar, las dos tuplas en A de valor a_2 tienen el mismo valor en C de c_2. No hay otra pareja de tuplas distintas que tengan el mismo valor en A. La dependencia funcional $C \longrightarrow A$ no se satisface. Consideremos las tuplas $t_1 = (a_2, b_3, c_2, d_3)$ y $t_2 = (a_3, b_3, c_2, d_4)$. Esta pareja de tuplas tienen el mismo valor en C de c_2, pero tienen diferente valor en A, a_2 y a_3, respectivamente. Entonces, encontramos un par de tuplas t_1 y t_2 tales que $t_1[C] = t_2[C]$, pero $t_1[A]$ no es igual a $t_2[A]$.

Existen otras dependencias funcionales que se satisfacen, incluyendo, por ejemplo, la dependencia funcional $AB \longrightarrow D$ [7].

Capítulo 3

Bioinformática

La Bioinformática puede ser definida como la aplicación de la tecnología de la computación al manejo de la información biológica. Esto abarca el estudio de la información genética, subyacente a la estructura molecular, resultando en funciones bioquímicas, y las características fenotípicas. Aquí las computadoras son usadas para unir, almacenar, analizar e integrar información biológica y genética. Las recientes investigaciones abarcan el diseño e implementación de programas y sistemas para el almacenamiento, manipulación, y análisis de las bastas cantidades del ácido desoxirribonucléico o ADN y datos de secuencias proteínicas. La *minería de datos biológica* es un campo de investigación y desarrollo enfocado en esta dirección. La gran cantidad de datos originados a partir de recientes experimentos biológicos han permitido la creación de muchas bases de datos que contienen información de genomas, secuencias proteínicas, expresiones de genes, y otros tipos de datos. En Biología Celular los investigadores a menudo recuperan información de estas bases de datos basándose en una característica, tal como la secuencia de aminoácidos, connotación del gen, o nombre de la proteína [17].

3.1. Bases de datos bioinformáticas

Como vimos en el capítulo 2, una base de datos puede definirse como una serie de datos organizados y relacionados entre sí, los cuales son recolectados y explotados por los sistemas de información. Los datos contenidos en las bases pueden ser de diversos tipos, para el caso de las bases de datos bioinformáticas el tipo de información es de tipo biológica, genética, celular, etc. Este proyecto está basado en información de tipo genética por lo que se dará una breve explicación de lo que es un gén y sus características para tener una mejor comprensión sobre el tema.

Un gen es un constituyente fundamental de cualquier organismo vivo [17]. La secuencia de genes en el cuerpo humano representa la firma o identidad de cada persona. Los genes son porciones de ácido desoxiribonucléico, o ADN, conformados de dos cadenas. Cada cadena esta compuesta de fósfato y moléculas de azúcar desoxirribosa unidas por cadenas covalentes. Una base nitrogenada esta unida a cada molécula de

Tipo	Base
Purina	**A**denina
	Guanina
Pirimida	**C**itocina
	Timina

Tabla 3.1: Cuatro bases nitrogenadas y sus tipos.

azúcar. Hay cuatro bases fundamentales: *adenina*[`A`], *citocina*[`C`], *guanina*[`G`], y *tímina*[`T`]. Desde la perspectiva teórica de la información, el ADN puede ser considerado como una cadena de símbolos. Cada símbolo es una de las cuatro bases descritas arriba `A, C, G, o T` [17].

La **purina** es una base nitrogenada. Dos de las bases de los ácidos nucleicos, adenina y guanina son derivados de la purina. En el ADN estas bases se unen con sus *pirimidinas* complementarias, la timina y la citocina respectivamente, a través de enlaces de hidrógeno.

La **pirimida** es un compuesto orgánico, pero con un anillo *heterocíclico*, esto es, un anillo formado por más de un tipo de átomo: dos átomos de nitrógeno susitituyen al carbono. Dos bases de los ácidos nucleicos(citocina y timina) son derivados pirimídicos. En el ADN, estas bases forman puentes de hidrógeno con sus purinas complementarias. Veáse la tabla 3.1.

La secuencia `A-T, G-C` de ADN determina el orden exacto de las bases a lo largo de una cadena. La manipulación del ADN involucra una determinada secuencia en algún lugar a lo largo de la cadena. Una aplicación utilizada de manera creciente está en el estudio de las relaciones evolutivas entre grupos de especies relacionadas. Otra aplicación aún mayor esta en el análisis de mutaciones que causan enfermedades genéticas. El conocimiento de las secuencias nucleótidas de un gen humano específico en asociación con el código genético puede ser usado para inferir la secuencia de amino ácidos de la proteína codificada - frecuentemente éste es un paso crucial en el diagnóstico y manipulación de un desorden heredado [21].

Entender que partes del genoma están codificados en que genes es el área principal de estudio en biología molecular computacional o *Bioinformática*. El resultado de algoritmos de empatamiento de cadenas y sus derivados han sido aplicados en búsquedas, análisis y secuencias de ADN, y otros avances en Bioinformática [17].

Los experimentos en microarreglos de ADN son hechos para producir patrones de expresiones de genes, que proveen información dinámica sobre funciones celulares. La gran cantidad de datos de expresiones de genes los hacen, y sus grandes dimensiones, los hacen candidatos para la aplicación de funciones de minería de datos [17], es por ello que las bases bioinformáticas están constituídas por este tipo de información y de ahí la importancia de su manipulación.

Como vemos la inforamción genética afecta cada vez mas los aspectos de la vida diaria, por lo que se requiere de un análisis cuidadoso de los datos genéticos, principalmente para la búsqueda de enfermedades genéticas humanas.

3.2. Reseña de los servidores existentes

Las secuencias del ADN son coleccionadas dentro de bases de datos de la misma manera que las secuencias preteínicas. Dentro de las principales bases de datos se encuentra `GenBank`,la Base de Datos de Secuencias Genómicas (`GSBD`) de los Estados Unidos, la Base de Datos de Secuencias Nucleótidas (`EMBL`) del Laboratorio de Biología Molecular Europeo, y el Banco de Datos de ADN de Japón (`DDBJ`) [22].

Nosotros nos referiremos a las bases bioinformáticas de mayor utilidad para el Departamento de Biología Celular del Cinvestav-IPN las cuales se listan a continuación:

`GenBank` es la base de datos de secuencias genéticas, contiene una colección de todas las secuencias de ADN disponibles para todo el público. `GenBank` forma parte de International Nucleotide Sequence Database Collaboration, el cual esta comprendido por el Banco de Datos de ADN de Japón y el laboratorio de Biología Molelecular Europeo [23].

`Swiss-Prot` es la base de datos de secuencias de proteínas, fué fundada en 1986 y mantenida desde 1987 por el primer Departamento de Bioquímica de la Universidad de Génova y actualmente por el Instituto Suizo de Bioinformática y el Instituto de Bioinformática Europeo [24].

`Medline`es la base de datos bibliográfica de referencia médica de la *National Library of Medicine (NLM)* de Estados Unidos que contiene citas bibliográficas y resúmenes de cerca de 3800 revistas biomédicas publicadas en Estados Unidos y otros setenta países, abarcando los campos de la medicina, enfermería, odontología, veterinaria, salud pública y ciencia preclínicas. En la actualidad contiene más de 12 millones de citas registradas de artículos publicados desde 1966 e incorpora semanalmente los últimos artículos publicados. Hasta hace poco la utilización de `Medline` estaba reservada a centros o bibliotecas con la suficiente cantidad para pagar una versión CD-ROM o la suscripción a un servicio de información en línea. Actualmente ofrece libre acceso a la informaciń utilizando la red. Es un servicio desarrollado por la *NLM* y la *National Center for Biotechnology Information (NCBI)* [25].

3.3. Accesos a base de datos bioinformáticas y sus lenguajes de consulta

La manera en la que se puede accesar a las bases puede ser de distintas maneras haciendo uso de diferentes lenguajes de consulta, a continuación se muestran las formas de acceso para las bases descritas arriba.

3.3.1. Acceso a `Medline`

La información de `Medline` está disponible para la mayoría de los propósitos. Su licencia esta bajo la *NLM* de los Estados Unidos o *Biblioteca Nacional de Medicina* por sus siglas en inglés.

La *Community of Science, Inc.(COS)* ofrece una poderosa herramienta via web, la cual es compilada por la NLM y publicada en la World Wide Web (www) mediante COS, *Medline* [26]. Los usuarios pueden elegir una de varias interfaces distintas deacuerdo al tipo de consulta, de entre ellas se mencionan las siguientes:

- Interface de búsqueda principal
- Búsqueda simple
- Búsqueda avanzada
- Búsqueda específica de revistas
- Búsqueda MEDLINE por términos MeSH

Esta interface se puede encontrar en la página `http://medline.cos.com`.

Otra forma de obtener la información de manera indirecta es haciendo uso del software *LingPipe*, el cual es un paquete de las librerías de *Java* para análisis linguístico.

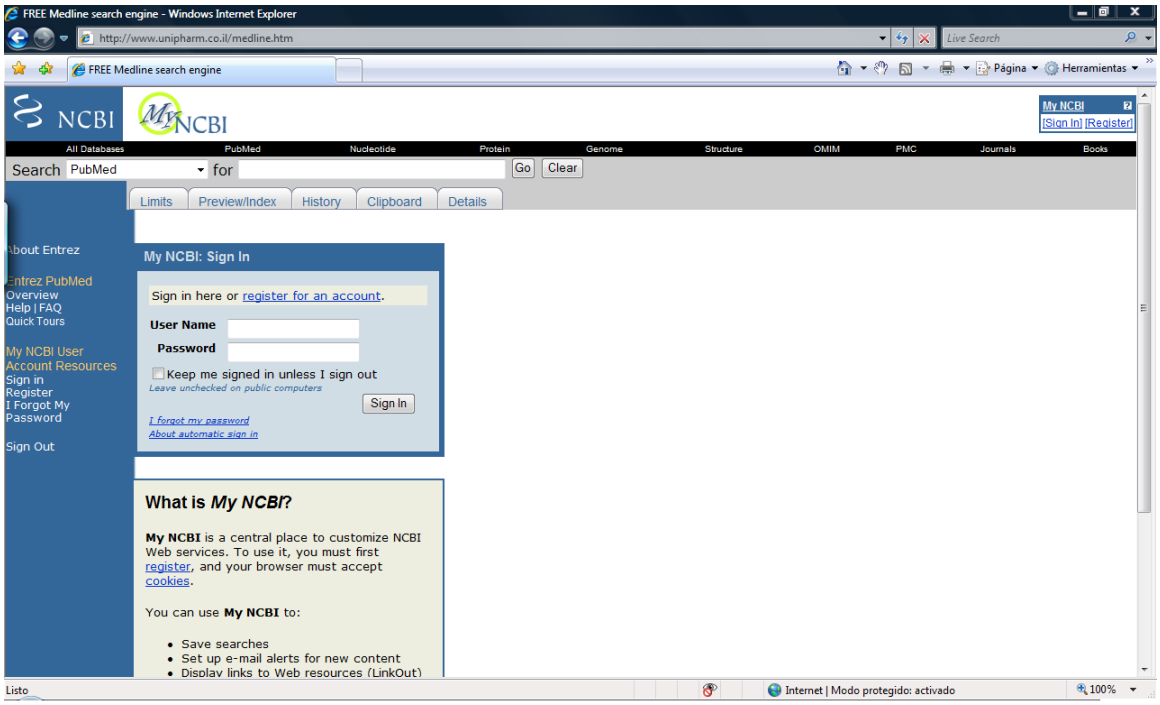
La informacón de NLM también puede ser descargada en XML, cada archivo XML contiene un conjunto de citas sobre un solo elemento, las cuales hacen referencia al contenido de cada elemento, para poder manipular dicha información se rquiere tener instalado tanto Java como LingPipe, para mayor información sobre esta configuración y métodos en Java consultar la página
`http://www.alias-i.com/lingpipe/demos/tutorial/medline/read-me.html`.

3.3.2. Acceso a `GenBank`

Los registros de los nucleótidos de GenBank se encuentran disponibles en las divisiones *CoreNucleotide*, *dbEST*, o *dbGSS* y pueden ser buscados en *Entrez* `http://www.ncbi.nlm.nih.gov/sites/gquery?tool=toolbar` juntos o de manera independiente. De manera independiente mediante *BLAST*.

La información de GenBank también puede ser descargada via FTP (`File Transfer Protocol`) en la página
`http://www.ncbi.nlm.nih.gov/Web/Newsltr/V15N2/GBrel.html`

Acontinuación se muestran otros posibles accesos mediante FTP:
`ftp://ftp.ncbi.nih.gov/genbank`
`ftp://bio-mirror.net/biomirror/genbank/`
`ftp://bio-mirror.jp.apan.net/pub/biomirror/genbank/ (Japon)`
`ftp://bio-mirror.kr.apan.net/pub/biomirror/genbank/ (Corea)`
`ftp://bio-mirror.sg.apan.net/biomirrors/genbank/ (Singapur)`



El acceso a `Medline`, requiere de la creación de una cuenta para el ingreso a los recursos contenidos en NCBI.

Figura 3.1: Acceso a Medline

El contenido de `GenBank`, es actualizado de manera periódica.

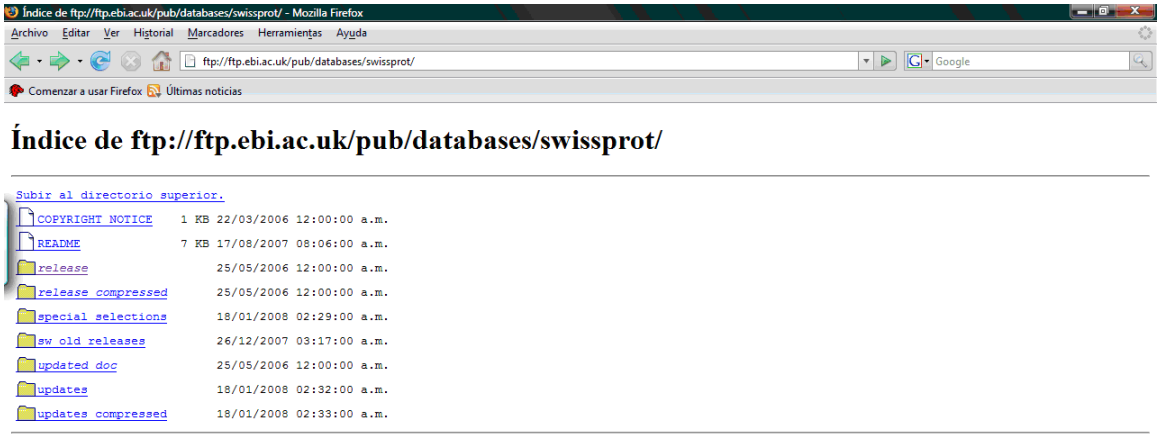
Figura 3.2: FTP GenBank

3.3.3. Acceso a `SwissProt`

La información contenida en SwissProt puede ser consultada en la página `http://ca.expasy.org/sprot/` o bien, al igual que en GenBank puede ser consultada mediante *BLAST* en la página
`http://ca.expasy.org/blast/`

Acontinuación se muestran otros posibles accesos mediante FTP:

`Australia: au.expasy.org`
`Bolivia: bo.expasy.org`
`Brasil: br.expasy.org`
`China: cn.expasy.org`
`Canada: ca.expasy.org`
`Corea: kr.expasy.org`
`Suiza: ftp.expasy.org`
`Taiwan: tw.expasy.org`
`USA: us.expasy.org`
`http://www.pir.uniprot.org/database/download.shtml`



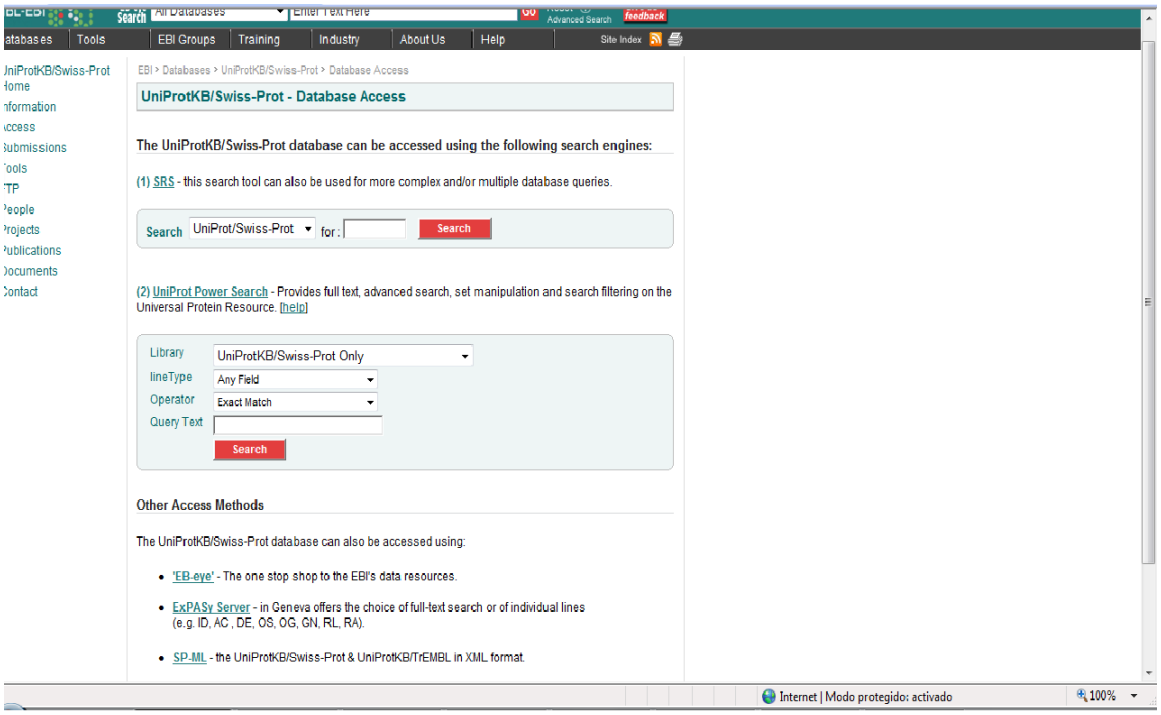
El acceso a `SwissProt`, puede ser mediante la descarga de diversos archivos a través de FTP.

Figura 3.3: FTP SwissProt

```
Sitio FTP:

   Organización   Swiss Institute of Bioinformatics
   Dirección      ftp.expasy.org (Suiza)
                  au.expasy.org (Australia)
                  br.expasy.org (Brasil)
                  ca.expasy.org (Canada)
                  cn.expasy.org (China)
                  kr.expasy.org (Corea del Sur)
   Directorio      /databases/swiss-prot


Nota:
Esta Base de Datos se encuentra actualmente disponible en:

/databases/uniprot/current_release/knowledgebase/complete.


A continuación, se describen los subdirectorios contenidos en /swiss-
prot:

+--swiss-prot--+
               |
               |--special_selections  Contiene diversos archivos
               |                      de un subconjunto taxonómico
               |                      específico ó ligados a una Base
               |                      de Datos en específico.
               |--sw_old_releases     Archivos 'tar' de versiones
               |                      previas de Swiss-Prot hasta la
               |                      versión 45.0.
               |
               +--updates             Archivos que contienen
               |                      actualizaciones semanales.
               |
               +--updates_compressed  Archivos 'gz' que continen las
                                      actualizaciones semanales.
Los siguientes subdirectorios han sido reemplazados por sus
correspondientes ubicaciones dentro del servidor FTP UniProt.

+--swiss-prot--+
               |
               |--release             Contiene archivos de la versión
               |                      actual de Swiss-Prot
               |
               |--release_compressed  Archivos 'gz' que continen la
               |                      versión actual de Swiss-Prot
               |
               +--sw_old_releases     Archivos 'tar' que contienen las
                                      versiones previas de Swiss-Prot.
```

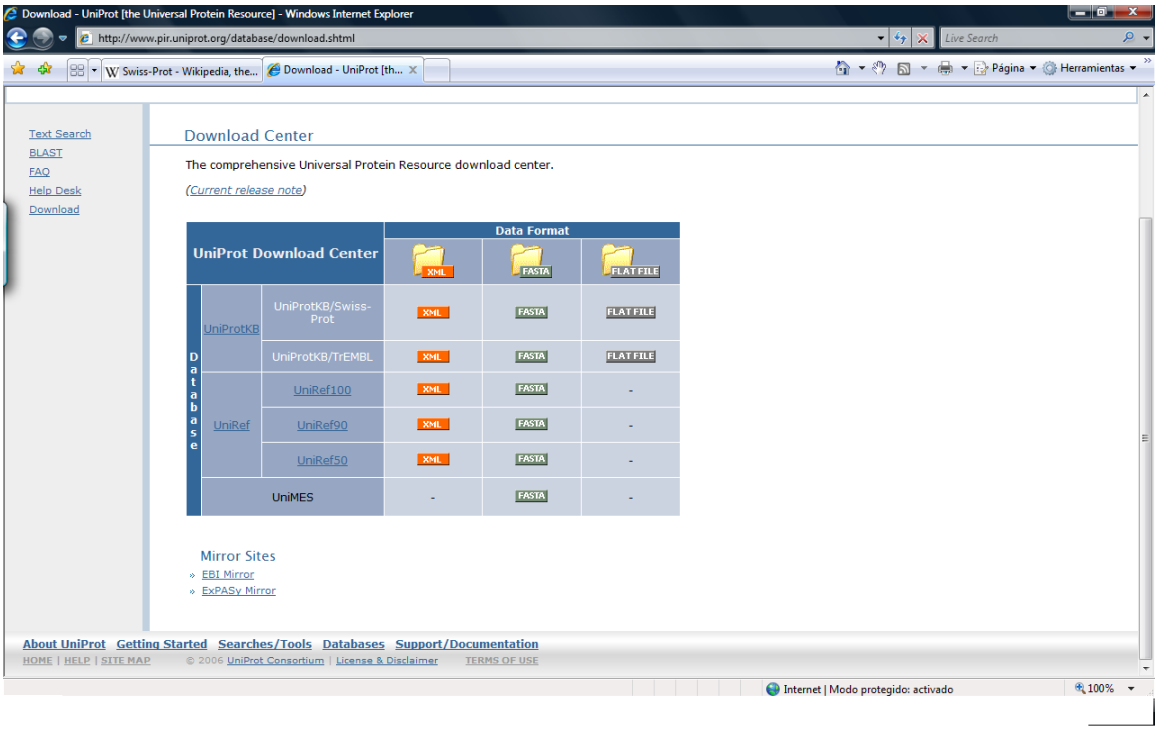
El contenido de `SwissProt`, se encuentra estructurado de la manera presentada en la parte de arriba.

Figura 3.4: FTP SwissProt



El contenido de `SwissProt`, también puede ser consultado a través del Web.

Figura 3.5: FTP SwissProt



Existen diversos sites en el Web en donde se puede descargar el contenido de `SwissProt`, en diversos formatos tales como XML, FASTA y Texto plano.

Figura 3.6: FTP SwissProt

Capítulo 4

Diseño de un sistema de recuperación de información

Un *sistema de información* (**SI**) es el sistema de personas, datos y actividades que procesan los datos y la información en una determinada organización, incluyendo procesos manuales y automáticos.

Este capítulo hace referencia al análisis y diseño y del nuevo sistema de información, para ello debemos conocer primero cuales son la herramientas de software que se adecúan mejor a las necesidades de éste sistema, y más adelante veremos el diseño del nuevo sistema.

4.1. Herramientos de software

Hoy en día la cantidad de herramientas de software para realizar un sistema informático es amplia, y su selección dependerá de varios factores, entre ellos podemos mencionar los siguientes: recursos con los que se cuenta, requerimientos del sistema a realizar, conocimiento previo de la herramienta, tiempo con el que se cuenta para realizar el sistema, entre otros. Se hablará sobre las herramientas utilzadas para el desarrollo de este proyecto: lenguaje de programación y sistema manejador de bases de datos.

4.1.1. Lenguaje de programación

Un *lenguaje de programación* es un lenguaje que puede ser utilizado para controlar el comportamiento de una máquina, particularmente una computadora. Consiste de un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones.

El lenguaje de programación seleccionado para esta aplicación es Java, el cual es un lenguaje de programación de alto nivel con que se pueden escribir tanto programas convencionales como para Internet.

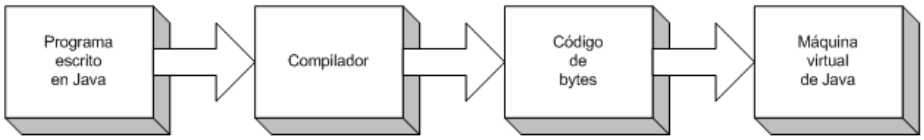


Figura 4.1: Esquema de procesos internos de Java

Una de las ventajas significativas de Java sobre otros lenguajes de programación es que es independiente de la plataforma, tanto en código fuente como en binario. Esto quiere decir que el código producido por el compilador Java puede transportarse a cualquier plataforma que tenga instalada una máquina virtual Java y ejecutarse. Pensando en Internet esta característica es crucial ya que esta red conecta ordenadores muy distintos. En cambio, C++, por ejemplo, es independiente de la platafomra sólo en códifo fuente, lo cual significa que cada plataforma diferente deber proporcionar el compilador adecuado para obtener el código máquina que tiene que ejecutarse.
Según lo expuesto, Java incluye dos elementos: un *compilador* y un *intérprete*. El compilador produce un código de bytes que se almacenan en un fichero para ser ejecutado por el intérprete Java denominado *máquina virtual java* [27].Este proceso se muestra en la figura 4.1

Los códigos de bytes de java son un conjunto de instrucciones correspondientes a un lenguaje máquina que no es específico de ningun procesador, sino de la máquina virtual de java [27].

4.1.2. Manejador de bases de datos

Decidir cual manejador de bases de datos utilizar depende de los requerimientos del sistema a realizar pues cada manejador tiene sus cualidades que lo convierten en el más adecuando según el tipo de proyecto a desarrollar. Para este proyecto se eligió PostgresSQL debido a que es el servidor de bases de datos de código abierto más potente que existente.

PostgreSQL es un poderoso sistema de bases de datos relacional de código abierto, tiene más de quince años de desarrollo contínuo y su arquitectura ha ganado una gran reputación por ser confiable, mantener integridad de datos y su sencillez para realizar correciones en los datos. Puede ser utilizado en la mayoría de los sistemas operativos, incluyendo Linux, UNIX(AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), y Windows. Tiene la capacidad de almancenar objetos largos binarios, incluyendo imagenes, sonido o video. Tiene interfaces de programación para C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC, entre otros. Es altamente escalable en cuanto a cantidad de datos que puede manejar y al número de usuarios que puede alojar. En la tabla 4.1.2 se mencionan las limitaciones de PostgreSQL [28].

Dado que se requiere de la manipulación de grandes cantidades de información, de una aplicación que funcione sobre cualquier plataforma convencional, y que esté de-

Limitante	Valor
Tamaño máx.de bases de datos	*Ilimitado*
Tamaño máx. tabla	32 TB
Tamaño máx. renglon	1.6 TB
Núm. máx. renglones por tabla	*Ilimitado*
Tamaño máx. columnas por tabla	250-1600 dependiendo del tipo de dato por columna
Núm. máx. índices por tabla	*Ilimitado*

Tabla 4.1: Capacidad de almacenamiento de PostgreSQL

sarrollada en código abierto; utilizando licencia GNU, se encontró que el manejador de bases de datos que se adecúa mejor a los requerimientos es PostgreSQL.

Una vez que se ha realizado el análisis sobre las herramientas de sofware que mejor se adecúan para este proyecto, veremos el modelado del sistema en la sección 4.3.

4.2. Estructura del nuevo sistema

El Sistema de Información o SI por sus iniciales está dividido en cinco módulos:

- Módulo de eliminación de redundancia
- Módulo de visualización de subtablas
- Módulo de dependencias funcionales
- Módulo de composición de tablas
- Módulo para búsqueda de registros

A continuación se hará una descripción de cada módulo descrito arriba donde se le denominará *usuario* a toda persona que interactúe con el SI.

4.2.1. Descripción de módulos del sistema

- **Módulo de eliminación de redundancia**

  Este módulo se encarga de migrar la información de un archivo de Microsoft Excel al manejador de bases de datos PostgreSQL. Los datos contenidos en la hoja de Excel son reescritos a un archivo de texto insertándoles a su vez las instrucciones en SQL para crear un *script* útil tanto para que el sistema realice automáticamente la migración de la información, como para el usuario en caso de querer migrar su información a otro manejador de bases de datos. Para que el *script* sea válido y pueda ser ejecutado en otros manejadores fué necesario

realizar un análisis de depuración de los datos, esto es, un análisis sintáctico eliminando caracteres como: ,;′"_, esto con la finalidad de evitar conflictos con sentencias en *SQL*. Una vez que se crea el *script*, éste es ejecutado por PostreSQL y finalmente, la información es migrada automáticamente.

Ahora bien, para eliminar la redundancia en los datos originales, se creó una llave única por registro, la cual se crea a partir de la concatenación de cada uno de los datos contenidos en cada campo, lo cual permite almacenar únicamente registros con datos nuevos sin importar si el dato es diferente en un solo campo o más. El diagrama que describe todo el proceso de eliminación de redundancia se muestra en la figura 4.2.

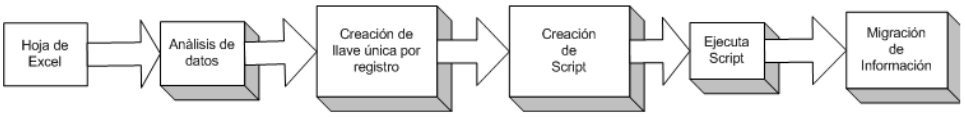


Figura 4.2: Diagrama del módulo de eliminación de redundancia

- **Módulo de visualización de subtablas**

  Este módulo se encarga de mostrar al usuario únicamente aquellos datos en los que el usuario está interesado por lo que se lo presentan de manera dinámica cada uno de los campos que contiene la base de datos elegida previamente por el usuario y que también ha sido analizada por el SI y por lo tanto dada de alta en el sistema. Además de tener la facilidad de elegir que campos desea ver, el usuario también puede elegir la cantidad de registros que desea consultar. El diagrama que describe todo el proceso descrito se muestra en la figura 4.3.

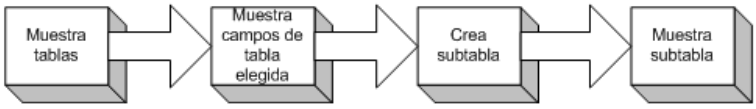


Figura 4.3: Diagrama del módulo de visualización de subtablas

- **Módulo de dependencias funcionales**

  Este módulo se encarga de obtener las dependencias funcionales a partir de una tabla y sus campos seleccionados por el usuario. El análisis se lleva a cabo de cuatro maneras distintas:

  1. Obteniendo dependencias funcionales mediante el uso de *empatamiento de patrones*, es decir, que los datos sean iguales tanto en el registro r_1 y r_2 y por orden de *frecuencia descendente*, esto es, la forma en la que los datos se ordenan de acuerdo al orden de prioridad de los atributos dado a partir de los datos de un atributo en el cual la cantidad de información es más

repetitiva entre registros en el mismo campo que en los demás, hasta el atributo con mayor cantidad de información diferente en sus registros.

2. Obteniendo dependencias funcionales mediante el uso de empatamiento de patrones y por orden de atributos dado por el usuario.

3. Mediante funciones de similitud, donde el usuario proporciona el umbral para el cual un dato sobre un campo o más se considera dentro del rango de análisis, y a partir de un ordenamiento de atributos por frecuencia descendente.

4. Mediante funciones de similitud, donde el usuario proporciona el umbral para el cual un dato sobre un campo o más se considera dentro del rango de análisis, y a partir de un ordenamiento de atributos dado por el usuario.

- **Módulo de composición de tablas**. Mediante este proceso el usuario puede manipular tablas, esto es, dados los nombre de dos tablas dadas de alta previamente en el sistema, el usuario puede realizar la unión de ambas tablas para hacer más rica la consulta de dicha base. En caso de que ambas tablas contengan los mismos atributos este proceso une automáticamente ambas tablas, en caso contrario se obtienen primero los atributos de una tabla no contenidos en la segunda y viceversa, de tal forma que dicho conjunto de atributos no pertenecientes a ambas tablas simultáneamente son eliminados, el resultado será nuevamente la unión de ambas tablas que contengan los mismos atributos.

- **Módulo para búsqueda de registros**. Este módulo como su nombre lo indica, sirve para consultar información sobre una tabla elegida por el usuario, se le pregunta al usuario sobre lo que más conoce propocionandole una serie de opciones sobre la cual puede hacer la consulta, ya sea por el nombre del gen, símbolo, por su descripción o bien, por alguna de las características con valor número. Se requiere del valor de la cota inferior y superior de similitud (rango de similitud) mediante el cual se va a determinar la exactitud de los resultados arrojados por el sistema, también se requiere el dato a ser comparado durante la búsqueda. Ahora bien, una vez proporcionado el rango y el dato sobre el cual se va a realizar la búsqueda, el sistema arroja todos aquellos datos similares al dato proporcionado que están dentro del rango de similitud dado, después se generan de manera dinámica cajas para ser marcadas por el usuario, en las cuales únicamente se marcarán aquellas en las que la consulta o dato sea relevante, de tal manera que el siguiente paso para el sistema es el de proporcionar la información del(os) registro(s) completo(s) sobre el cual el dato fué marcado.

4.3. Diagramas en Lenguaje Unificado de Modelado (UML) del nuevo sistema

UML es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construír y documentar un sistema de software. UML utiliza como términos técnicos *actores* y *casos de uso*. Un *actor* es un usuario del sistema *que tiene un rol particular*. En realidad un actor puede ser un sistema externo que es como un usuario desde el punto de vista de nuestro sistema: el punto crucial es que se está diseñando alguien o algo externo al sistema, que interactúa con nuestro sistema y que puede hacerle peticiones [29].

4.3.1. Diagramas de casos de uso

Un *caso de uso* es una tarea que un actor necesita ejecutar con la ayuda del sistema [29].

Ahora bien, en la figura 4.4 se puede ver el sistema con sus actores y los precesos principales que en él intervienen, el único actor es el usuario esto es debido a que cualquier investigador del departamento puede hacer uso del sistema, esto con la finalidad de que toda persona que se encuentre realizando alguna investigación pueda tener la facilidad de hacer más rapidas sus busquedas.

El sistema completo se encuentra dividido en siete procesos:

1. Buscar datos similares
2. Encontrar dependencias funcionales
3. Crear composición de tablas
4. Eliminar redundancia
5. Visualizar subtablas
6. Seleccionar registro
7. Proporcionar datos

De los cuales las funcionalidades del primer proceso al quinto han sido descritas en la subsección 4.2.1 a manera de módulos, el sexto es parte del proceso “Buscar datos similares”, esto es debido a que se pueden seleccionar registros completos una vez que se haya hecho una busqueda de información por algun dato y su rango de similitud, este proceso se describe mejor de manera grafica mediante los diagramas de la subsección 4.3.2. En cuanto al último proceso, “Proporcionar datos”, se refiere a la interacción del usuario con el sistem de tal forma que el usuario se va a dedicar a proporcionar datos necesarios para la ejecución de las tareas del propio sistema, aunque no es un proceso que como veremos mas adelante deba ser detallado como el

resto, es parte importante del diseño pues podemos detallar el tipo de interacciones y acciones usuario-sistema.

Ahora bien, los datos necesarios para cada proceso contenido en diseño global del sistema, ver figura 4.4, se muestran en las figuras 4.5, y 4.6. Donde para la figura 4.5 el caso de uso de "***Visualizar subtablas***" requiere como datos de entrada: el nombre de la tabla a visualizar, la selección de atributos que sean de interes para el usuario y la cantidad de registros que se requieran para una consulta óptima. Para "***Buscar datos similares***" se requieren los datos: nombre de la tabla de donde se va a realizar la búsqueda, la cota tanto inferior como superior de similitud para saber sobre que rango de similitud se van a arrojar los resultados, y el dato a buscar así como al tipo de atributo al que pertenece. Con esto el sistema desplegará una serie de datos sobre un solo campo, los cuales van a ser similares al dato proporcionado en el rango de similitud dado. En caso de que algún o algunos datos sean de interés para el usuario, y solo hasta ese momento, se ocupará entonces el " ***Seleccionar registro***" el cual recibe como parámetros de entrada los resultados arrojados por el "proceso de búsqueda de datos similiares", mencionado anteriormente, y que hayan sido seleccionados por el usuario como datos de interés, como salida nos dará la información de los registros completos sobre los datos ya seleccionados.

Por otra parte la figura 4.6 hace referencia en primer lugar al de"***Encontrar dependencias funcionales***", el cual, como se muestra en la figura, recibe como datos de entrada: nombre de la tabla a analizar, número de registros, atributos de interés, el tipo de búsqueda de dependencia funcional; ya sea por *empatamiento de patrones*; es decir por igualdad sobre datos, o bien por búsqueda de dependencias funcionales mediante el uso de funciones de similitud. En caso de que se haya seleccionado el segundo tipo de búsqueda se deberá dar la cota inferior y superior de similitud sobre cada atributo seleccionado. También es importante mencionar que toda búsqueda de dependencias funcionales está relacionada de manera directa con la forma en la que los datos son ordenados, de tal forma entonces se pide también el orden que van a llevar los atributos sobre los datos, ya sea que por orden de frecuencia descendente (atributos con mayor número de repeticiones en sus datos hasta el atributo con mayor cantidad de datos distintos) o bien, por un orden arbitrario por el usuario.

El siguiente caso de uso a describir es "***Crear composición de tablas***", la manera en la que funciona es de la siguiente: recibe como datos de entrada los nombres de dos tablas para crear la composición, y el atributo mediante le cual ambas tablas van a ser relacionadas una de la otra. Esto dará como resultado la compaginación de tablas hechas por distintos usuarios haciendo mas rica la búsqueda. La forma en la que procede es mediante la instrucción `INNER JOIN USING(atributo)` de SQL.

El último caso de uso, y muy importante, es "***Eliminar redundancia***", este caso de uso surge de la gran cantidad de registros repetidos contenidos en las tablas con las que trabajan los investigadores de Biología Celular. Los parámetros de entrada son: la ruta en la que se encuentra la tabla en Microsoft Excel, el nombre del script que se va a crear, así como la ruta donde se desea crear; éste tiene la finalidad de proporcionar un respaldo de la información contenida en la tablas de los usuarios

con la facilidad de migrar dichos datos a otro manejador de bases de datos para poder ser manipulados de la manera que mejor convenga.

4.3.2. Diagramas de actividades

Los *diagramas de actividades* describen cómo se coordinan las actividades, pueden utilizarse para indicar cómo podría implementarse una operación . Un diagrama de actividad es particularmente útil cuando se sabe que una operación tiene que alcanzar un número de cosas distintas, y se quiere modelar cuáles son las dependencias fundamentales entre ellas, antes hay que decidir en qué orden se van a hacer [29].

Ahora bien, a continuación se hará una descripción más detallada de cada caso de uso mediante sus *diagramas de actividades*.

"***Encontrar dependencias funcionales***", aparece en la figura 4.7, este proceso consiste, dado el nombre de la tabla a analizar, sus atributos de interés y la cantidad de registros, pregunta si la búsqueda de dependencias funcionales se va a realizar por *empatamiento de patrones*, en caso de que la búsqueda sea llevada a cabo de esa manera, pregunta por el orden los atributos, si los atributos se ordenan de manera descendente deacuerdo al número de apariciones de datos repetidos, entonces el sistema devuelve las dependencias funcionales sobre los datos proporcionados al principio, en caso de que el orden no sea descendente se le pregunta al usuario por el orden de los atributos. Para el caso en el que el usuario desee una búsqueda de dependencias mediante funciones de similud, se le pedirá que proporcione la cota inferior y superior de similitud mediante el cual se compararán todos los datos sobre el atributo y dato dado. Posteriormente pedirá el orden de los atributos y seguirá con los mismos pasos que en la búsqueda de dependencias por empatamiento de patrones.

"***Crear composición de tablas***", aparece en la figura 4.8, recibe como parámetros los nombres de dos tablas mediante las cuales se realizará la composición, procederá a verificar si existe igualdad entre algunos o todos los atributos pertenecientes a ambas tablas, en caso de que sean iguales el siguiente paso es unir las tablas y finalemnte desplegarlas al usuario. En caso de que existan atributos que no pertenezcan a alguna de las dos tablas se procede solo a realizar la unión de ambas tablas sobre los atributos iguales.

"***Eliminar redundancia***", aparece en la figura 4.9, recibe la ruta del archivo que contiene la tabla en *Excel*, verifica si la ruta es correcta, en caso de serlo realiza la depuración de los datos mediante la creación de una llave única por registro, la cual es creada cuando al menos uno de los datos sobre uno o mas atributos es distinto sobre un registro completo. Después crea un "script" adicionandole a cada registro su llave, ejecuta el script, si alguna de las líneas en el archivo del script marca error a la hora de ser ejecuta por el manejador de bases de datos (Postgres) indica que el registro ya ha sido almancenado en la base de datos y salta a la siguiente línea del archivo para ser ejecutada. Una vez realizado lo anterior se tiene como resultado una nueva tabla dada de alta en el sistema libre de redundancia en registros y un script que sirve como respaldo para el usuario.

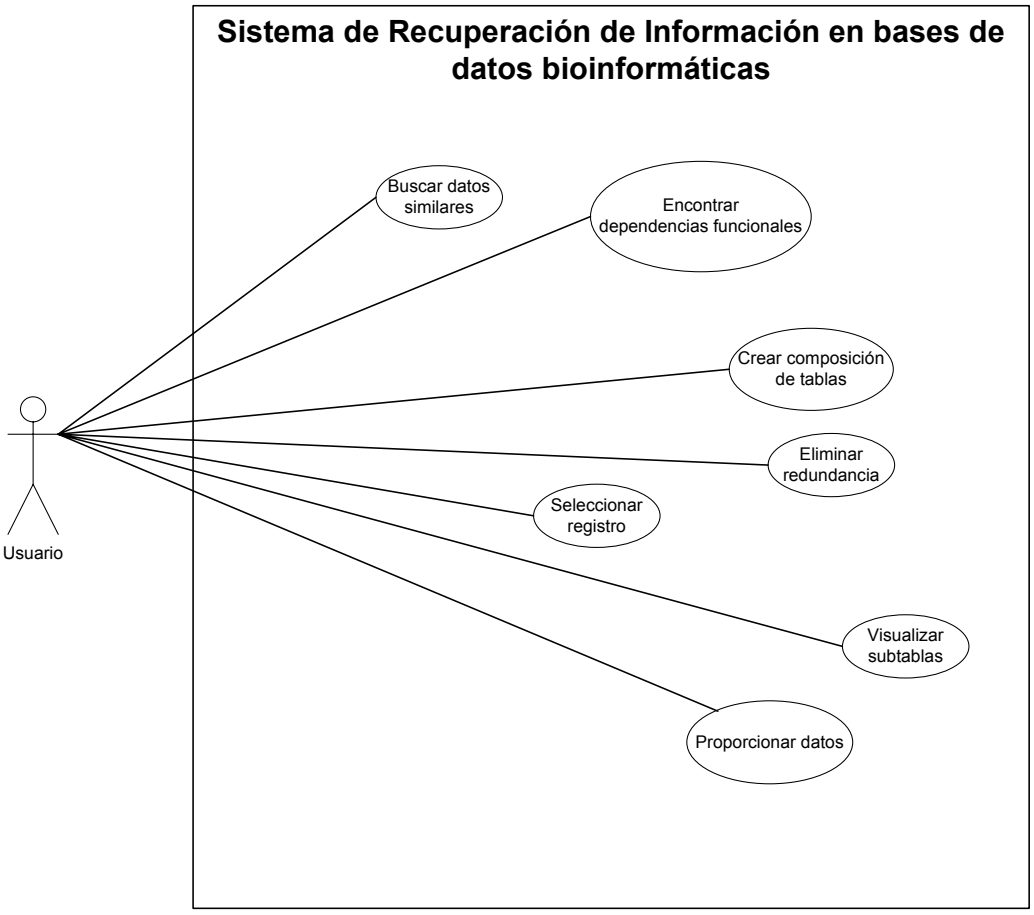


Figura 4.4: Diagrama de casos de uso y actores del sistema desarrollado.

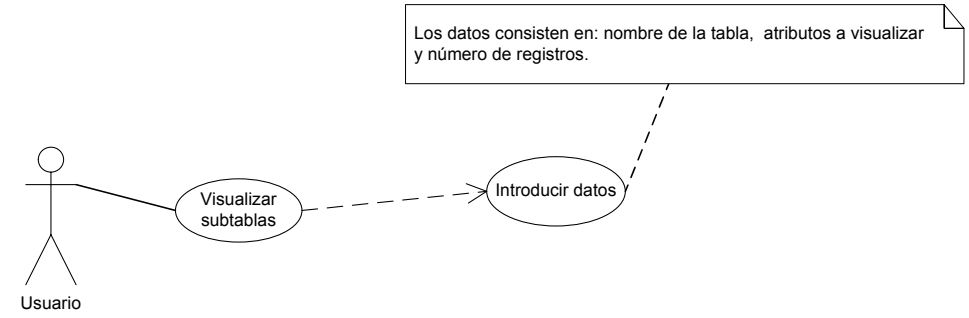
"***Seleccionar registro***", aparece en la figura 4.10, esta actividad depende directamente de los resultados de la "Búsqueda de datos similares", una vez que estos han sido arrojados por el sistema se seleccionan los datos interesantes para el usuario, realiza la consulta del registro completo sobre el dato seleccionado y la muestra al usuario.

"***Buscar datos similares***", ver el primer diagrama de la figura 4.11, recibe como parámetros el nombre de la tabla, nombre del atributo sobre el cual se va a realizar la búsqueda, el dato sobre el cual se va a realizar la comparación, y finalmente las cotas inferior y superior mediante el cual los datos comparados van a estar dentro de ese rango. Dependiendo del atributo seleccionado se aplica la funcion de similitud a evaluar. A continuación se menciona la función aplicada para cada atributo. La forma en la que fueron implementadas se verán mas adelante en el capítulo 5.

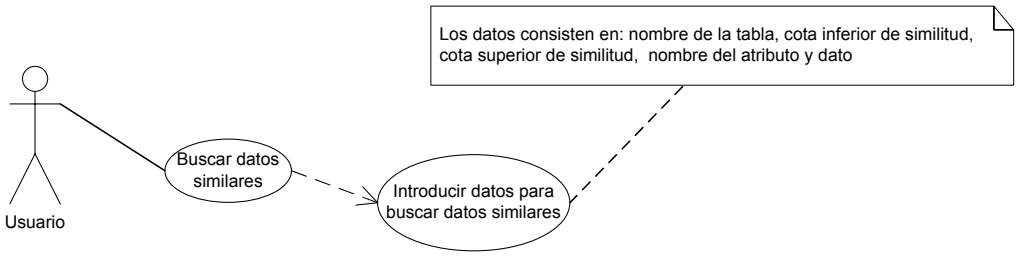
1. ID
2. Symbol-------(simiSymbol)
3. Genename ------(geneName)
4. oligoid ------(oligoID)
5. humanortholog-------(simiSymbol)
6. mouseorthologs -------(simiSymbol)
7. GBaccession -------(simiSymbol)
8. DescriptionRGD ------(geneName)
9. Function1k ------(geneName)
10. GObiologicalprocess ---------(simiGO)
11. GOmolecularfunction ---------(simiGO)
12. GOcelllocation --------------(simiGO)
13. CHROMOSOME -------- (simiChromosome)
14. FISHBAND -------(simiSymbol)
15. ENT1NLratio1k --------
16. ENT5NLratio1k ---------------
17. HCCNLratio1k ------------------------
18. pvalueENT1NL1k --------------------------
19. pvalueENT5NL1k ----------------------------(simiValores)
20. pvalueHCCNL1k ------------------------------
21. FDR1k ----------------------------------
22. Venn1k ---------------------------------
23. ENT1NLratio28k ------------------------
24. HCCNLratio28k ------------------------
25. HCCNTratio28k ----------------------
26. pvalueENT1NL28k----------------
27. pvalueHCCNL28k --------------
28. pvalueHCCNT28k -----------
29. FDR28k -----------------
30. Venn28k----------------

Descripción general de atributos. La lista de atributos descrita arriba corresponde a una base de datos obtenida de la tesis doctoral del Dr.Julio Isael Pérez Carreón en el laboratorio del Dr. Saúl Villa Treviño. Los datos corresponden a una matriz de 29009×30 atributos. Esta tabla se generó del análisis de expresión de genes (transcriptoma) por microarreglos de ADN, en un modelo de inducción de cáncer de hígado experimental en rata. La información experimental fué enriquecida con información de los genes proveniente del Rat Genome Database (RGD). Esta base de datos fué seleccionada como modelo de manipulación y procesamiento de datos de la aplicación bioinformática de la presente tesis.

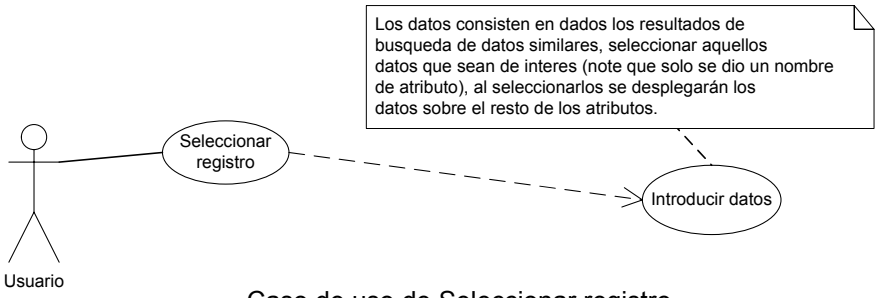
"***Visualizar subtablas***", ver el segundo diagrama de la segunda figura 4.11, como se puede ver, recibe como datos de entrada el nombre de la tabla a visualizar, los atributos de interés y la cantidad de registros a visualizar, el sistema crea una tabla con nombres de columnas de los atributos seleccionados, y del tamaño del número de registros, la cual es rellenada con datos contenidos en la base de datos dada de alta con el nombre de la tabla elegida, lo cual evita redundancia en la información y permite al usuario solo ver los datos que le son útiles en ese momento.



Caso de uso de Visualizar subtablas

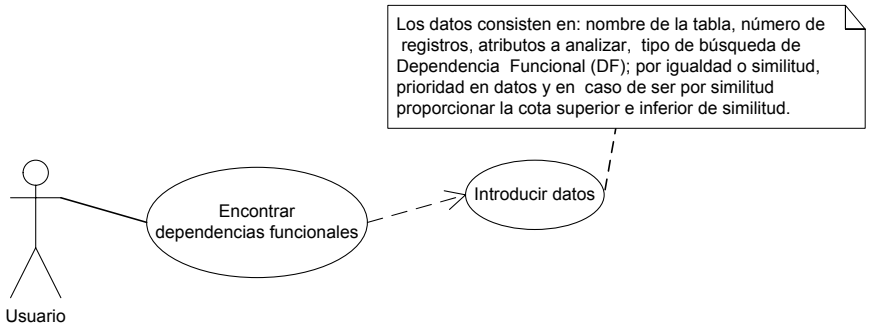


Caso de uso de Buscar datos similares

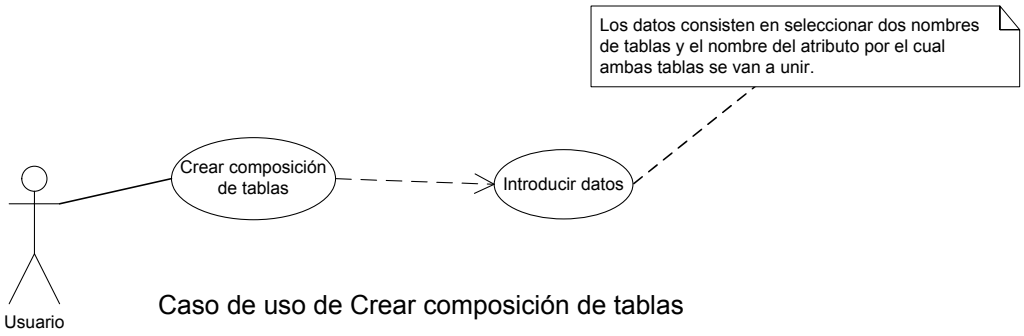


Caso de uso de Seleccionar registro

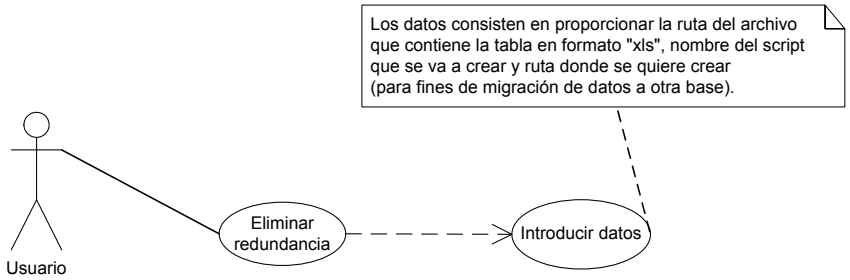
Figura 4.5: Diagrama de casos de uso: visualizar datos seleccionados y buscar datos dadas las cotas de similitud.



Caso de uso de Encontrar dependencias funcionales

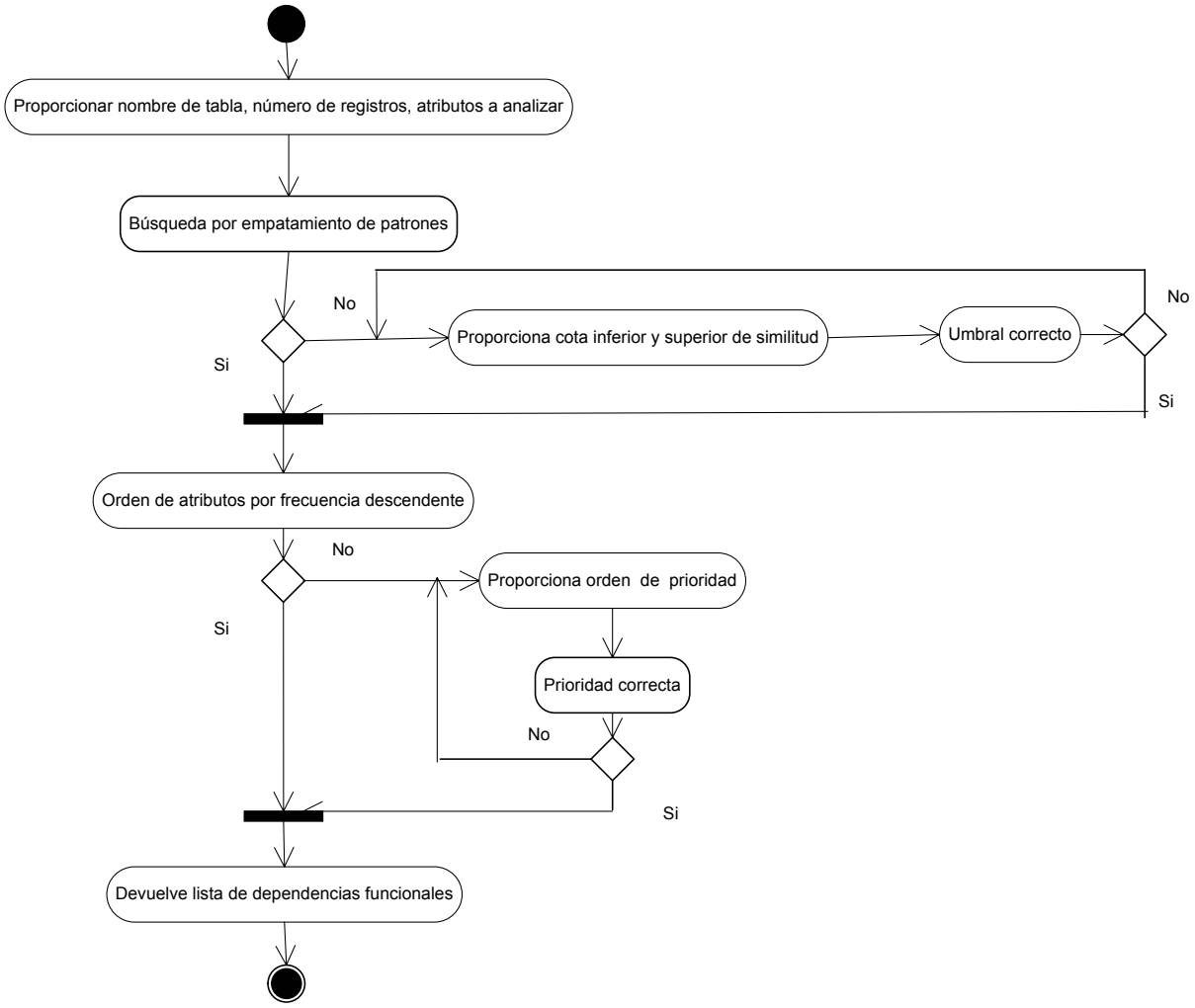


Caso de uso de Crear composición de tablas



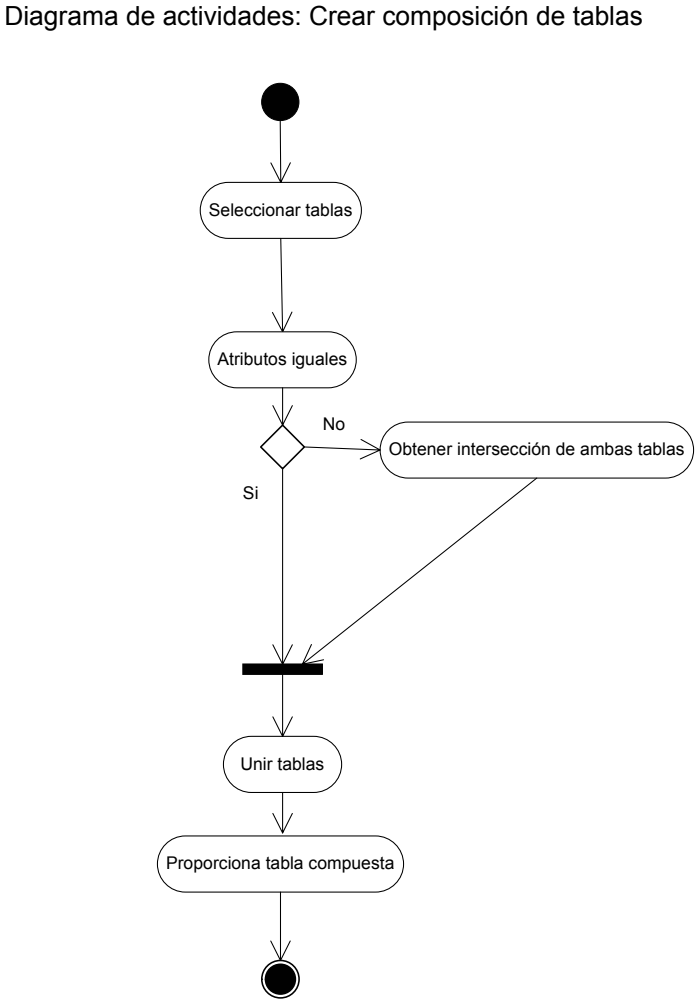
Caso de uso de Eliminar redundancia

Figura 4.6: Diagrama de casos de uso para la búsqueda de dependencias funcionales, composición de tablas y eliminación de redundancia.



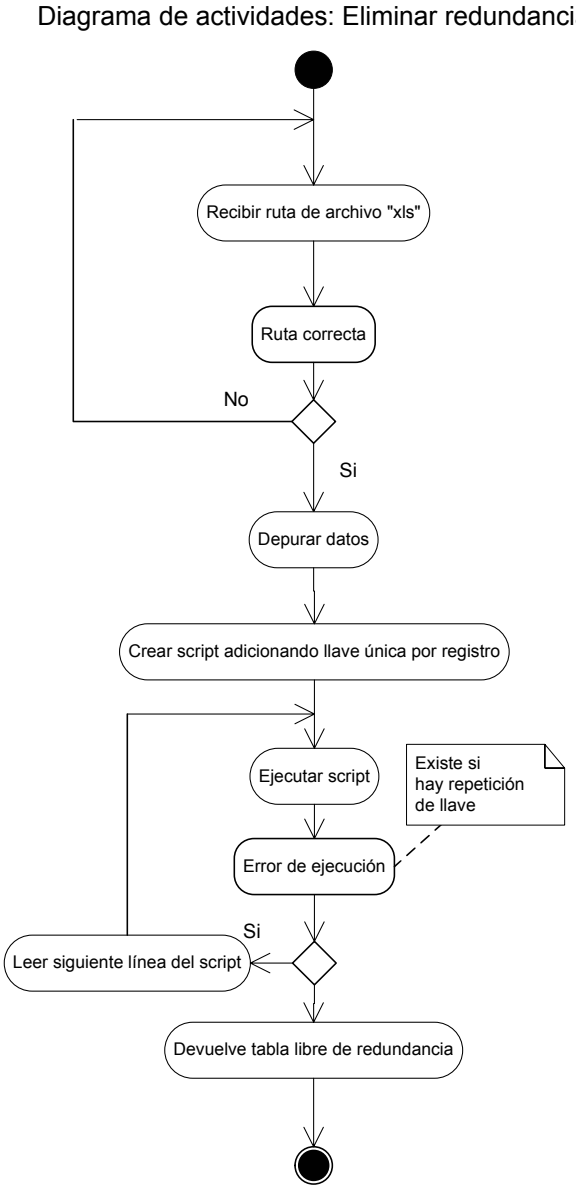
Flujo de datos para la búsqueda de dependencias funcionales dados los atributos a analizar, tipo de búsqueda y orden de atributos.

Figura 4.7: Diagrama de actividades para encontrar las dependencias funcionales.



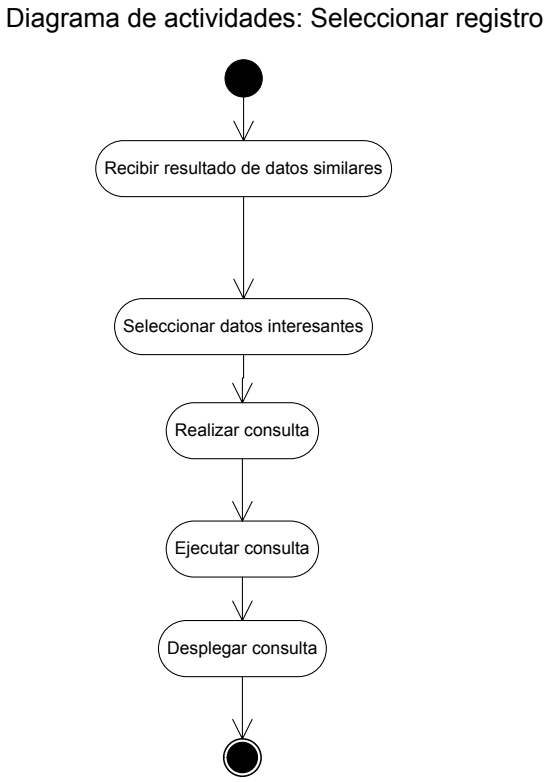
Se realiza la búsqueda de atributos en común y la unión entre ambas mediante uno de los atributos encontrados.

Figura 4.8: Diagrama de actividades para crear la composición de tablas.



Con una tabla en formato “xls” se crea un “script”, se ejecuta y se obteniene la migración de la información.

Figura 4.9: Diagrama de actividades para eliminar redundancia de registros.



Se requiere como proceso previo la búsqueda de datos similares.

Figura 4.10: Diagrama de actividades para seleccionar registros.

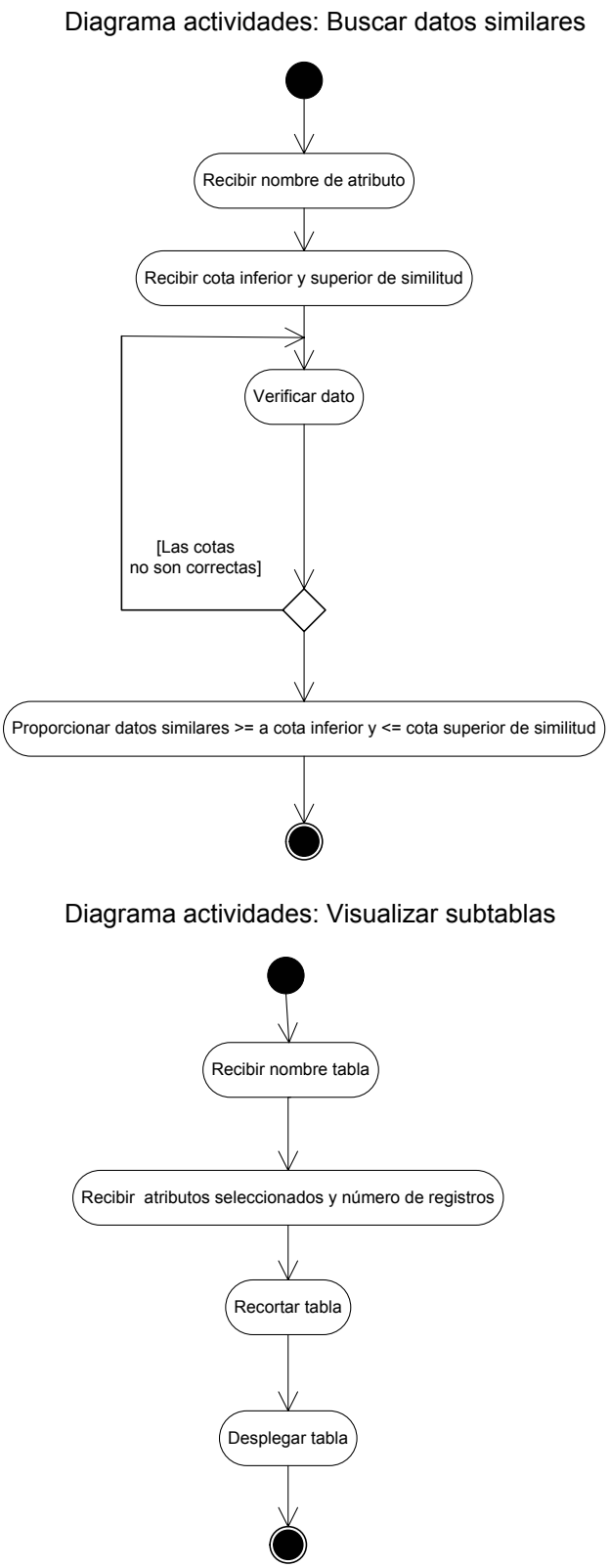


Figura 4.11: Diagrama de actividades para búsqueda de datos similiares y visualizar solo los campos seleccionados.

Capítulo 5

Implementación del nuevo sistema

Todo sistema informático está conformado por varios procesos que se ejecutan a medida que son requeridos por el usuario. Llamamos *implementación de un sistema informático* a la manera en la que dichos procesos son construídos o programados y a la lógica para su creación.

5.1. Algoritmos para explotación de información

Un **algoritmo** es una lista de instrucciones para transformar una determinada *entrada* en una correspondiente *salida*. Puede haber algoritmos de diferentes tipos, dependiendo de su proceder en todo estado de la computación:

deterministas la configuración siguiente y el siguiente paso a ejecutar están bien determinados a partir de la configuración y el paso actual,

no-deterministas o **indeterministas** existen varias (incluso ninguna) configuraciones a asumir el rol de la configuración siguiente y, consiguientemente, la siguiente instrucción puede ser cualquiera de las asignadas a la configuración siguiente,

probabilistas son casos particulares de los no-deterministas, donde la configuración siguiente se elige en función de la configuración actual y del valor de una variable aleatoria, o

cuánticos la transición a una siguiente configuración se hace considerando transformaciones unitarias en estados de superposición.

[30]

Llamamos *algoritmos para explotación de información* a aquellos que sin importar su configuración son útiles para el análisis de datos y en conjunto contribuír a la búsqueda de información relevante. Los algoritmos desarrollados para la búsqueda de dicha información para este proyecto están pensados a partir de la resolución de los siguientes problemas:

1. Dados $B,C \subset A$, $B \cap C = 0$, decidir si $B \rightarrow C$ vale en R.

2. Encontrar dos conjuntos $B,C \subset A$: $B \rightarrow C$ vale en R.

Entrada. $R,B,C \subset A$, $B \cap C \Longrightarrow 0$
Salida. $\begin{cases} 1 \text{ si } B \rightarrow C \text{ vale en R} \\ 0 \text{ en otro caso} \end{cases}$
Procedimiento. AP1

```
(1) m := NumReg(R), Currtest:= true, i = 1 ;
    while(Currtest && i<m) do
      {
       prefijo := R[i; B]; sufijo := R[i; C]; j:= i + 1 ;
        while(Currtest && j<m) do
       {
            if R[i; B] == prefijo then
           {
            Currtest := (sufijo ==R[j, C]);
            if(Currtest)
                return 1 ;
            else
                  return x,y ;
           }
       }
      }.
```

Tabla 5.1: Algoritmo para búsqueda de dependencias funcionales por empatamiento de patrones.

3. Encontrar todos los conjuntos $B,C \subset A$: $B \rightarrow C$ vale en R.

El algoritmo de solución al primer problema se encuentra en la tabla 5.1

El algoritmo propuesto para dar solución al segundo problema se muestra en la tabla 5.2.

Ahora bien, para resolver el problema de encontrar todos los conjuntos $B,C \subset A$: $B \rightarrow C$, esto es, encontrar todos los conjuntos de atributos que dependen de otro conjunto de atributos, se aplicaron dos algoritmos principales: *algoritmos para búsqueda de depencias funcionales por empatamiento de patrones* y *algoritmos para búsqueda de dependencias funcionales mediante funciones de similitud*. En la subsección 5.1.1 se dará una explicación más detallada sobre los algoritmos diseñados para la realización de este trabajo.

Ahora bien, la manera en la que se realiza la búsqueda de dependencias funcionales mucho depende de la forma en la que los datos se encuentren ordenados; independientemente del tipo de búsqueda de dependencia funcional, es por ello que se pensó en dos criterios distintos de ordenación. El primer criterio está dado deacuerdo a las frecuencias de aparición de distintos datos sobre cada atributo, en donde se hace un conteo sobre el número de datos distintos sobre un solo atributo; esté conteo se realiza para cada uno de los atributos, determinando así cual es el atributo que contiene la

Entrada. R, A donde A contiene el conjunto de atributos a analizar
Salida. B,C : $B \rightarrow C$ vale en R
Procedimiento. AP2
(1) Finish:=False ; B:= $\{A_1\}$;
(2) while(!Finish) do
{
(3) $C = A - B$;
Aplicar AP1(B,C) hasta obtener una respuesta α
{
if(α)
{
Devolver B,C;
Exit;
Finish = $True$;
}
else
{
Usar la pareja (r_1, r_2) que viola la regla $B \rightarrow C$
Se debe encontrar el primer atributo A_j tal que $r_1[A_j] \neq r_2[A_j]$;
$B := B \cup \{A_j\}$
}
}
}

Tabla 5.2: Algoritmo para búsqueda de dependencias funcionales obteniendo atributos que violan la dependencia dentro de conjuntos.

mayor cantidad de datos tistintos, cual es el segundo atributo que le sigue con mayor número de datos distintos y así sucesivamente hasta encontrar el atributo con menor cantidad de datos dintintos. Con éste conteo se puede entonces crear una ordeanción de manera descendente sobre los datos, es decir, ordenar los datos deacuerdo a los atributos de mayor a menor frecuencia de repetición sobre su información. El segundo criterio de ordenación está basado en la prioridad sobre atibutos que el usuario del sistema le quiera dar, obviamente siempre y cuando los atributos a ordenar se encuentren dentro de la o las bases de datos a analizar.

5.1.1. Algoritmos para búsqueda de depencias funcionales

En esta subsección realizamos una descripción detallada sobre el algoritmo diseñado para la búsqueda de dependecias funcionales, por empatamiento de patrones y por uso de funciones de similitud, véase la tabla 5.3. Este algoritmo como su nombre lo indica encuentra las dependencias funcionales utilizando un *backtraking* que almacena conjuntos de atributos que servirán como *antecedentes* de los cuales se determinará si cada conjunto depende de otro conjunto de atributos *consecuentes*, para ello realiza una poda de conjuntos generados por cada iteración, esto con la finalidad de eliminar *superconjuntos* es decir grupos de atributos que en iteraciones anteriores contengan conjuntos de atributos que ya han sido clasificados como conjuntos de atributos que generan dependencias funcionales. Para realizar la poda de atributos candidatos es necesario ordenar tanto la lista que contiene las dependencias funcionales como el conjunto de atributos candidatos a encontrar una dependencia funcional. Otro factor importante para la búsqueda de dependecias es la manera en la que se encuentran ordenados los datos de la tabla a analizar. Ahora bien, la ordenación de cualquier conjunto de atributos se realiza mediante el agoritmo *quicksort* debido a que es más rápido que los demás algoritmos de ordenación. Véase las tablas 5.3 y 5.4.

Ahora bien, como puede verse en las tablas 5.3 y 5.4 existen ocho funciones que se utilizan a lo largo del proceso de búsqueda de dependencias funcionales, en esta sección no se describirán tres de ellas las cuales son: *quickSortIterativo, OrdenaVectores* y *estaDentro* debido a que se consideran funciones básicas y las dos últimas mencionadas dependen de la función *quickSortIterativo*.

Cabe mencionar que una de las funciones más elementales es *Esmenorque*, mediante el cual varias funciones dependen a su vez de esta, que como su nombre lo indica, tiene la función de comparar dos valores. Como sabemos el tipo de datos a analizar es de tipo biológico entonces se realizan comparaciones entre cadenas. Éste algoritmo compara dos cadenas lexicográficamente. La comparación está basada en valores *Unicodigo (Unicode)*, para saber mas acerca de esto ver el apéndice A, para cada caracter en la cadena.

El algoritmo para la búsqueda de **dependencias funcionales mediante funciones de similitud** es similar al utilizado para la búsqueda por empatamiento de patrones, la diferencia esta en la línea número 20 en lugar de utilizar la función *VerDFOrd* se utiliza la función *VerDFOrdSimilitudes*, véase el algoritmo en las tablas 5.3

Entrada. A, nombre de la tabla a analizar
Donde A contiene los conjuntos de atributos a analizar
desde $A_1, A_2, ...$ hasta A_n, esto es $A = A_1, A_2, ..., A_n$.
Salida. $B \subset C$
Donde B contiene los conjuntos de atributos que dependen del subconjuto C,
donde $C \subset A$.
Procedimiento. Búsqueda de dependencias funcionales.

```
1.  boolean descendente := true, flag;
2.  boolean primeravez := true;
3.  String tabla[numregistros][atributos elegidos];
4.  Vector T;
5.  Vector ant;
6.  Vector Lista;
7.  (1) frecuencias := cuentafrecuencias(nombretabla);
8.  (2) contenedor:= ordenafrecuencias(atributos, frecuencias);
9.      while(A ! vacio) do
10.         {
11.          antecedentes :=primer elemento de valoresAdecidir ;
12.          eliminar primer elemento en A ;
13.          (3)consecuentes := Consecuentes(prefijo, A);
14.           if(primeravez)
15.              {
16.                  (4)tabla := quickSortInterativo(tabla, longitudTabla,
17.                  atributosElegidos, contenedor, descendente);
18.                  primeravez := false;
19.              }
20.               (5)T := VerDFOrd(tabla, antecedente, consecuentes, atributos);
21.           if(T[0] == 0)
22.              {
23.                ant := copia de antecedentes;
24.                if(tamaño de ant > 1)
25.                (6) ant :=ordenaVectores(ant);
26.               if(Lista es igual a vacio)
27.                  {
28.                    agrega a Lista el último valor de ant;
29.                  }
30.               else
31.                  {
32.                   (7) flag:=estaDentro(ant, Lista) ;
33.                    if(!flag)
34.                    agrega a Lista el último valor de ant;
35.                  }
36.                escribe en archivo el último valor de la Lista;
37.             }
```

Tabla 5.3: Algoritmo para búsqueda de dependencias funcionales por empatamiento de patrones con orden de atributos deacuerdo a sus frecuencias de manera descendente.

Continuación del procedimiento. Dependencias funcionales por empatamiento de patrones con orden de atributos deacuerdo a sus frecuencias de manera descendente.

```
38.          else
39.          {
40.            (8)antAux2 := generaConsecuentesSinLista(Ant, atributos);
41.            for(i=0; i< tamaño de antAux2; i++)
42.             {
43.               union := antAux2(i);
44.               if(tamaño de union > 1)
45.                {
46.                  union = ordenaVectores(union);
47.                  agrega a antAuxOrdenado union;
48.                }
49.               else
50.                 agrega a antAuxOrdenado valor de antAux(i);
51.             }
52.              if(Lista es igual a vacio)
53.               {
54.                 agrega a v2Decidir cada uno de los;
55.                 valores ordenados de antAuxOrdenado;
56.               }
57.               else
58.                {
59.                 for(i=0; i< tamaño antAuxOrdenado; i++);
60.                  {
61.                    aux:= antAuxOrdenado(i);
62.                    esta:= estaDentro(aux, Lista);
63.                    if(!esta)
64.                    agrega aux a v2Decide;
65.                  }
66.                }
67.          }
68.          if(Lista no esta vacía)
69.          v2Decide:= podaValores(v2Decide, Lista);
70.      }.
```

Tabla 5.4: Algoritmo para búsqueda de dependencias funcionales por empatamiento de patrones con orden de atributos deacuerdo a sus frecuencias de manera descendente(continuación).

y 5.4. La función **VerDFOrdSimilitudes** se encarga de obtener la similitud entre dos registros, indendientemente del atributo en el que estén contenidos y verificar si la similitud obtenida se encuentra dentro del rango de similid dado por el usuario. Véase la tabla 5.5.

Cuenta frecuencias es el proceso que se encarga de contar la cantidad de atributos distintos por cada atributo dada una lista de atributos y la tabla de datos. La manera en la procede es la siguiente, accesa a la base de datos para saber que tabla se está analizando, lee cada uno de los atributos recibidos (atributos de interés para el usuario) y busca la columna a analizar dentro de la tabla, para cada columna de la tabla realiza una consulta en SQL para saber el número de datos distintos por atributo elegido, y finalemente detiene la cuenta deacuerdo al número de registros dado por el usuario. Véase tabla 5.6.

Ordena frecuencias este proceso se encarga de ordenar el número de datos distintos por atributo, dada la lista de atributos de interés para el usuario y la cantidad de datos distintos de cada atributo. La manera en la que procede es: realiza n iteraciones desde cero hasta el número de atributos dado, en cada iteración agrega a una lista el valor númerico del atributo realizando el mapeo con respecto al número de columna sobre la tabla a analizar, en seguida agrega a la lista el número de datos distintos para ese atributo, así sucesivamente hasta obtener una lista completa con la forma $atributo_1$,$frecuencia_1$,$atributo_2$,$frecuencia_2$, ..., $atributo_n$,$frecuencia_n$; este paso se realiza para saber que *frecuencia* le corresponde a que *atributo*. Después ordena de manera descendente todas las *frecuencias* y procede a realizar una segunda iteración desde cero hasta el número de frecuencias, dentro de esta iteración vendrá otra iteración que va desde cero hasta el tamaño de la *liga*(lista formada por atributos y frecuencias), a cada valor que tome *j*, donde *j* corresponde al número de iteración de *liga*, se le saca el módulo dos y el resultado es comparado con cero, si es diferente de cero se pregunta si la lista de frecuencias ordenadas en la posición i(número de iteracion de acuerdo al número de frecuencias) es igual a *liga* en la posición *j*, de ser cierto se agrega a *contenedor* el valor de *liga* en la posición *j-1*, remueve en *liga* el valor en la posición j y agrega a *liga* en la posición j el valor de x, termina con todas la iteraciones y devuelve la lista *contenedor* la cual contiene los atributos ordenados deacuerdo a sus frecuencias ordenados de manera descendente. Véase table 5.7.

Consecuentes este proceso tiene como tarea generar conjuntos de atributos que no se cuentren en los antecedentes; es decir generar el complemento de un *antecedente*(conjunto de atributos), los cual es importante para encontrar conjuntos de atributos candidatos para la búsqueda exitosa de dependencias funcionales. Para llevar a cabo este proceso se requiere de una lista de *atributos*; los cuales son de interés para el usuario y que van a ser analizados, con esta lista de atributos se pregunta si el actual conjunto de atributos *Ant* contiene uno de

Entrada. Listas de todos los atributos a analizar *atributos*
Lista de atributos *antecedentes*
Lista de atributos *consecuentes*.
Salida. { *Listadeatributos* si *nohaydependencia*
0 si *haydependencia*
Procedimiento. Verificación de Dependencia funcional
mediante funciones de similitud (VerDFOrdSimilitudes)

```
1.  while(i1 < tamaño de tabla y sigue es verdadero)
2.      {
3.          i2=i1+1;
4.          while(j¡tamaño de antecedentes  flag)
5.          { valatr := elemento j de antecedentes;
6.                  for(n=0; n¡tamaño de atributos; n++)
7.                  {
8.                      valatr2 := elemento n de atributos;
9.                      if(valatr es igual valatr2)
10.                     {
11.                         umbralaux := valor de similitud dado por el usario en el atributo n;
12.                         n := columna que puede leido en tabla;
13.                         valatr = atributo;
14.                         Calcular la similitud entre los registros i1 e i2;
15.                         flag := similitud dentro del umbral;
16.                     }
17.                 }
18.                 j++;
19.             }
20.             if(flag es igual a falso)
21.             {
22.                 sigue := falso;
23.                 val := verdadero;
24.             }
25.             else
26.             Repetir desde 4 hasta el 24 pero con consecuentes con j = 0
27.     if(val es igual a falso)
28.         resultante := lista de consecuentes;
29.     else
30.         Agregar a resultante un cero;
31.     Regresar resultante;
32. Terminar;
```

Tabla 5.5: Algoritmo para búsqueda de dependencias funcionales mediante funciones de similitud.

los elementos de la lista de *atributos*, de no estarlo dicho elemento es agregado a una lista llamada *consecuentes*, esta pregunta se realiza para cada elemento de la lista de *atributo*, finalmente arroja como resultado la lista *consecuentes*. Véase tabla consecuentes.

Genera antecedentes (generaConsecuentesSinLista) este proceso se encarga de generar nuevos *antecedentes* canditados para la búsqueda exitosa de una dependencia funcional, para ello dado un antecedente *Ant* y la lista de *atributos* de interés para el usuario, verifica si de entre los atributos a analizar, algunos se encuentran dentro del *Ant* de no ser así se genera el nuevo antecedente mediante la inserción del atributo que no se encuentra; por elemento que no se encuentra en el *Ant* de la lista de atributos, se genera un nuevo conjunto. Véase tabla 5.11.

Atributos:={12345} Ant:={24} Se generan los siguientes:

$Ant_1 := 241$
$Ant_2 := 243$
$Ant_2 := 245$

Verificación de dependencias funcionales por empatamiento de patrones (VerDFOrd) este proceso es uno de los más importantes dentro de la búsqueda de dependencias funcionales pues tiene la tarea de realizar la búsqueda de dependencias funcionales dados dos conjuntos de atributos, uno de *antecedentes* y otro de*consecuentes*, de donde se encontrará si los *antecedentes* dependen de los *consecuentes*, esto es $antecedentes \longrightarrow consecuentes$, para ello debe recibir la tabla de datos previamente ordenada deacuerdo al orden de prioridad de atributos dada. Encuentra bloques de datos iguales una vez encontrado dicho cantidad de registros iguales sobre el primer atributo de la lista de *antecedentes* busca el siguiente segmento de datos iguales sobre el siguiente atributo de la lista, empezando por el registro inicial y final del segmento de datos encontrados en el bloque anterior. Éste barrido lo realiza hasta terminar con la lista de *antecedentes*, si todos los datos son iguales en el mismo segmento sobre toda la lista de *antecedentes* pueden continuar con la verificación de empatamiento de patrones sobre los datos contenidos en cada uno de los atributos de la lista de *consecuentes*, si resulta que al realizar el barrido sobre la lista de *consecuentes* al menos existe un dato sobre el alguno de los atributo *consecuentes* es diferente, entonces se ha encontrado un conjunto de *antecedentes* que dependen de sus *consecuentes*. Véase tablas 5.9 y 5.10.

Entrada. Nombres de atributos(*nombres*), nombre de tabla(*nombretabla*),
número de registros(*numreg*).
Salida. El número de elementos distintos por atributo .
Procedimiento. Cuenta frecuencias(cuentafrecuencias).

```
{
Accesar a la base de datos;
for(i=0; i< número de nombres atributos; i++)
    {   j:=0;
        nombrecolumna := nombres(i);
        cadena :="SELECT DISTINCT"+nombrecolumna+"FROM"+nombretabla+" " ;
        Ejecuta la consulta cadena;
        while(existan tuplas y j es menor a numreg)
                j++;
        Agregar a frecuencias el valor de j;
    }
Regresar frecuencias;
}
```

Tabla 5.6: Algoritmo para contar el número de valores distintos por atributo en una lista de atributos dados.

Entrada. Lista de atributos (*atributos*),
Lista que contiene la cantidad de datos distintos por atributo(*frecuencias*).
Salida. Lista que contiene las *frecuencias* ordenadas de manera descendente manteniendo el número de atributo al cual corresponde.
Procedimiento. Ordena frecuencias(ordenaFrecuencias).

```
{
    for(i=0; i< número de atributos; i++)
        { Agregar a liga el valor de i;
          Agregar a liga el valor de frecuencias(i);
        }
        ordenados := ordenaVectores(frecuencias);
    for(i=0; i< tamaño de ordenados; i++)
        { for(j=0; j< tamaño de liga; j++)
            {
              valor:=jMod2;
              if(valor es diferente de 0)
              { if(ordenados en la posición i es igual a liga en la posición j)
                    {Agregar a contenedor el valor liga(j-1);
                    Remover de liga el valor en la posición j;
                    Agregar a liga en la posición j el valor x;
                    }
              }
            }
        }
    Regresar contenedor;}
```

Tabla 5.7: Algoritmo que ordena de manera descendete la cantidad de datos distintos por atributo manteniendo el número de atributo al cual corresponde.

```
Entrada. Lista de atributos antecedetes(Ant),
   Lista de atributos a analizar(atributos).
Salida. Vector de vectores que contiene conjuntos
   de complementos del antecedente.
Procedimiento. Consecuentes(Consecuentes).
   {
      Vector consecuentes;
         for(i=0; i< número de atributos; i++)
                {
                      if(Ant no contiene el elemento atributos de la posición i )
                      Agregar a consecuentes el valor de atributos de la posición i;
                }
                Regresar consecuentes;
   }
```

Tabla 5.8: Algoritmo que calcula el complemento(consecuente) de un valor(antecedente).

Entrada. La tabla a analizar (*tabla*)
Lista de atributos antecedentes(*antecedente*),
Lista de atributos consecuentes(*consecuentes*),
Lista de atributos a analizar(*atributos*).
Nota Importante: Se debe dar como parametro de entrada
la tabla ya ordenada respecto al *antecedente*.
Salida. Vector de vectores que contiene conjuntos
de complementos del antecedente.
Procedimiento. Dependencia Funcional Empatamiento(VerDFOrd).

```
1.{ int i1:=0,i2:=0;
2.  boolean val:= verdadero;
3.  While(i1 sea menor a la longitud de la tabla y val sea igual a verdadero)
4.  {   i2:=i1 + 1;
5.      for(n=0; n< tamaño de la lista de antecedentes; n++)
6.      {   valatr:= valor de antecedentes en la posición n
7.          for(i=0; i< tamaño de la lista de atribtuos; i++)
8.          {   if(valatr es igual a atributos en la posición i )
9.              {   valatr:=i;
10.                 i:= tamaño de atributos;
11.             }
12.         }
13.         val1:=el contenido de la tabla en la posición [i1][valatr];
14.         x:=x+val1;
15.         val2:=el contenido de la tabla en la posición [i2][valatr];
16.         y:=y+val2;
17.     }
18.     if(i2 es menor a longitud tabla y x es igual a y)
19.         i2++;
20.     if(i2 es diferente a (i1 + 1))
21.     {
22.         for(i3=i1+1; i3<=i2−1; i3++)
23.         {   for(n=0; n< consecuentes;n++)
24.             {   valatr3:=consecuentes en la posición n;
```

Tabla 5.9: Algoritmo que encuentra las dependencias funcionales por empatamiento de valores en bloques mediante la lista de atributos dada.

Procedimiento. Dependencia Funcional Empatamiento(VerDFOrd)CONTINUACION.

```
25.              for(i=0; i tamaño de atributos;i++)
26.              {
27.                if(valatr3 es igual a atributos en la posición i)
28.                {
29.                    valatr3:=i;
30.                    i:=tamaño de atributos;
31.                }
32.              }
33.               val3:=tabla en la posición [i1][valatr3];
34.               x2 := x2+val3;
35.               val4:=tabla en la posición [i3][valatr3];
36.               y2 := y2+val4;
37.            }
38.             if(x2 no es igual a y2)
39.            {
40.               i3 := i2 − 1;
41.               val:=falso;
42.            }
43.             x2:="";
44.             y2:="";
45.           }
46.        }
47.         i1:= i2;
48.  }
49.  if(val es igual a falso)
50.    resultante:= toma valor de consecuente;
51.  else
52.      Agregar a resultante el valor 0;
53.  regresar resultante;
54. }
```

Tabla 5.10: Algoritmo que encuentra las dependencias funcionales por empatamiento de valores en bloques mediante la lista de atributos dada(continuación).

Entrada. Lista de atributos antecedentes (*Ant*),
Lista de atributos a analizar (*atributos*),
Salida. Lista que contiene los nuevos antecedentes para búsqueda de probables dependencias.
Procedimiento. Genera antecedentes sin una lista de dependencias encontradas(generaConsecuentesSinLista).

```
{
  for(i := 0;i < tamaño de atributos; i + +)
  {
     if(Ant no contiene el elemento i de atributos)
    {
       Agregar a extensión el elemento atributos en la posición i;
     }
   }
}.
```

Tabla 5.11: Algoritmo que se encarga de generar nuevos antecedentes para busqueda de probables dependencias validad,para ello dado un antecedente "Ant" verifica si de entre los atributos a analizar, algunos se encuentran dentro de "Ant" de no ser así se genera el nuevo antecedente insertandole el atributo que no esta; elemento por elemento.

5.1.2. Funciones de similitud

Vamos a definir a una *función de similitud* como aquella que sirve para buscar la semejanza o parecido entre dos o más datos, donde el grado de semejanza está dado por un porcentaje de cero a ciento por ciento, en el cual cero por cientro quiere decir que ambos datos no tienen parecido alguno, y el ciento por ciento se refiere a que ambos datos son idénticos. En este proyecto se crearon funciones de similitud para encontrar el parecido sobre distintos tipos de datos, dependiendo del tipo de dato se aplicó cierta función de similitud. Véase la tabla 5.12.

Selección de función de similitud este proceso se encarga de ejecutar la función de similitud deacuerdo al atributo dado, devolviendo como salida un "1" si los datos a comparar se encuentran dentro del rango de similitud, de lo contrario devuelve un "0". Ahora bien, la manera en la que se ejecutan las funciones de similitud de acuerdo al número de atributo se muestra en la tabla 5.12.

A continuación se realiza una descripción de cada una de las funciones implementadas:

Función simiSymbol se encarga de encontrar el valor de similitud entre dos valores sobre los campos enumerados en la tabla 5.12 para el cual no existe un patrón definido para los datos sobre dichos campos, es por ello que se les aplica una función de similitud basada en la máxima cadena común. La lógica que se siguió para la creación de esta función de similitud (f_1) dadas las cadenas σ y τ es la siguiente:

$D_1 = Symbol|GBaccession|FISHBAND|$

$F_1 = A^* \times A^* \longrightarrow [0, 1]$

Dado $\sigma, \tau \in A^*$
Donde σ, τ son cadenas de caracteres.

$\mu := \sigma \cap \tau$ (máxima cadena común)

$z := 1 - \frac{long(\sigma-\mu)+long(\tau-\mu)}{long(\sigma)+long(\tau)}$

Para $f_1(\sigma, \tau) := z$

Algunos ejemplos sobre el tipo de dato que se maneja para aplicar esta función de similitud se muetra en la tabla 5.13, la cual muestra los datos contenidos en los campos *Symbol, GBaccession* y *FISHBAND*.

La manera en la que se obuvo la máxima subcadena común entre registros se puede ver en las tablas 5.29 y 5.30 obtenidos de los apuntes de "Análisis y Diseño de algoritmos por Dr. Guillermo Morales Luna". Véase la referencia [30].

número de atributo	**nombre de atributo**	**función de similitud**
1	ID	no tiene
2	Symbol	simiSymbol
3	GeneName	simiDiccionario
4	oligoid	oligoID
5	humanortholog	simiDiccionario
6	mouseorthologs	simiDiccionario
7	GBaccession	simiSymbol
8	DescriptionRGD	simiDiccionario
9	Function1k	simiDiccionario
10	GObiologicalprocess	simiGO
11	GOmolecularfunction	simiGO
12	GOcelllocation	simiGO
13	CHROMOSOME	simiChromosome
14	FISHBAND	simiSymbol
15	ENT1NLratio1k	
16	ENT5NLratio1k	
17	HCCNLratio1k	
18	pvalueENT1NL1k	
19	pvalueENT5NL1k	simiValores
20	pvalueHCCNL1k	
21	FDR1k	
22	Venn1k	
23	ENT1NLratio28k	
24	HCCNLratio28k	
25	HCCNTratio28k	
26	pvalueENT1NL28k	
27	pvalueHCCNL28k	
28	pvalueHCCNT28k	
29	FDR28k	
30	Venn28k	

Tabla 5.12: Tabla de asignación de funciones de similitud de acuerdo al número de atributo.

Symbol	**GB-accession**	**FISHBAND**
Kif22_predicted	BC088421	q36
Kif2b	BC083841	10q26
Kif5b		q12.1

Tabla 5.13: Muestra de algunos datos sobre los campos *Symbol, GBaccession* y *FISBAND*.

Función simiDiccionario este proceso calcula la similitud entre dos enunciados dados sobre los campos mencionados en la tabla 5.12. Estos atributos contienen descripciones de información de genes por lo que el cálculo de la similitud esta basado en una especie de diccionario por enunciado, así como en el número de palabras iguales ambos registros dados sobre el campo a analizar, véase la tabla 5.16. La lógica para esta función es la siguiente:

$D_2 := Gene{-}name|HumanOrtholog|MouseOrthologs|DescriptionRGD|Function1k$
$domD_2 := (palabra)^*$
$f_2 := (palabra)^* \times (palabra)^* \longrightarrow [0,1]$

Función de similitud igual a la f_1 solo cambiando el tipo de entrada, en lugar de cadenas de caracteres acepta enunciados.

$$z := 1 - \frac{long(\sigma-\mu)+long(\tau-\mu)}{long(\sigma)+long(\tau)}$$

donde:
μ := número de palabras comunes contenidas en ambos diccionarios. σ := primer enunciado. τ := segundo enunciado.

Nota Importante: antes de realizar todo el proceso es necesario analizar ambas cadenas(enunciados) con la finalidad de eliminar signos raros para considerar únicamente palabras. Véase la tablas 5.17 y 5.18.

Función oligoId encuentra la similitud entre dos registros del campo *oligoId* apartir de cuatro funciones de similitud: 1)funUnoOligo, 2)funDosOligo, 3)funTresOligo y 4)funCuatroOligo. La similitud resultante surge de la siguiente ecuación:

$$\text{similitud:}= \frac{sim1}{9} + \frac{5*sim2}{9} + \frac{sim3}{9} + \frac{2*sim4}{9}$$

Ahora bien, $sim1, sim2, sim3$ y $sim4$ son el resultado de las funciones de similitud mencionadas arriba, la forma en la que cada resultado es dividido entre nueve es debido a que el patrón que sigue este campo es de nueve caracteres los que conforman un dato, y cada uno se multiplica por el número de caracteres que conforman cada segmento de dato; esto es, $sim1$ está compuesto por un solo caracter, $sim2$ está compuesto por cinco caracteres, $sim3$ por un caracter y finalemente $sim4$ por dos caracteres. La forma en la que se presentan los datos es la siguiente:

```
Rn30017743
Rn30017704
Rn30000026
R001262_01
R005315_01
```

Rn30000011
Rn30005034
Rn30005035
Rn30008908
Rn30000371
Rn30012360
Rn30000820
Rn30012826
R002160_01
Rn30013228
R001477_01
R004554_01

De donde el comportamiento del campo *oligo_id* se describe abajo. $D_3 := Oligo\_id$
$domD\_3 := A_3$

$A\_3 ::= vacia|R\langle a_2\rangle\langle c_5\rangle\langle s_1\rangle\langle p\rangle$

$\langle a_2\rangle :: n|0$
$\langle c_5\rangle :: [0-9]^5$
$\langle s_1\rangle :: \_|[0-9]$
$\langle p\rangle :: [0-9]^2$

$f_{31} : A_3 \times A_3 \longrightarrow [0,1]$

La manera en la que se realizó la división de caracteres y la suma de los resultados de las funciones de similitud multiplicados por un factor.

$$f_3 2(\sigma,\tau) := a_1 f_{321}(\sigma_1,\tau_1) + a_2 f_{322}(\sigma_2,\tau_2) + a_3 f_{323}(\sigma_3,\tau_3) + a_4 f_{324}(\sigma_4,\tau_4)$$

donde:
$a_1 = \frac{1}{9}$
$a_2 = \frac{5}{9}$
$a_3 = \frac{1}{9}$
$a_4 = \frac{2}{9}$
Para ver más detalles sobre esta función véase la tabla 5.19.

Función OligoId (funUnoOligo,f_{321}) esta función de similitud analiza el segundo caracter de cada cadena de entrada, mediante los cuales surgen cuatro combinaciones posibles de aparación en los datos, si ambos caracteres son distintos, esto es el primer caracter de la cadena σ es igual al primer caracter de la cadena

τ la función de similitud f_{321} arrojará como resultado un *uno*, en caso contrario arrojará un *cero*.

primer caracter:=σ
segundo caracter:=τ
`n n` $\longrightarrow 1$
`n 0` $\longrightarrow 0$
`0 n` $\longrightarrow 0$
`0 0` $\longrightarrow 1$

Para ver la forma en la que está funcionando este proceso véase la tabla5.20.

Función OligoId (funDosOligo,f_{322}) esta función de similitud analiza cinco caracteres de dos cadenas σ y τ a partir de la tercera posición de cada cadena, como los cinco caracteres siempre son numéricos se realiza la conversión de cada caracter al tipo de dato entero formando asi un número de cinco dígitos por cadena leida. Una vez realizado esto, se obtiene la diferencia entre ambos valores, se divide entre el valor máximo que puede obtener el dígido formado por una cadena, y el resultado es multiplicado por menos uno y sumado a uno, con estas operaciones se obtiene el valor de similitud entre la dos cadenas σ y τ. Véase tabla 5.21.

$$f_{322} : (\sigma_2, \tau_2) \longrightarrow |\sigma_2 - \tau_2| := 1 - \frac{|\sigma_2 - \tau_2|}{99999}$$

Función OligoId (funTresOligo,f_{323}) esta función de similitud analiza el caracter de la posición ocho de las cadenas σ y τ, de lo cual la similitud tomará los valores descritos abajo, dependiendo del caracter de entrada.Véase tabla 5.22.

$$f_{323} :\longrightarrow \begin{cases} 1 \text{ si } x = y \\ 1 \textit{ si } \text{x=\_ \&\& } y \in [0-9] \\ 1 - \frac{x-y}{9} \text{ si } x, y \in [0-9] \end{cases}$$

Función OligoId (funCuatroOligo,f_{324}) esta función de similitud obtiene la similitud analizando dos caracteres de dos cadenas x y y a partir de la novena posición de cada cadena, ambos caracteres siempre son numéricos en ambas cadenas, por lo que se realiza la conversion de cada caracter al tipo de dato entero formando asi un número de dos dígitos por cadena leida. Una vez realizado esto, se obtiene la diferencia entre ambos valores, se divide entre el valor máximo que puede obtener el dígido formado por una cadena, y el resultado es multiplicado por menos uno y sumado a uno, con estas operaciones se obtiene el valor de similitud entre la dos cadenas x y y. Véase tabla 5.23.

$$f_{324} : (x, y) \longrightarrow 1 - \frac{|x-y|}{99}$$

GO-biological-process	**GO-molecular-funciton**	**GO-cell-location**
0006413:translational initiation	0045182:translation regulator activity	
0007154:cell communication	0016503:pheromone receptor activity	0016021:integral to membrane
0008151:cell growth and/or maintenance	0017076:purine nucleotide binding	0016021:integral to membrane
0050794:regulation of cellular process	0000166:nucleotide binding	0005856:cytoskeleton
0008151:cell growth and/or maintenance	0005488:binding	0016020:membrane
0016032:viral life cycle	0005488:binding	0019028:viral capsid
0050874:organismal physiological process	0030234:enzyme regulator activity	0005576:extracellular

Tabla 5.14: Muestra de algunos datos sobre los campos *GObiologicalprocess, GOmolecular-functon* y *GOcelllocation.*

Función simiGO esta función de similitud es aplicada a los atributos `GO-biological-process`, `GO-molecular-function` y `GO-cell-location`. En la tabla 5.14 es mostrado el tipo de contenido en estos campos.

El patron que siguen los datos para estos campos es el siguiente:

$D_{go} := GO\_biological == GO\_molecular == GO\_cell$
$domD\_go := A_{go}$

$A\_go ::= vacia|\langle a_7\rangle\langle c_1\rangle\langle texto\rangle$

$\langle a_7\rangle :: [0-9]^7$
$\langle c_1\rangle :: :$
$\langle texto\rangle :: ([a-z]|[A-Z])^*$

$f_{simigo} : A_{go} times A_{go} \longrightarrow [0,1]$

Ahora bien, el análisis de un dato para obtener su valor de similitud en el dominio de los tres atributos anteriormente mencionados, se puede determinar particionando el dato en dos partes, una parte será antes de los ":"; denomidada *cadena1* y la segunda después de los ":" que se mencionará como *cadena2*. La similitud entre registros será la suma de las dos similitudes resultantes de la *cadena1* y de *cadena2* dividida entre dos. Como se puede ver *cadena1* es un

número compuesto por siete dígitos, por lo que para calcular la similitud de dos registros *r1* y *r2*, se obtiene entonces la *cadena1 de r1* y *cadena1 de r2*, se obtiene la diferencia entre ambos números y se divide entre el valor máximo que puede tomar un número con siete dígitos, con esto ya tenemos la *primer similitud*, véase la tabla 5.25. Ahora, para obtener la *segunda similitud* se utiliza la función *simiDiccionario*, véase las tablas 5.17 y 5.18, la cual recibirá como parámetros de entrada *cadena2 de r1* y *cadena2 de r2*. Véase la tabla 5.24.

Función simiChromosome esta función calcula la función de similitud entre dos registros *r1* y *r2* en el campo `CHROMOSOME` la manera en la que lo realiza es preguntando si *r1* o *r2* son iguales a "Null", en caso de que así sea, el valor de la similitud es cero, en caso contrario realiza un "cast" de *r1* y *r2* a valores enteros, obtiene la diferencia entre ellos y el resultado es divido entre noventa y nueve, esto debido a que los datos siempre son como máximo de dos dígitos, luego el resultado es multiplicado por menos uno y se le suma el valor de uno. Véase tabla 5.26.

$$similitud := 1 - \frac{|r1-r2|}{99}$$

Función simiValores este proceso calcula la similitud entre dos registros sobre los campos:

15. `ENT1NLratio1k`
16. `ENT5NLratio1k`
17. `HCCNLratio1k`
18. `pvalueENT1NL1k`
19. `pvalueENT5NL1k`
20. `pvalueHCCNL1k`
21. `FDR1k`
22. `Venn1k`
23. `ENT1NLratio28k`
24. `HCCNLratio28k`
25. `HCCNTratio28k`
26. `pvalueENT1NL28k`
27. `pvalueHCCNL28k`
28. `pvalueHCCNT28k`
29. `FDR28k`
30. `Venn28k`

para ello convierte las cadenas de estos campos a valores númericos y utiliza la función de similitud $similitud := 1-(resta/nueve)$ donde *resta* es la magnitud de la diferencia entre dos registros elegidos sobre el mismo atributo, y *nueve* es el resultado de la función *obtennueves*, véase tabla 5.28, el cual concatena tantos nueves como digitos existan convirtiéndolo a un valor numérico listo para se aplicado en la ecuación de similitud. Véase tabla 5.27.

Entrada. Tabla a analizar $tablaOr[][]$,
dos indices de registros $reg1, reg2$,
número de atributo *atributo*.
Salida. Valor de similitud entre dos cadenas
Procedimiento. Función simiSymbol.

```
double simulitud;
int sigma2;
String sigma := tablaOr[registro1][atributo];
String tao := tablaOr[registro2][atributo];
 if(sigma es true)
  {
   similitud :=0.0;
  }
 else
  {
   int longsigma:= longitud de sigma;
   int tao:= longitud de tao;
   String B[][] := maximaSubComun(sigma, tao);
   int i := longitud de sigma;
   int j := longitud de tao;
   String v[][] := generaMSC(B, i, j, sigma);
   if(v es Nulo)
    {
     sigma2:=0;
      else
       {
        v:= subcadena de v iniciando en la posición 5
        hasta la longitud de v;
        sigma2:= longitud de v;
       }
       double sigma0prima := longsigma - sigma2;
       double sigma1prima := longtao - sigma2;
       similitud := 1 - ((sigma0prima + sigma1prima)/( longsigma + longtao));
    }
  }
   regresa similitud;
```

Tabla 5.15: Algoritmo función de similitud *simiSymbol*que calcula la similitud entre dos cadenas.

Entrada. Tabla a analizar $tablaOr[][]$,
dos indices de registros $reg1, reg2$,
número de atributo *atributo*.
Salida. Valor de similitud entre dos cadenas.
Procedimiento. Función geneName.

```
String cadena1;
String cadena2;
double similitud := 0.0;

  cadena1 := tablaOr[registro1][atributo];
  cadena2 := tablaOr[registro2][atributo];
  if(cadena1 es Nula)
    {
      similitud :=0.0;
    }
  else
    {
      similitud := simiDiccionario(cadena1, cadena2);
    }
     regresar similitud;
```

Tabla 5.16: Algoritmo de función de similitud *geneName*que calcula la similitud entre dos cadenas.

Entrada. Dos cadenas *cadena*1 y *cadena*2
Salida. Valor de similitud entre dos cadenas
Procedimiento. Función simiDiccionario.

```
{
 Vector vector1[];
 Vector vector2[];
 double sigma0prima;
 double sigma1prima;
  While(cadena1 tenga elementos) do
   {
    cadena1:= el valor del siguiente elemento;
    if (cadena1 contiene comillas)
     {
      if (la posición 0 de la cadena1 contiene comillas)
       {
        Agregar a vector1 el substring de cadena1 empezando en la posición 2 hasta
        la longitud de cadena1;
       }
      else
       {
        Agregar a vector1 el substring de cadena1 empezando en la posición 0 hasta
        la longitud de cadena1 menos 1;
       }
     }
    else
    {
       if (cadena1 contiene coma)
       {
          Agregar a vector1 el substring de cadena1 empezando en la posición 0 hasta
          la longitud de cadena1 menos 1;
       }
       else
       {
          Agregar a vector1 la cadena1;
       }
    }
   }
```

Tabla 5.17: Algoritmo que calcula la similitud entre dos enunciados completos.

Procedimiento. Función simiDiccionario CONTINUACION.

```
While(cadena2 tenga elementos) do
 {
  cadena2 := el valor del siguiente elemento;
  if (cadena2 contiene comillas)
   {
    if (la posición 0 de la cadena2 contiene comillas)
     {
      Agregar a vector2 el substring de cadena2 empezando
      en la posición 2 hasta
      la longitud de cadena2;
     }
    else
     {
      Agregar a vector2 el substring de cadena2 empezando
      en la posición 0 hasta
      la longitud de cadena2 menos 1;
     }
   }
  else
  {
     if (cadena2 contiene coma)
     {
        Agregar a vector2 el substring de cadena2 empezando
        en la posición 0 hasta
        la longitud de cadena2 menos 1;
     }
     else
     {
        Agregar a vector2 la cadena2;
     }
  }
 }
  contador:= Número de palabras iguales en ambos vectores;
  sigma0prima := longitud de cadena1 - contador;
  sigma1prima := longitud de cadena2 - contador;
  regresar similitud;
}
```

Tabla 5.18: Algoritmo que calcula la similitud entre dos enunciados completos(Continuación).

Entrada. Tabla a analizar $tablaOr[][]$,
dos indices de registros $reg1, reg2$, número de atributo *atributo*
Salida. Valor de similitud entre dos cadenas
Procedimiento. Función oligoId.

String $cadena1$:= $tablaOr[registro1][atributo]$;
String $cadena2$:= $tablaOr[registro2][atributo]$;
double $simiTotal$;
if($cadena1$ o $cadena2$ son nulas)
{
$simiTotal$:=0.0;
}
else
{
double $sim1$:= $funUnoOligo(cadena1, cadena2)$;
double $sim2$:= $funDosOligo(cadena1, cadena2)$;
double $sim3$:= $funTresOligo(cadena1, cadena2)$;
double $sim4$:= $funCuatroOligo(cadena1, cadena2)$;

$simiTotal$:= ($sim1$/9) + ((5*$sim2$)/9) + ($sim3$/9) + ((2*$sim4$)/9);
}
regresar $simiTotal$;

Tabla 5.19: Algoritmo de función de similitud *oligoId*, compuesta por cuatro funciones de similitud para el atributo que lleva el nombre de la función.

Entrada. Dos cadenas a analizar $cadena1$ y $cadena2$
Salida. Valor de similitud entre dos cadenas.
Procedimiento. Función funUnoOligo.

double val:=0.0;
if(El elemento 2 de $cadena1$ es igual al elemento 2 de $cadena2$)
{
val:=1.0;
}
regresar val;

Tabla 5.20: Algoritmo de la primera función de similitud para *oligoId* (f_{321}).

Entrada. Dos cadenas a analizar *cadena*1 y *cadena*2
Salida. Valor de similitud entre dos cadenas.
Procedimiento. Función funDosOligo.

double *similitud*:=0.0;
String *sigma* := Substring de *cadena*1 empezando en el elemento 3 hasta el 8;
String *tao* := Substring de *cadena*2 empezando en el elemento 3 hasta el 8;
double *sigmaval* := *sigma*;
double *taoval* := *tao*;
double *resta* := *sigmaval*- *taoval*;
if(*resta* es menor a cero)
{
resta := *resta**(-1);
}
similitud = 1 - (*resta*/99999);
regresar *similitud*;

Tabla 5.21: Algoritmo de la segunda función de similitud para *oligoId* (f_{322}).

Entrada. Dos cadenas a analizar *cadena*1 y *cadena*2
Salida. Valor de similitud entre dos cadenas.
Procedimiento. Función funTresOligo.

```
double val:=0.0;
if(El elemento 8 de cadena1 y cadena2 son iguales)
 {
   val:=0.0;
 }
else
 {
   if (El elemento 8 de cadena1 ó el de cadena2 son iguales a guión bajo)
    {
     if(El elemento 8 de cadena1 ó el de cadena2 no son iguales)
       {
        val:=1.0;
       }
    }
   else
    {
      String num1 := substring de cadena1 iniciando en posición 8 y
      finalizando en posición 9;
      String num2 := substring de cadena2 iniciando en posición 8 y
      finalizando en posición 9;
      double num1_ := num1;
      double num2_ := num2;
      double resta := num1_ - num2_;
      if (resta es menor a cero)

      {
       resta = resta*(-1);
      }
       val= 1 -(resta/9);
    }
 }
  regresar val;
```

Tabla 5.22: Algoritmo de la tercera función de similitud para *oligoId* (f_{323}).

Entrada. Dos cadenas a analizar *cadena*1 y *cadena*2
Salida. Valor de similitud entre dos cadenas.
Procedimiento. Función funCuatroOligo.

double *similitud*:=0.0;
String *sigma* := substring de *cadena*1 desde la posición 9 hasta la posición 11);
String *tao* := substring de *cadena*2 desde la posición 9 hasta la posición 11);
double *sigmaval* := *sigma*;
double *taoval* := *tao*;
double *resta*:= *sigmaval* - *taoval*;
if(*resta* es menor a cero)
{
resta := *resta**(-1);
}
similitud = 1 - (*resta*/99);
regresar *similitud*;

Tabla 5.23: Algoritmo de la cuarta función de similitud para *oligoId* (f_{324}).

Entrada. $tablaOr, registro1, registro2$ y $atributo$
Salida.
Procedimiento. Función simiGO.

String $cadena1$;
String $cadena2$;
double $similitud$:=0.0;
$cadena1 := tablaOr[registro1][atributo]$;
$cadena2 := tablaOr[registro2][atributo]$;
if($cadena1$ ó $cadena2$ son $Nulos$)
{
double $simiuno := simiGOuno(cadena1, cadena2)$;
String $cadena3$:= substring de $cadena1$ desde posición 9 hasta la longitud de $cadena1$ menos 1;
String $cadena4$:= substring de $cadena2$ desde posición 9 hasta la longitud de $cadena2$ menos 1;
double $simidos := simiDiccionario(cadena3, cadena4)$;
$similitud := (simiuno + simidos)/2$;
}
regresar $similitud$;

Tabla 5.24: Algoritmo para obtener la similitud entre dos registros de los campos *GObiologicalprocess, GOmolecularfuncton* y *GOcelllocation.*

Entrada. Dos cadenas *cadena1* y *cadena2*.
Salida. Similud entre las cadenas dadas
Procedimiento. Función simiGOuno.

```
String cadena1;
String cadena2;
double similitud := 0.0;
cadena1 := tablaOr[reg1][atributo];
cadena2 := tablaOr[reg2][atributo];
if(cadena1 ó cadena2 son Nulos)
{
   similitud = 0.0;
}
else
{
   double sigmaval := cadena1;
   double taoval := cadena2;
   double resta := sigmaval - taoval;
  if(resta es menor a 0)
   {
     resta := resta*(-1);
   }
 similitud := 1 - (resta/9999999);
}
regresar similitud;
```

Tabla 5.25: Algoritmo para obtener la similitud entre dos registros, considerando los primeros siete dígitos de los campos *GObiologicalprocess, GOmolecularfuncton* y *GOcell-location*.

Entrada. *cadena*1 y *cadena*2
Salida. Similitud entre dos cadenas
Procedimiento. Función simiChromosome.

double *similitud*;
String *sigma* := substring de *cadena*1 iniciando en posición 0 hasta la 7;
String *tao* := substring de *cadena*2 iniciando en posición 0 hasta la 7;
double *sigmaval* := *sigma*;
double *taoval* := *tao*;
double *resta* := *sigmaval* - *taoval* ;
if(*resta* es menor a 0)
{
resta = *resta**(-1);
}
similitud = 1 - (*resta*/9999999);
regresar *similitud*;

Tabla 5.26: Algoritmo para obtener la similitud entre dos registros sobre el campo *CHROMOSOME*.

Entrada. Dos cadenas *cadena*1 y *cadena*2 a analizar
Salida. Similitud entre dos cadenas
Procedimiento. Función simiValores.

```
double similitud;
String cadena1;
String cadena2;
cadena1 := tablaOr[registro1][atributo];
cadena2 := tablaOr[registro2][atributo];
if(cadena1 ó cadena2 es Nula)
{
 similitud :=0.0;
}
else
{
   sigmaval := cadena1;
   taoval := cadena2;
   resta := sigmaval - taoval;
   if (resta es menor a cero)
   {
    resta = resta*(-1);
   }
   nueve := obtennueves(resta);
   similitud := 1 - (resta/ nueve);
regresar similitud;
```

Tabla 5.27: Algoritmo para obtener la similitud entre dos registros sobre los campos *ENT1NLratio1k, ENT5NLratio1k, HCCNLratio1k, pvalueENT1NL1k, pvalueENT5NL1k, pvalueHCCNL1k, FDR1k,Venn1k, ENT1NLratio28k, HCCNLratio28k, HCCNTratio28k, pvalueENT1NL28k, pvalueHCCNL28k, pvalueHCCNT28k, FDR28k* y *Venn28k.*

Entrada. Un número *resta* del tipo *double*.
Salida. El valor máximo que puede tomar el valor de
entrada antes de incrementar un dígito, conservando la misma
cantidad de dígitos.
Procedimiento. Función obtennueves.

```
Hacer cadena := Conversión de resta a String
  while(cadena contenga el caracter del 1 al 9 en su posición i) do
  {if (La cadena en su elemento i es diferente a .)
    {
     Aumentar cuenta
    }
   Aumentar i;
  }      String aux :=" ";
   String val9 :="9";
  for (i:=0;i¡cuenta;i++)
  { aux:=val9+aux;
  }

   nueve := Convertir aux a double;
   totalpal := Longitud de cadena;
   cuenta2 := totalpal - cuenta -1;
  if (cadena en al posición (cuenta + 1) es igual con 0 y cuenta2 es igual a1)
  { nueve2:=0.0;
  }
  else
  {aux :=" ";
     for (i:=0;i¡cuenta2;i++)
         {
           aux:= val9+aux;
         }
          nueve2 := Convertir aux a double;
          nueve2 := nueve2/(nueve2 + 1);
  }
 double divisor := nueve + nueve2;
 regresar divisor;
```

Tabla 5.28: Algoritmo para calcular el valor máximo con respecto al valor de entrada antes de incrementar en un dígito, conservando la misma cantidad de dígitos entre la entrada y la salida.

Entrada. Dos cadenas σ y τ.
Salida. Matriz B con la orientación entre dos palabras
para encontrar la máxima subcadena común.
Procedimiento. Máxima subcadena común(maximaSubComun).

```
{
  int i,j,m,n;
  m := longitud de σ;
  n := longitud de τ;
  int C[m + 1][n + 1];
  String B[m + 1][n + 1];
  //inicializando matriz C
  for(i = 0;i < m;i + +) {  for(j = 0;j < n;j + +)
       C[i][j]=0;
   }
  for(i = 1; i <= m;i + +)
      C[i][0]= 0 ; //anúlese la primera columna
  for(j = 1; j <= n;j + +)
      C[0][j]= 0 ; // anúlese el primer renglón
  for(i = 1; i < m + 1; i + +){
  for(j = 1;j < n + 1;j + +){
     if(caracter de σ en la posición i−1
       es igual al caracter de τ en la misma posición)
      {  C[i][j]:=C[i − 1][j − 1]+1;
         B[i][j]:= "NO";
      }
      else{
        if( La matriz C en la posición [i-1][j]
          es mayor o igual en su posición [i][j-1])
        {
            C[i][j] := C[i − 1][j];
            B[i][j] :="N";
        }
        else
        { C[i][j] := C[i][j − 1];
          B[i][j] :="O";
        }
      }
    }
  }
  //devuelve C & B
  Regresar B;
}
```

Tabla 5.29: Algoritmo que calcula la matriz de orientación hacia la máxima subcadena común entre dos cadenas.

Entrada. Matriz bidimensional B, int i, int j, caden σ
Salida. La máxima subcadena común.
Procedimiento. Genera máxima subcadena común(generamaximaSubComun).

```
{
  String v;
 if(i es igual a 0 o j es igual a 0)
 {
    v := null;
 }
 else
 {
   if( la matriz B en la posición [i][j] es igual a "NO")
   {
    v := generaMSC(B, i − 1, j − 1, sigma);
    v := v+ el caracter de σ en la posición(i − 1);
   }
   else
   {
     if( la matriz B en la posición [i][j] es igual a "N")
        v := generaMSC(B, i − 1, j, sigma);
     else
         v := generaMSC(B, i, j − 1, sigma);
   }
 }
 Regresar v;
}
```

Tabla 5.30: Algoritmo que genera la máxima subcaden común dada la matriz B de orientación.

5.1.3. Algoritmos para migración de información

Estos algoritmos sirven para traspasar información entre bases de datos. Se aplican cuando las bases de datos son muy grandes y el manejador que se está utilizando ya no es suficiente para manipular toda la información, por lo que se requiere de un sistema gestor de base de datos (SGBD) mas potente. La *migración de los datos* consiste en convertir los datos desde un sistema de base de datos a otro mediante la creación de tablas o modificaciones de las ya existentes, cambios en tipos de datos que existen en una base de datos pero no en otras, etc.

Existen algunos tipos de datos que deben ser tratados con mas cuidado como los campos de fecha, numéricos(enteros, reales, etc.), campos para imágenes, etc., debido a que cada SGBD espera los datos de manera diferente.

Para este proyecto se creó un "script" para la migración de la información, en el cual se ha utilizado de manera conveniente el manejo de SQL estándar con la finalidad de evitar incompatibilidades en cuanto al manejo de tipos de dato. Ahora bien, el proceso que se llevó para el manejo de la información se describe a continuación.

Primero se recibe la ruta del archivo en *formato XLS* donde se cuentra la tabla de datos a migrar, la ruta donde se desea crear el archivo "Script", y el nombre de la nueva tabla a crear.

Segundo, se leyeron los datos desde Microsoft Excel mediante un API de Java para realizar la interacción entre ambas herramientas, donde se considera el primer renglón de la hoja de cáculo de Excel para obtener los nombres de los atributos para crear la nueva tabla en Posgres, eliminando de cada nombre de columna los símbolos de guión bajo y guión alto para evitar conflictos de sintaxis con el lenguaje de SQL.

Tercero, se realizó la depuración de todos los datos sobre los diferentes campos elimando comillas dobles, simples y apóstrofes únicamente en la parte inicial y final de las cadenas, esto con la finalidad de cumplir con la sintáxis que maneja SQL para la inserción de datos en una base de datos.

Cuarto, se creó una llave única por registro mediante la concatenación de datos evitando entonces la migración de información redundante en la nueva tabla a generar.

Hasta aquí, todo se fue escribiendo en un archivo de texto con el nombre "Script-SQL" que automáticamente se guarda en la unidad C. Véase tablas 5.31 y 5.32.

Finalmente, se ejecuta de manera transparente el "Script" para realizar la migración de datos desde Excel al manejador de base de datos Posgres y así ser manipulada por todo usuario que ingrese al sistema. Véase la tabla 5.33.

Entrada. Ruta de la hoja de cálculo en Excel (*rutatabla*),
Nombre de la nueva tabla a crear (*nombretabla*) y
Ruta donde se desea crear el archivo de texto para migrar los datos (*rutarecipiente*).
Salida. Una nueva tabla lista para ser manipulada.
Procedimiento. Función creaScript.

int *i*,*j*,*n*;
rutarecipiente := Ruta y Nombre del archivo que contendrá el Script de carga en BD.;
String *llave_pk*,*aux*,*car*;
Crear archivo *rutarecipiente*, en el que se escribirá el Script de carga en BD.;
Crear instancia que comunique Java con Excel;
Extraer primera hoja del archivo Excel;
int *numreg*:= Obtener el número de regisros de la hoja de Excel;
int *numcol* := Obtener el número de columnas de la hoja de Excel;
String *registro*,*cadena*,*cadena_aux*;
Escribir en archivo *rutarecipiente* "CREATE TABLE + *nombretabla* (";

for (*i*:=0;*i* < *numcol* ;*i*++)
{ *registro* := Obtener el contenido de la celda en la posición *i* del primer
registro de la hoja de Excel;
registro := Reemplazar por espacios en blanco de *registro* caractéres como -, _, (y) ;
st := *registro*;

while (*st* tenga elementos)
{ *cadena_aux* := Siguiente elemento de *st*;
if(*cadena_aux* no es espacio en blanco)
{ *cadena* := *cadena*+*cadena_aux*;
Escribir *cadena* a archivo *rutarecipiente*;
cadena_aux := Limpiar *cadena_aux*;
cadena := Limpiar *cadena*;
}

Escribir al final del archivo *rutarecipiente* "llave TEXT PRIMARY KEY"
Hacer salto de línea en archivo *rutarecipiente*;
char *array*[];
String *val*;

Tabla 5.31: Algoritmo que crea un "Script" para realizar la migración de datos.

Entrada. *rutatabla*, *nombretabla* y *rutarecipiente*
Salida.
Procedimiento. Función creaScript CONTINUACION.

```
        for (i=1; i < numreg ; i++)
        { Escribir en la segunda línea de archivo rutarecipiente "INSERT INTO
          + nombretabla + Values (";
          for (j=1; j < numcol ; j++)
          {registro := Obtener contenido de la celda j en del registro i;
            registro := Eliminar especios en blanco de registro
            if (registro contiene ')
            {array := Convertir registro en arreglo de caractéres;
              for (n=0;n < Longitud de registro;n++)
              { if (array[n] no contiene ')
                    aux := aux + array[n];
              }
            }

            else
            {  aux := registro;
              aux := Eliminar espacios en blanco de aux;
            }
            if (aux es cadena vacía ó espacio en blanco y j es diferente
             a numcol)
              aux := Null;
             Escribir a archivo rutarecipiente el contenido de aux seguida
             de tabulador;
             llave_pk := llave_pk + aux;
             aux := Cadena vacía;
          }
           Escribir a archivo rutarecipiente el contenido de llave_pk seguida;
           de salto de renglón;
        }
      }
       Cerrar archivo rutarecipiente;
    }
```

Tabla 5.32: Algoritmo que crea un "Script" para realizar la migración de datos (Continuación).

Entrada. Ruta del archivo SQL a ejecutar.
Salida. Nueva tabla en la base de datos(migración de información).
Procedimiento. Cargar script(cargaSQL).

```
{
  Crear conexión con Postgres;
  String creatabla := Leer la primer línea del archivo ScriptSQL;
  Ejecutar creatabla;
  while(existan líneas por leer en archivo)do
  {
      creatabla := Leer siguiente línea del archivo;
      Ejecutar creatabla;
  }
  Teminar conexión con Postgres;
}
```

Tabla 5.33: Algoritmo que ejecuta un script en SQL para migrar información a la base de datos Posgres.

Capítulo 6

Resultados

En los capítulos 4 y 5 hemos visto como se diseñó y realizó la implementación del sistema, en éste capítulo veremos los resultados arrojados por el nuevo **SI** desarrollado.

6.1. Resultados del nuevo sistema

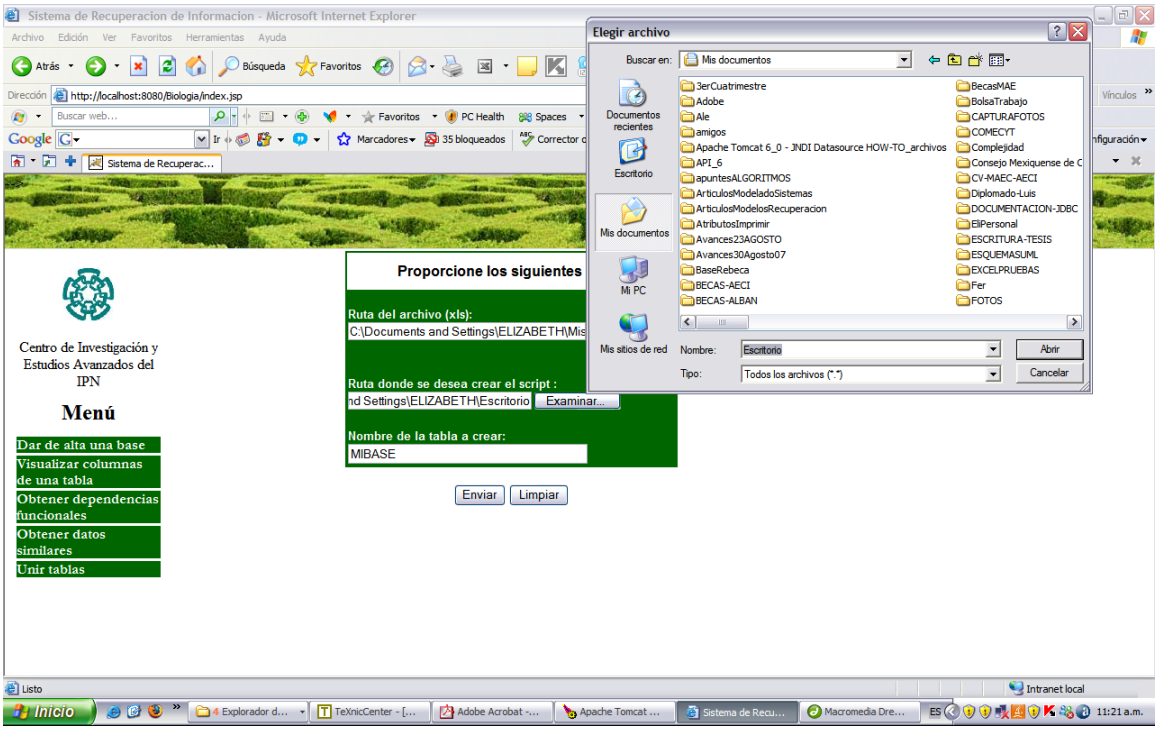
El SI está dividido, como sabemos, en cinco módulos, cada módulo fué implementado con varias de las funciones mencionadas en el capítulo anterior. A continuación se realiza un descripcion de los resultados de cada módulo diseñado, y de la manera en la que fueron involucradas algunas de las funciones creadas. Incluyendo pantallas de captura del funcionamiento del sistema.

- Módulo de eliminación de redundancia.

  Dar de alta una tabla implica migrar la información contenida en tablas de Excel al manejador de bases de datos Posgres. El proceso de migración de la información implica a su vez, para este proyecto, eliminar la redundancia en los datos. En esta parte se muestra como el usuario de manera transparente colabora con la migración de su información introduciendo únicamente la ruta del archivo en formato "xls", la ruta donde se desea crear su archivo de respaldo ("script"), y el nombre de la tabla con la que va a reconocer su información. Véase la figura 6.1.

- Módulo de visualización de subtablas.

  Se realizó una intefaz capaz de darle la opción al usuario de elegir una tabla de entre todas las que ya estan dadas de alta en el sistema, así como la cantidad de renglones que desea visualizar. Las tablas dadas de alta en el sistema se muestran de manera dinámica de tal forma que cualquier tabla dada de alta en el sistema puede ser vista por cualquier usuario facilitando las investigaciones de toda persona dentro del área de investigación correspondiente al tipo de datos que se están manipulando. Véase la figura 6.2. Una vez que se ha seleccionado la tabla a visualizar se muestra al usuario una tabla creada dinámicamente de



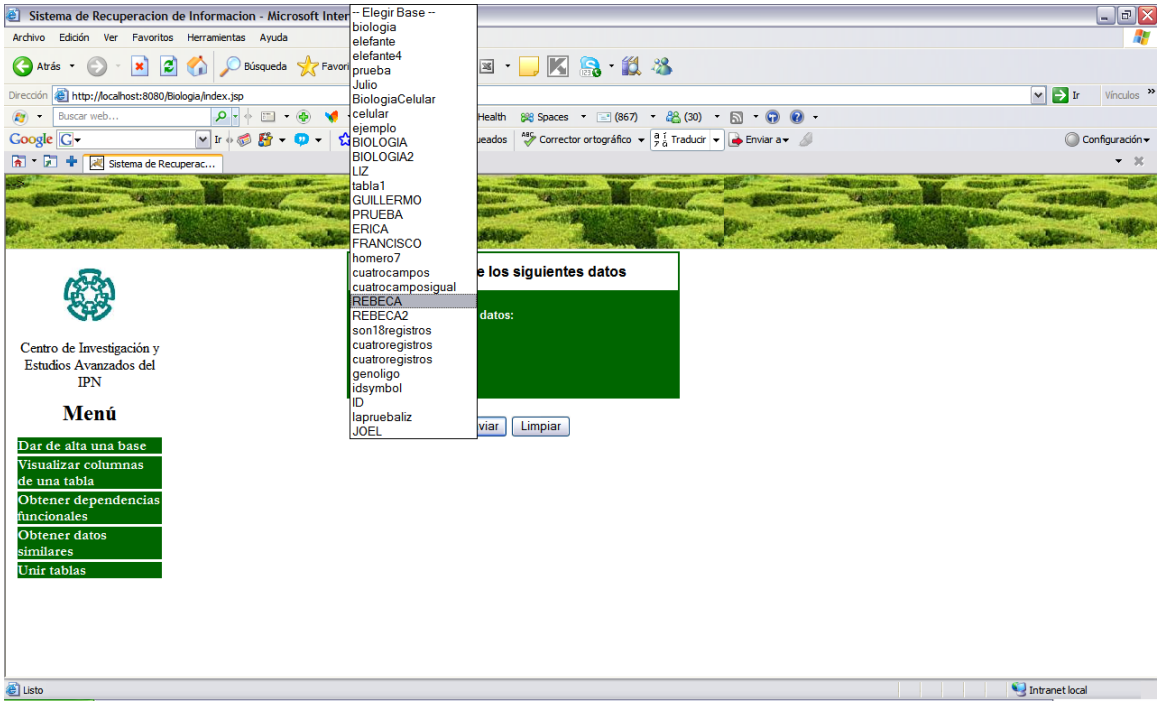
Se puede observar que los parámetros de entrada del sistema son la ruta de la tabla en EXCEL, ruta donde se desea crear un "script" para respaldar la información o bien, hacer copias de ella, y finalmente el nombre con el que una tabla será identificada. Para hacer mas comoda la búsqueda de los archivos de EXCEL se colocó un botón para examinar todos los archivos de usuario evitando que de manera manual se escriban las rutas.

Figura 6.1: Interfaz para dar de alta una tabla

acuerdo al número de campos que contiene cada tabla, con la opción de que cada campo pueda ser seleccionado para ver únicamente los datos que al usuario le interesan. Para que se mostrara este tipo de información se aplicó el concepto de *metadatos*(véase el apéndice A); donde por cada tabla dada de alta se actualiza otra tabla dentro de la base de datos cuyos datos son el *nombre de la tabla y los atributos que contiene cada tabla*. Véase la figura 6.3. Ahora bien, la forma en la que el sistema muestra el resultado de la consulta puede verse en la figura 6.4.

- Módulo de dependencias funcionales.

  El resultado de este módulo es una interfaz para el usuario donde dado el nombre de una tabla a analizar, de una lista de tablas dadas de alta en el sistema, el número de registros a examinar, los atributos a analizar, la manera en la que

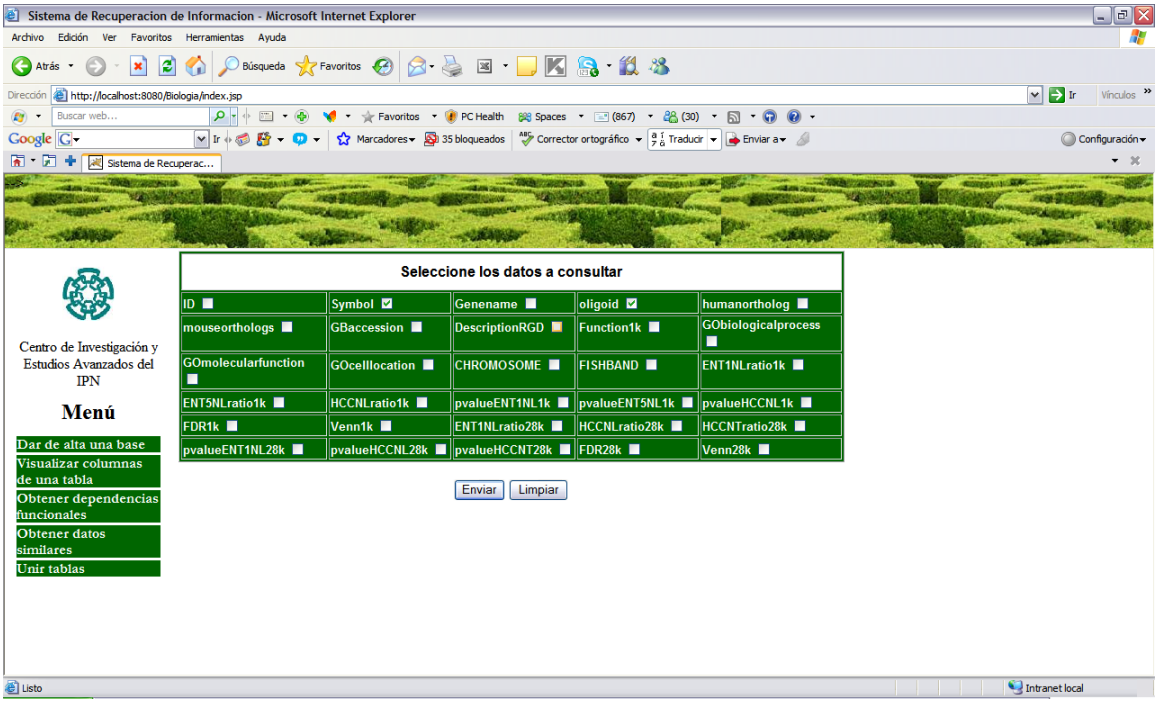


Se puede observar que los parámetros de entrada son el nombre de la tabla y el número de registros a visualizar dentro de la tabla. La manera en la que se puede seleccionar la tabla es de manera dinámica pues el sistema muestra automáticamente que tablas están dadas de alta en el sistema.

Figura 6.2: Interfaz para visualizar los campos de una tabla eligiendo una tabla dada de alta en el sistema.

se ordenarán los datos deacuerdo a los atributos dados y el tipo de búsqueda de dependencia funcional, ya sea por igualdad de datos (empatamiento de patrones) o bien mediante el uso de funciones de similitud, el sisteme devuelve un archivo en la unidad "C" con los resultados de la búsqueda. De acuerdo al tipo de búsqueda seleccionada se generan archivos con nombres distintos. Nótese que el nombre de la tabla y el número de registros elegidos son independientes del nombre del archivo resultante. Abajo se describen las características de cada archivo generado. Véase las figuras 6.5 y 6.6.

1. `DependenciasOrdenFrecuencia.txt`. Se genera cuando la selección de la búsqueda es la siguiente: 1)Búsqueda de dependencias por empatamiento de patrones y 2) *orden de atributos por frecuencia descendente*, esto es que los datos fueron ordenados de acuerdo a los atributos desde el que tenía mayor cantidad de valores repetidos hasta el que contenía menor cantidad de datos repetidos. El archivo resultante contiene conjuntos de atributos



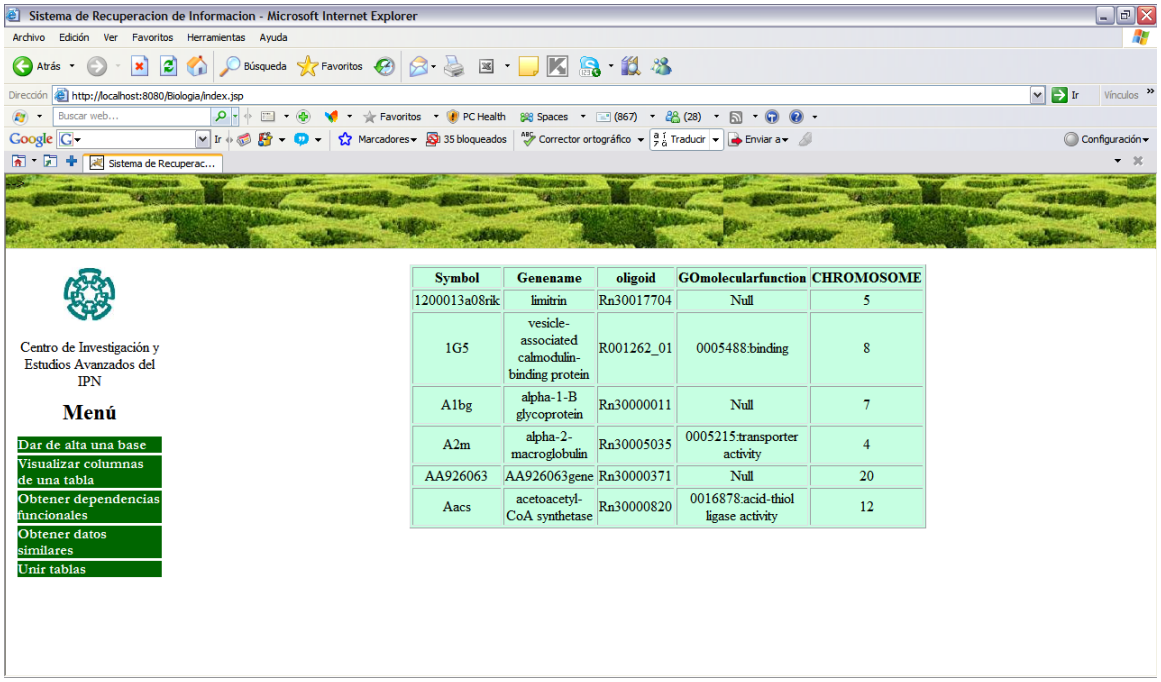
Los parámetros de entrada son los campos de interés para el usuario

Figura 6.3: Interfaz para visualizar los campos de una tabla eligiendo los atributos de interés para el usuario.

expresados en su forma númerica, cada línea del archivo corresponde a un conjunto distinto de atributos donde el complemento de dichos valores númericos por conjunto se encontró que dependen del conjunto de atributos escritos en el archivo.

2. `DependenciasNombreOrdenFrecuencia.txt`. Surge de la misma selección de búsqueda que en el anterior por lo que el resultado es el mismo, la diferencia es que los resultados los muestra con los nombres de los atributos, a diferencia del archivo anterior, el cual muestra resultados numéricos. Este archivo se genera al mismo tiempo que `DependenciasOrdenFrecuencia.txt`.

3. `DependenciasOrdenFrecuenciaDADO.txt`. Se genera cuando la selección de la búsqueda es la siguiente: 1)Búsqueda de dependencias por empatamiento de patrones, y 2) *orden de atributos dado por el usuario*, esto es que los datos fueron ordenados de acuerdo a un orden arbitrario o no dado por el usuario. El archivo resultante contiene conjuntos de atributos expresados en su forma númerica, cada línea del archivo corresponde a un conjunto distinto de atributos donde el complemento de dichos valores númericos por conjunto se encontró que dependen del conjunto de atributos escritos en el archivo.



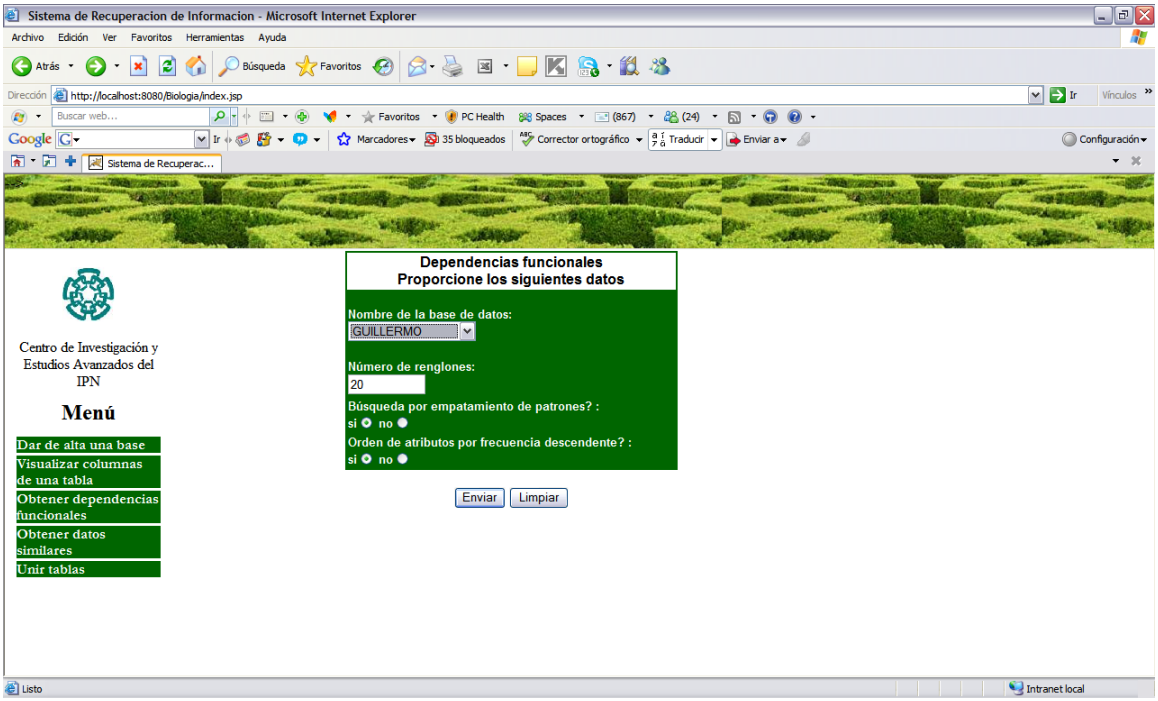
Muestra el número de registros dado por el usuario así como únicamente los campos que son de interés para el usuario.

Figura 6.4: Interfaz para visualizar los registros y campos de interés de una tabla elegida por el usuario.

4. `DependenciasNombreOrdenFrecuenciaDADO.txt`. Surge con la selección de búsqueda que el del archivo `DependenciasOrdenFrecuenciaDADO.txt` por lo que el resultado es el mismo, la diferencia es que los resultados se muestran con los nombres de los atributos, a diferencia del último mencionado que muestra resultados numéricos. Este archivo se genera al mismo tiempo que `DependenciasOrdenFrecuenciaDADO.txt`

5. `DependenciasSimilitudOrdenFrecuencia.txt`. Se genera cuando la selección de la búsqueda es por: 1) Búsqueda de dependencias funcionales haciendo uso de funciones de similitud, y 2) *orden de atributos por frecuencia descendente*. El resultado de las dependencias depende mucho del grado de similitud que le haya proporcionado el usuario por cada tributo seleccionado. De la misma manera que los archivo anteriores, cada línea del archivo corresponde a un conjunto distinto de atributos donde el complemento de dichos valores númericos, por conjunto, se encontró que dependen de los atributos escritos en el archivo.

6. `DependenciasSimilitudNombreOrdenFrecuencia.txt`. Se genera al mismo tiempo que el último archivo descrito, solo que los resultados los mues-



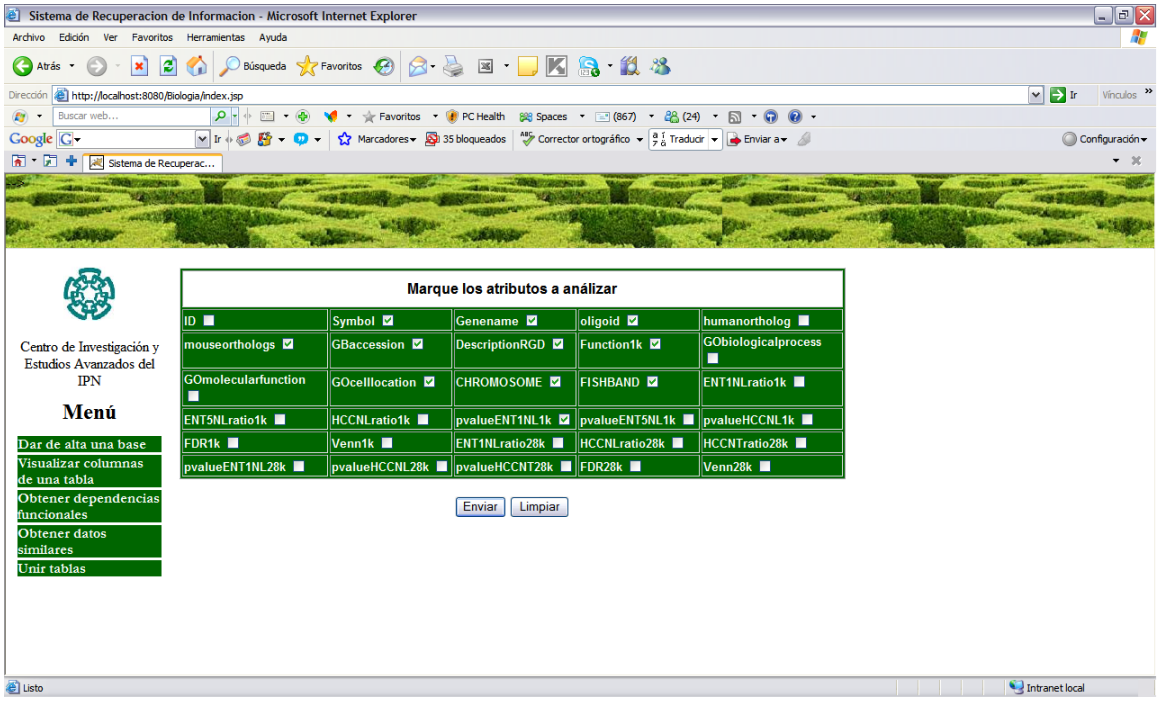
Muestra el nombre de la tabla a analizar, número de registros a examinar y la manera en la que se va a realizar la búsqueda de dependencias funcionales, para este caso es por empatamiento de patrones y por orden de atributos descendente, es decir, desde el atributo que posee mayor cantidad de datos repetidos hasta el que contiene menos repeticiones. Esta interfaz se muestra para cualquier tipo de búsqueda de dependencia funcional.

Figura 6.5: Interfaz para seleccionar la manera en la que se va a realizar la búsqueda de dependencias funcionales, inpendientemente del tipo de búsqueda.

tra con el nombre del atributo en vez de su valor numérico.

7. `DependenciasSimilitudOrdenFrecuenciaDADO.txt`. Se genera cuando la selección de la búsqueda es la siguiente: 1)Búsqueda de dependencias mediante funciones de similitud, y 2) *orden de atributos dado por el usuario*, esto es que los datos fueron ordenados de acuerdo a un orden arbitrario o no dado por el usuario. El archivo resultante contiene conjuntos de atributos expresados en su forma númerica, cada línea del archivo corresponde a un conjunto distinto de atributos donde los atributos que no aparecen dentro de un conjunto resultaron ser dependientes de los que están escritos en el archivo.

8. `DependenciasSimilitudNombreOrdenFrecuenciaDADO.txt`. Surge de la misma selección de la misma selección de datos de entrada que el último



Muestra los atributos que posee la tabla seleccionada así como los atributos que se desean analizar mediante la selección de ellos por medio de una caja que le sigue a cada nombre de atributo.

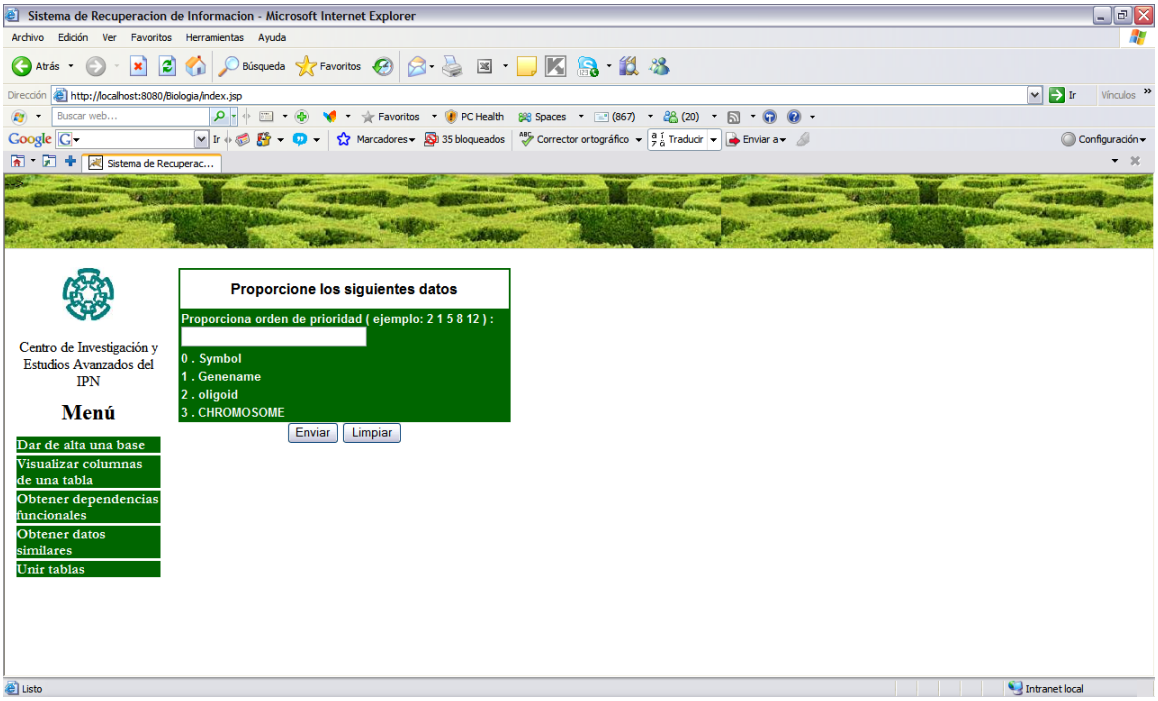
Figura 6.6: Interfaz para seleccionar los atributos a analizar en la búsqueda de dependencias funcionales independientemente del tipo de búsqueda.

archivo descrito por lo que el resultado es el mismo, la diferencia es que los resultados se muestran con los nombres de los atributos, a diferencia del anterior. Este archivo se genera a la par que `DependenciasSimilitud-OrdenFrecuenciaDADO.txt`.

La manera en que todos los archivos muestran el resultado de la búsqueda de dependecias funcionales se ve como en la figura 6.10.

- Módulo de composición de tablas. Este módulo se encarga de unir dos tablas, véase la figura 6.11, mediante un atributo en común. Ambas tablas comparten el mismo valor sobre un atributo seleccionado por el usuario, de tal forma que el sistema devuelve como resultado registros completos sobre atributos que comparten ambas tablas donde el valor en el atributo elegido es igual para todos los registros.

  La manera en la que procede es mediante la intefaz gráfica donde se le pide al usuario que eliga dos tablas, previamente dadas de alta en el sistema, distintas, de lo contrario mandará un aviso de error para que el usuario eliga otra u otras



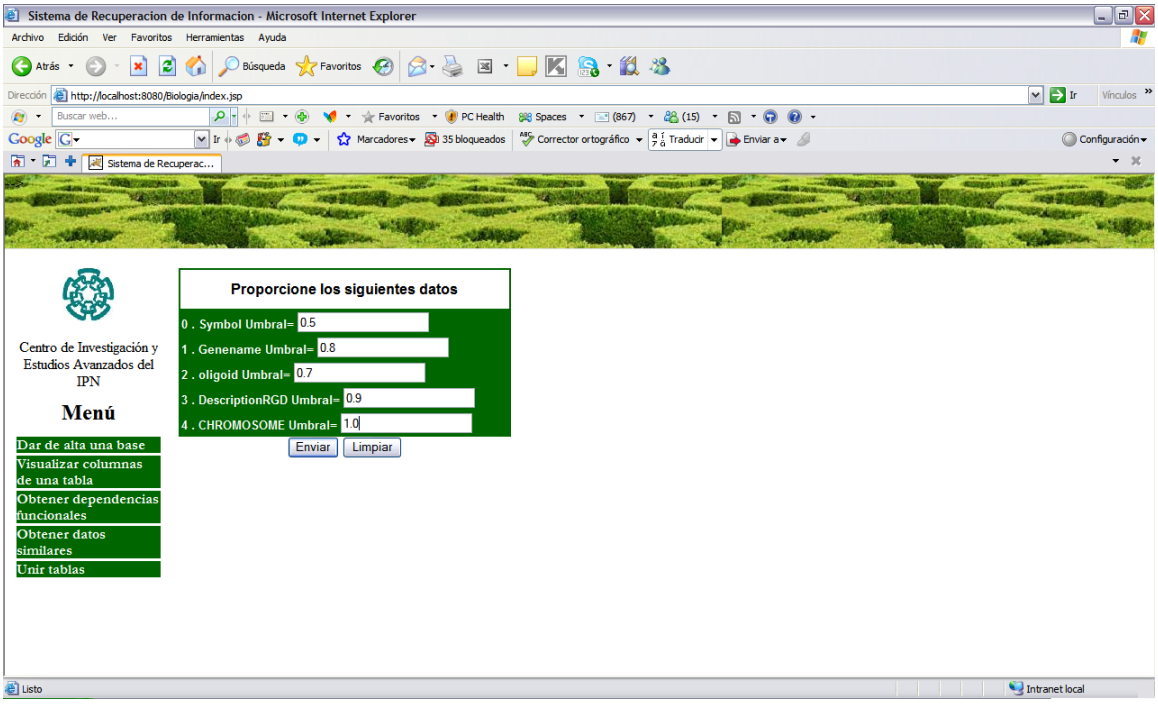
Muestra la interfaz para la búsqueda de dependencias funcionales por empatamiento de patrones y orden de atributos dado por el usuario, este proceso genera los archivos `DependenciasOrdenFrecuenciaDADO.txt` y `DependenciasNombreOrdenFrecuenciaDADO.txt`.

Figura 6.7: Interfaz que se muestra para búsqueda de dependencias funcionales por empatamiento de patrones y un orden de atributos dado por el usuario.

tablas distintas, una vez realizado esto el sistema busca los atributos que ambas tablas tienen en común, véase la figura 6.13, de no haber atributos en común no se podrá realizar la unión y se le enviará al usuario un mensaje que lo indique, véase la figura 6.12. De haber atributos en común le pide al usuario que eliga uno de ellos, con esto el sistema procesa los datos y realiza la unión y muestra la consulta al usuario.

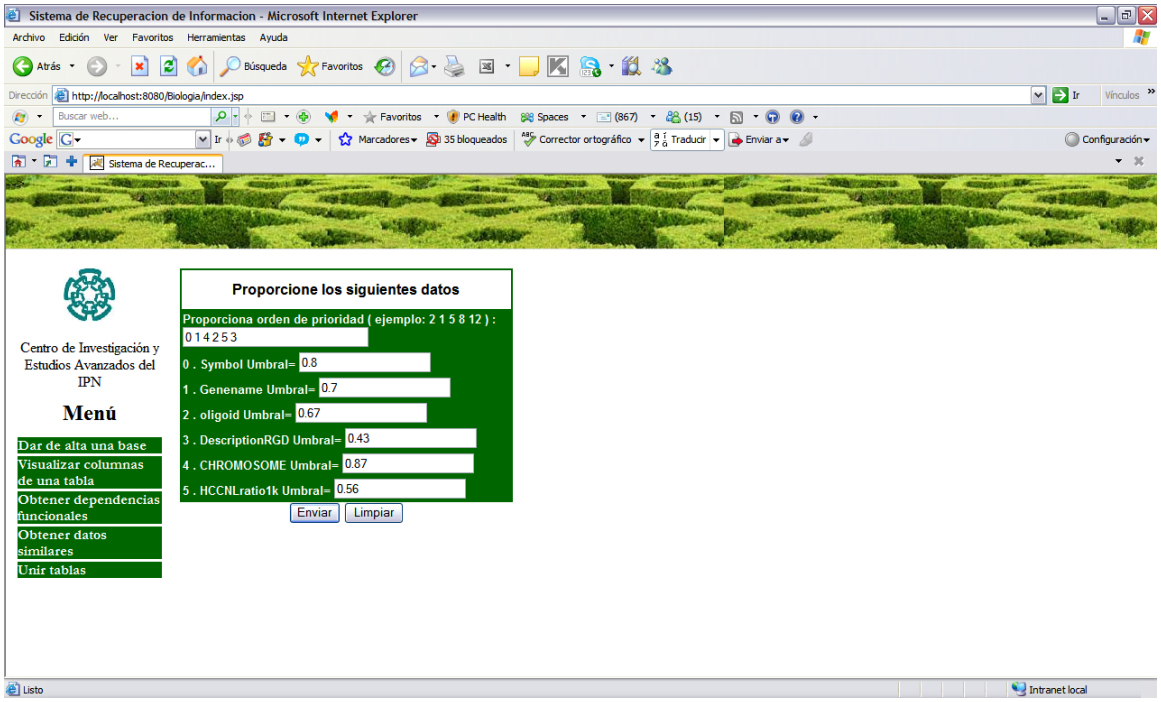
- **Módulo para búsqueda de registros**. La manera en la que procede este módulo es mediante la búsqueda de registros sobre una tabla seleccionada por el usuario. El sistema identifica de manera automática todos los atributos que contiene dicha tabla por lo que procede a pedirle que seleccione el atributo sobre el cual se va a realizar el análisis, así como el dato mediante el cual se realizarán todas las comparaciones de todos los datos sobre dicho atributo. El resultado de esta búsqueda será todos los datos sobre el atributo elegido cuya semejanza entre el dato introducido y los valores devueltos estará en el rango de similitud

dado, esto es, que los datos devueltos por el sistema tengan un porcentaje de semejanza al dato dado con valor mayor o igual al dado en la cota inferior y menor o igual al de la cota superior de similitud. Véase la figura 6.15 y 6.16. Una vez que se ha dado los datos similares se da la opción de que el usuario eliga los datos que le son de interés para proceder a ver el registro completo de los datos elegidos. Véase la figura 6.17.



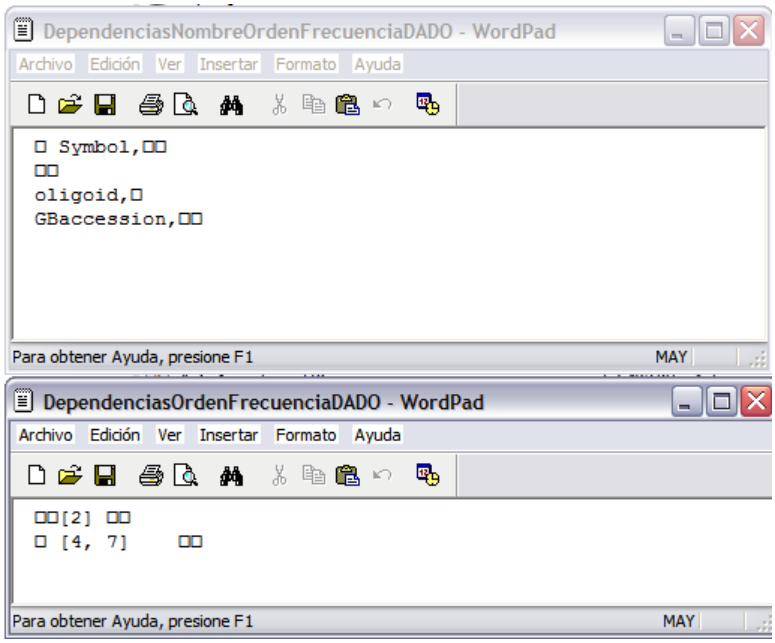
Muestra la interfaz para la búsqueda de dependencias funcionales mediante el uso de funciones de similitud, el ejemplo que se muestra pide como datos los umbrales de similitud para cada atributo seleccionado por el usuario, dado que el orden de los atributos es por orden de frecuencia de datos descendente no se pide el orden en el que van los datos pues el sistema lo proporciona. Mediante este proceso se generan los archivos `DependenciasSimilitudOrdenFrecuencia.txt` y `DependenciasSimilitudNombreOrdenFrecuencia`.

Figura 6.8: Interfaz que sirve para obtener dependencias funcionales mediante funciones de similitud, pide los umbrales por atributo seleccionado para ser analizado.



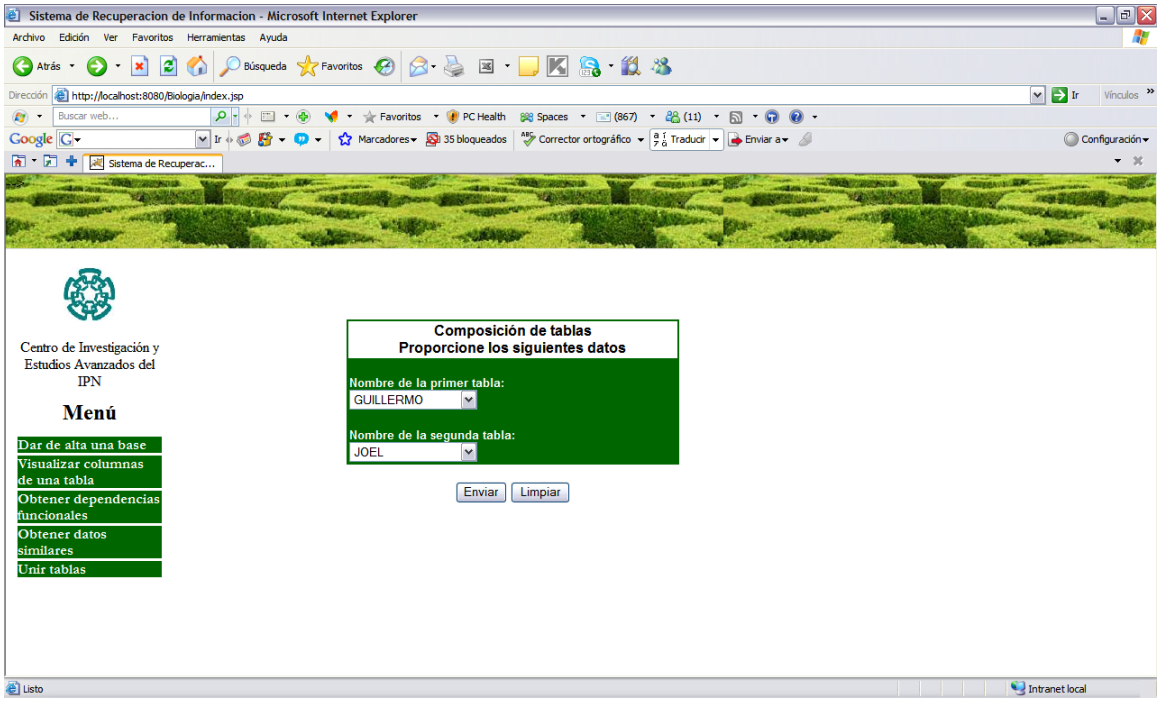
Muestra la interfaz para la búsqueda de dependencias funcionales mediante el uso de funciones de similitud, el ejemplo que se muestra pide como datos los umbrales de similitud para cada atributo seleccionado por el usuario, dado que el orden de los atributos es por orden diferente al de la frecuencia de datos descendente se pide el orden en el que van los datos. Mediante este proceso se generan los archivos `DependenciasSimilitudOrdenFrecuenciaDADO.txt` y `DependenciasSimilitudNombreOrdenFrecuenciaDADO`.

Figura 6.9: Interfaz que sirve para obtener dependencias funcionales mediante funciones de similitud, pide los umbrales por atributo seleccionado para ser analizado y el orden de los atributos mediante los cuales se van a ordenar los datos.



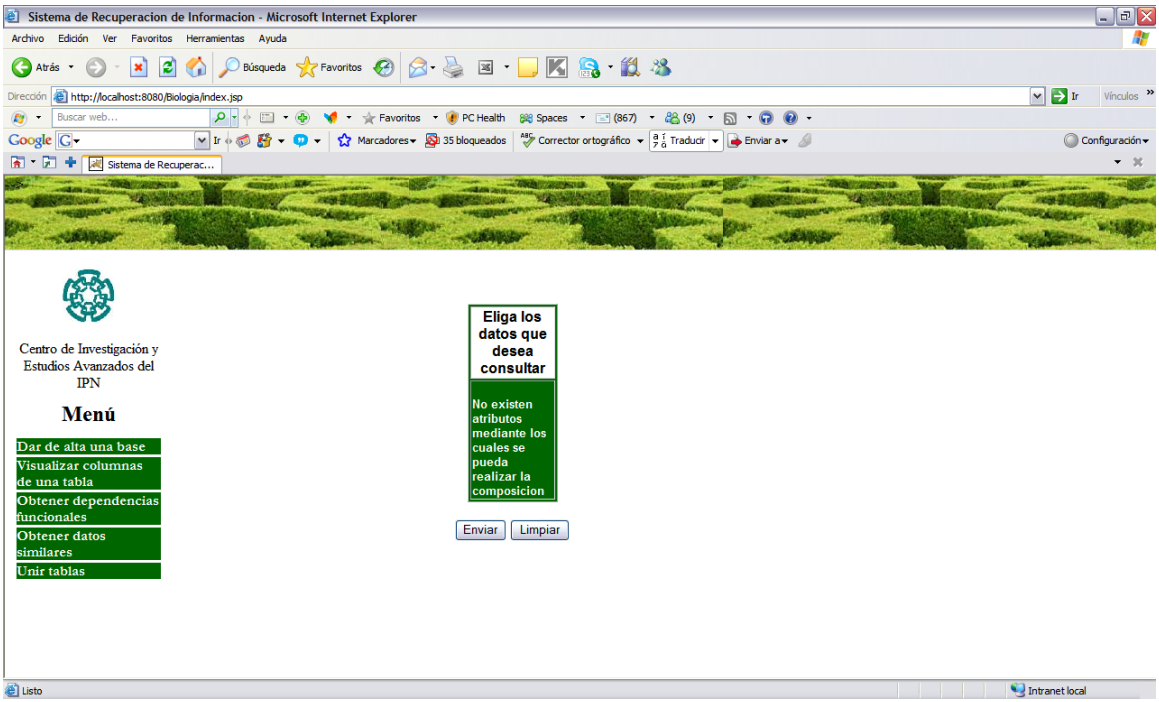
De esta manera se ven los archivos de salida en la búsqueda de dependcias funcionales independientemente de la manera en la que se hicieron. Cada renglón del archivo representa un grupo de atributos donde los atributos que no aparecen dependen de los atributos escritos.

Figura 6.10: Forma de despliegue de resultados al realizar las búsquedas de dependencias funcionales.



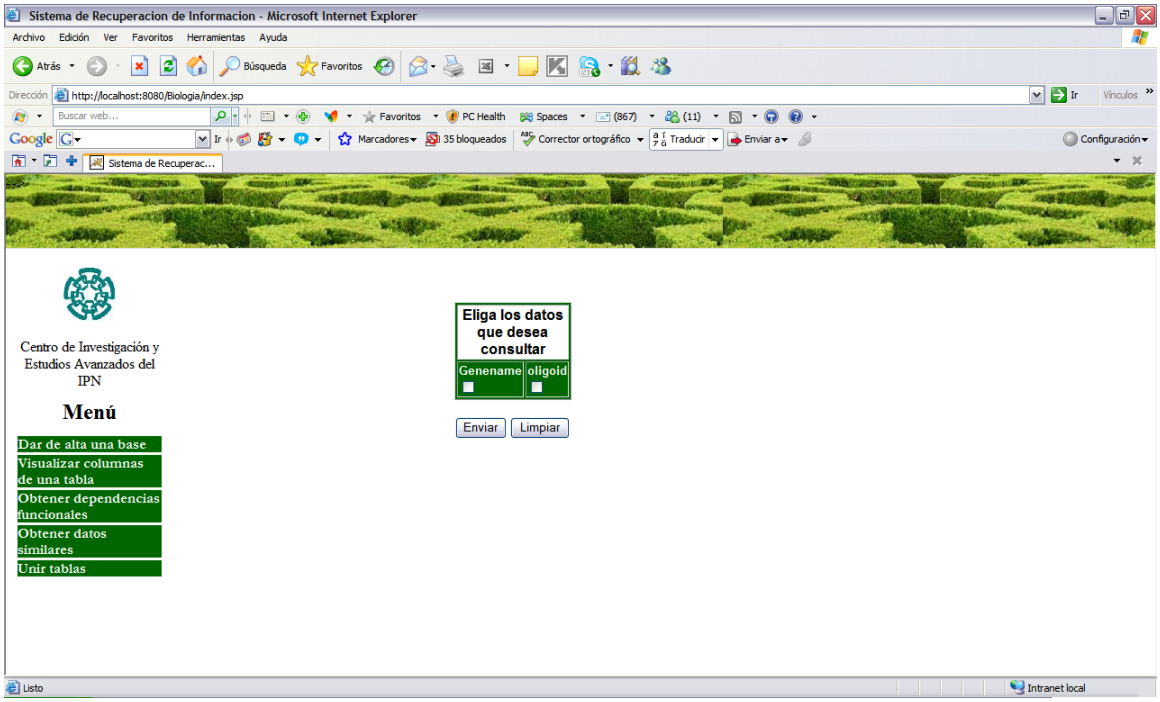
De esta manera se muestra la interfaz para que el usuario seleccione las tablas que desea unir.

Figura 6.11: Interfaz para unir dos tablas seleccionadas por el usuario.



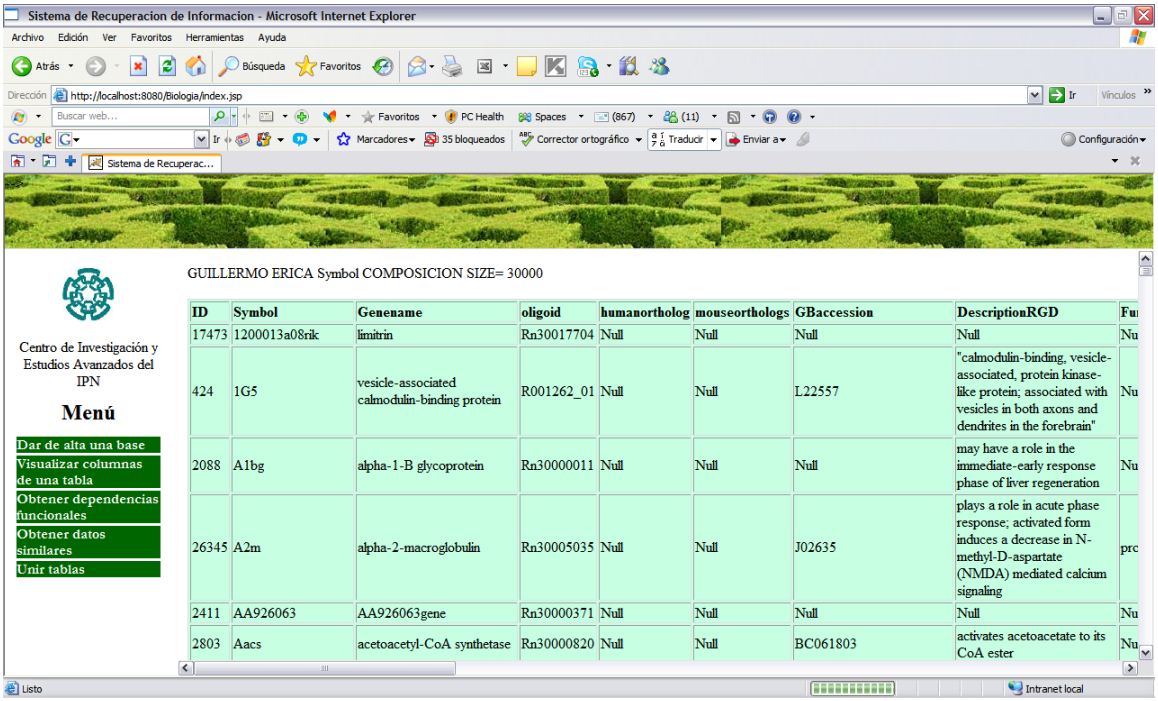
De esta manera se muestra un aviso al usuario de que no se puede realizar la composición de dos tablas pues no comparten algún atributo en común.

Figura 6.12: Mensaje de salida cuando dos tablas no se pueden unir.



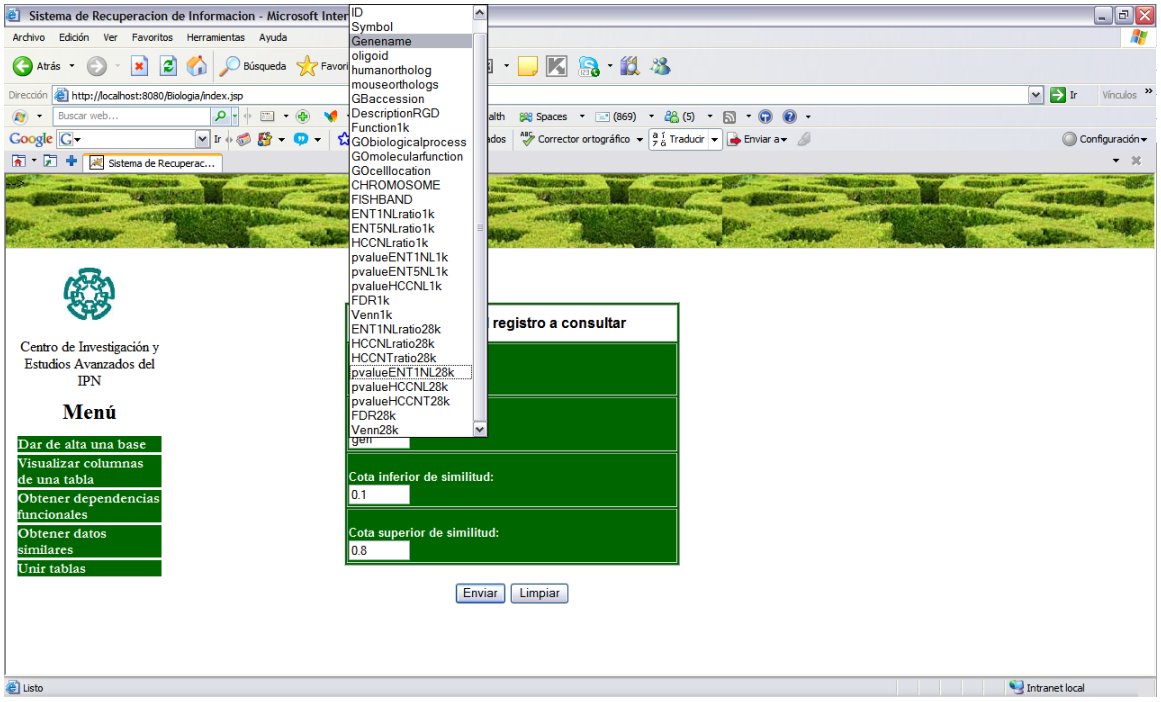
El sistema muestra únicamente los atributos que ambas tablas seleccionadas comparten.

Figura 6.13: Atributos comunes en dos tablas a unir.



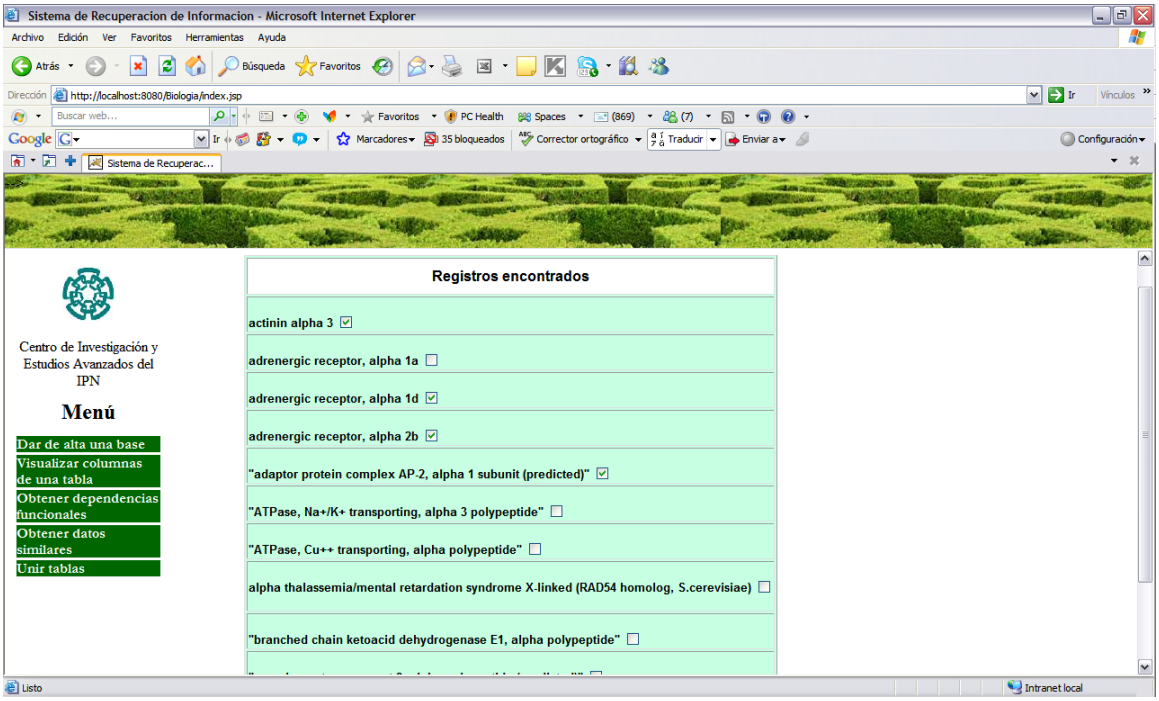
Se muestra el resultado de la unión de dos tablas un llamada “Guillermo” y otra “Erica”, ambas comparten los mismos atributos, el atributo por el que se unieron fué mediante `Symbol`.

Figura 6.14: Resultado de la unión de dos tablas que tienen los mismos atributos en común.



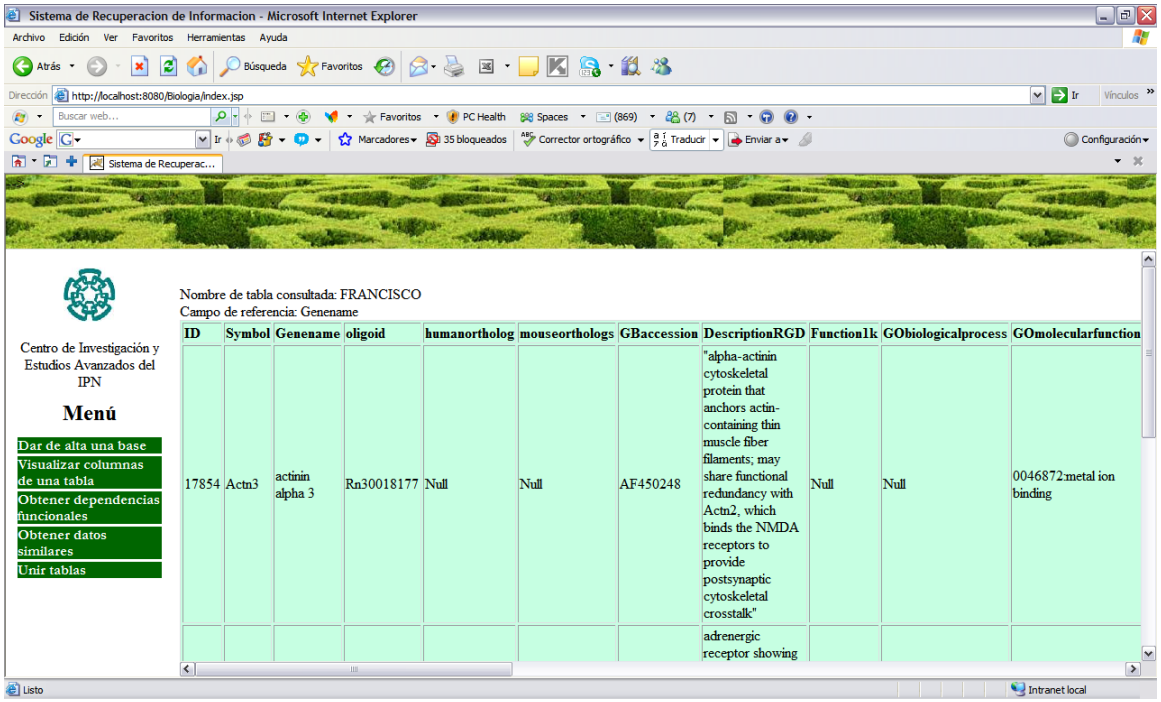
Se muestra la interfaz para el usuario donde el sistema da a elegir al usuario el atributo sobre el cual se realizará la búsqueda, y pide como parámetros adicionales el dato mediante el cual se realizarán las comparaciones con los demás datos, así como la cota inferior y superior de similud.

Figura 6.15: Parámetros de entrada para realizar la búsqueda de datos similares.



Muestra los datos encontrados en el rango de similitud dado, el campo sobre el cual se realizó la búsqueda fué `Genename` y el dato a comparar fué `alpha`.

Figura 6.16: Resultado de una búsqueda de registros similares dado el valor de un dato.



Se muestran los registros completos de los datos de interés (elegidos) para el usuario.

Figura 6.17: Resultado de una consulta como consecuencia de la búsqueda de registros similares a un dato dado.

Capítulo 7

Conclusiones y trabajo futuro

Uno de los campos en los que se está enfocando la minería de datos es en el área biológica, debido a la abundante información que se está generando. Por ejemplo, en el campo de la Genómica, se está obteniendo información extensa sobre la secuencia del genoma de los organismos biológicos. Otra importante fuente de información corresponte al surgimiento de anomalías genéticas por las que estamos pasando hoy día. Se requieren de métodos eficientes en la búsqueda de datos sobre grandes bases de datos que contienen información de suma importancia para la realización de las investigaciones. El método que utilizamos para el análisis de datos biológicos es mediante el uso de funciones de similitud. El sistema desarrollado en este proyecto nos ayuda a encontrar similitudes sobre los datos más utilizados en el área de Biología Celular, estás comparaciones se realizan de entre una gran cantidad de datos recabados de diversas bases de datos biológicas de distinta procedencia, ayudando a los investigadores a obtener información de manera simultánea sobre distintos genes y sus características, mostrándolos en un ambiente amigable y proporcionando la opción de visualizar solo aquellos datos que sean similares en el porcentaje de similitud dado. Otro módulo creado y que es también importante es la creación de "vistas" para la unión de distintas tablas hechas por usuarios que han realizado investigaciones previas, enriqueciendo aún más la búsqueda de información. Se desarrolló un módulo para la búsqueda de dependencias fucionales capaz de encontrar las dependencias entre distintos datos no necesariamente biológicos, resolviendo el problema de búsqueda de atributos llave sobre cualquier tabla de datos con información no necesariamente conocida por el usuario. Finalmente, se creó una interfaz de usuario para que la interacción entre sistema-usuario sea más fácil y rápida de manipular, así como la creación de un módulo para el respaldo de la información de los usuarios y la migración de los datos, ya recabados en Microsof Excel a un manejador de bases de datos potente como lo es Posgres. Otra ventaja del sistema es que está desarrollado bajo los términos de la licencia GNU, esto es, que se utilizaron herramientas de software libre por lo que el código puede ser reciclado para la creación de una extensión del sistema, para su mejora o bien para el reciclado de algunos de sus módulos.

A pesar de todas la ventajas del sistema aún se le pueden agregar más módulos para ayudar a realizar investigaciones, pero algo que he concluido es que todo sistema

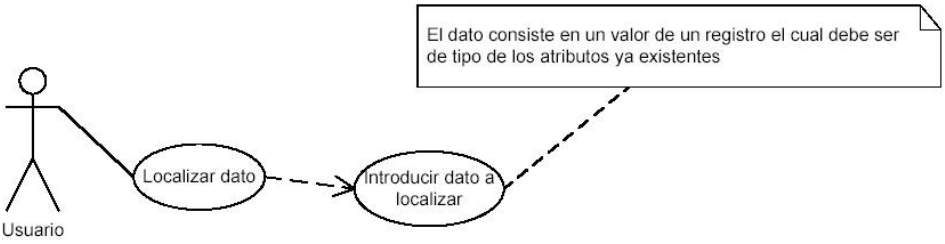


Figura 7.1: Caso de uso de Localizar dato

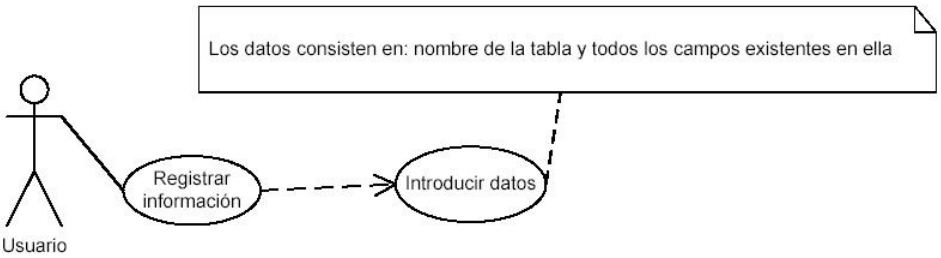


Figura 7.2: Caso de uso de Registrar información

está basado en la solución de problemas, aunque no todo sistema estará completo mientras existan problemas que resolver.

Así, con el presente sistema se define el cimiento de una aplicación más compleja que satisfaga la inquietud y necesidades de los investigadores en Bioinformática. La aplicación computacional desarrollada será incorporada en el servidor de Departamento de Biología Celular para su uso.

En cuanto al trabajo a futuro a realizar se requiere crear una aplicación capaz de homogeneizar la información a consultar que funcione como traductor bidireccional entre el usuario y las bases de datos a consultar. Si bien todas las bases contienen información sobre una misma consulta, no todas las bases la representan de la misma forma. También se proponen nuevos módulos para el sistema como son "Localizar un dato en particular" mediante el uso del lenguaje SQL, y "Registro de información" sobre nuevos descubrimientos realizados, para lo cual se han realizado algunos diseños en UML para estos módulos. Véase las figuras 7.1, 7.2 y 7.3.

Otros procesos que se pueden agregar al sistema son el acceso a diversas bases mediante *sockets*; en procedimientos fuera de línea, la creación de un "minimanejador" de BD para búsquedas particulares, y síntesis de tablas a partir de la información recopilada fuera de línea.

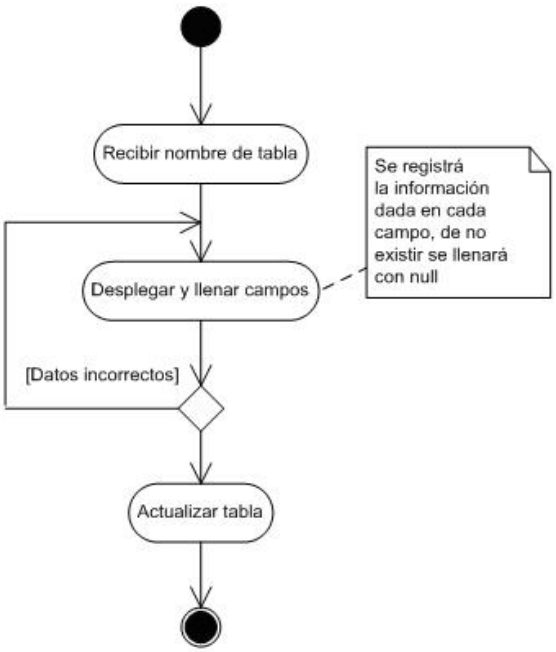


Figura 7.3: Diagrama actividades: Registrar información

Apéndice A

Glosario

Unicódigo. Antes de que se inventara el valor *unicódigo*, había cientos de sistemas para asignar estos valores, pero ninguno se adecuó para todos los tipos de letras, puntuación, y símbolos técnicos de uso común.

Estos sistemas de asignación de valores también tenían conflictos unos con otros. Esto es, dos sistemas podían usar el mismo número o valor para dos diferentes caracteres, o bien utilizaban diferentes numeros para el mismo caracter.

El unicódigo provee un valor único para cada caracter sin importar la plataforma, el programa o el lenguaje que se utilice. El *estándar unicódigo* ha sido adoptado por líderes industriales como `Apple, Hp, IBM, JustSystem, Microsoft, Oracle, SAP, Sun, Sybase, Unisys entre otras` , y es la forma oficial para implementar `ISO/IEC 10646`. El cual es soportado en varios sistemas operativos, todos los sistemas modernos, y otros productos.

La definición de *elementos de tipo texto* frecuentemente cambia dependiendo de la forma en la que sean procesados. Por ejemplo, en el lenguaje español ordenar la "ll" cuenta como un solo elemeto de texto. Para evitar decidir que es un elemento de texto y que no lo es, el *estándar unicódigo* define *elementos código* (comunmente llamados "caracteres"). En el caso del lenguaje español "ll", el *estándar unicódigo* define a cada "l" como un elemendo de código separado. La tarea de combinar dos "ll" juntas para una ordenación alfabética es hacia la izquierda [31].

Script. Un "script" es un conjunto de instrucciones creadas para automatizar tareas.

Metadatos. Los metadatos son datos altamente estructurados que describen información, describen el contenido, la calidad, la condición y otras características de los datos.Es información sobre la información o datos sobre datos [32].

Apéndice B

Programas realizados

B.1. Disco compacto anexo a esta tesis

Incluímos en la tesis un disco compacto cuyo contenido se describe abajo.

El directorio `Documentos` contiene la carpeta `Presentaciones` el cual marca las presentaciones que se han realizado a lo largo del desarrollo de este proyecto, y la carpeta `Registro` en la cual se encuentra el protocolo de tesis mediante el cual se registró este trabajo, así como el documentos de tesis que está leyendo.

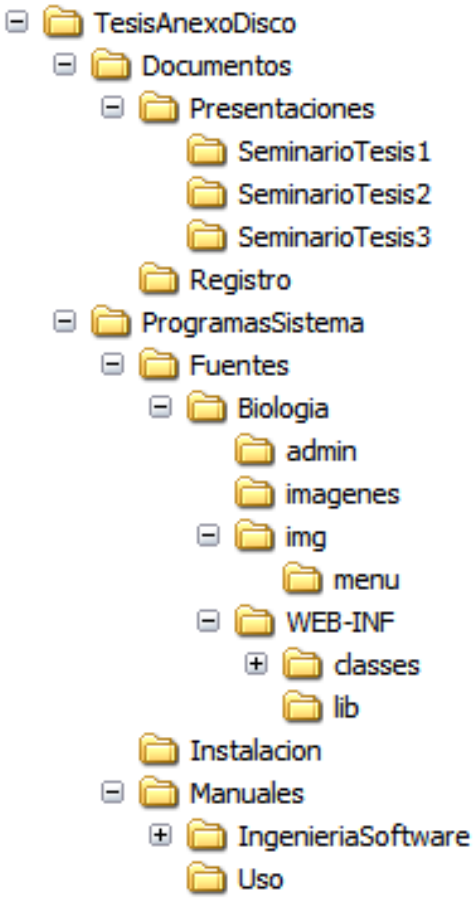
El directorio `ProgramasSistema` incluye el código fuente del sistema desarrollado, contenido en la carpeta `Fuentes`, y un manual de instalación del sistema en `Instalacion`. El directorio `Manuales` contiene dos directorios `IngenieriaSoftware` y `Uso`, en el primero se muestran los diagramas de diseño del sistema en UML ,y en el segundo el manual de usuario. Véase la figura B.1.

B.2. Estructura de directorios del sistema y programas realizados

Desarrollamos los programas para la creación de un sistema de recuperación de información. Los programas desarrollados y la manera en la que debe ser instalados para su ejecución son mostrados como abajo se suscribe.

Las clases programadas en el lenguaje JAVA son las siguientes:

`ActualizaTablas.java`
`AntecedentesBean.java`
`Caracteres.java`
`Composicion.java`
`Conexion.java`
`Consecutivos.java`
`ConsultaBase.java`



Estructura de directorios anexo al disco compacto de esta tesis.

Figura B.1: **Disco compacto anexo a esta tesis.**

`ConsultaBean.java`
`Contiene.java`
`Dependencias.java`
`Esmenorque.java`
`Estadisticas.java`
`EstructuraFrecuencias.java`
`Finales.java`
`LeeFinales.java`
`MayorFrecuencia.java`
`MSC.java`
`Ordenacion.java`
`OrdVectores.java`
`Principal.java`
`QuickSort.java`

Script.java
Similitudes.java
TablasBean.java

Los programas JSPs son los siguientes:

ComposicionTablas.jsp
ComposicionTablas2.jsp
ComposicionTablas3.jsp
Consulta.jsp
Consulta2.jsp
ConsultaValores.jsp
DependenciaFuncional.jsp
DependenciaFuncional2.jsp
DependenciaFuncional3.jsp
DependenciaFuncionalEmpNewOr.jsp
Dependencias
ElegirRegistroSimilitud.jsp
ElegirRegistroSimilitud2.jsp
ElegirRegistroSimilitud3.jsp
ElegirRegistroSimilitud4.jsp
EliminarRedundancia.jsp
EliminarRedundancia2.jsp
VisualizarSubtabla.jsp
VisualizarSubtabla2.jsp
VisualizarSubtabla3.jsp

B.3. Instalación del sistema

La instalación del sistema requiere de la instalación de Java, el servidor web Apache-Tomcat y manejador de base de datos Posgres. El equipo de cómputo utilizado para el desarrollo del sistema y sus características se muestran en la figura B.2, para que el sistema funcione de manera óptica se requiere un servidor con al menos las mismas características de hardware.

Apache-Tomcat puede ser descargado de la página
`http://tomcat.apache.org/tomcat-6.0-doc/)`, para que pueda ser utilizado con Java se debe crear una estructura de directorios como la que se muestra en la figura B.3. También se deben crear nuevas variables de ambiente en el sistema operativo que se está utilizando. Ver más detalles en la página
`http://www.coreservlets.com/Apache-Tomcat-Tutorial/#Catalina-Home`.

En la tesis se anexa un directorio llamado "Biologia" este directorio es el que contiene el sistema completo y deber ser insertado en la misma ruta que se muestra en la figura B.3.

Figura B.2: Características del equipo utilizado para el desarrollo del sistema.

Dentro del directorio "Biologia" se encuentran contenidos los JSPs que sirven como interfaz para el usuario. Las clases de java se encuentran en la ruta `Biologia/webapps/WEB-INF/classes`. Véase la figura B.4.

Ahora bien, para que JAVA pueda accesar a las bases de datos de POSGRES se requiere de un JDBC, el que se requiere ya viene incluido en el software. La versión utilizada en el sistema es `postgresql-8.2-505.jdbc3`, este controlador se puede descargar en la página `http://jdbc.postgresql.org/download/postgresql-8.2-505.jdbc3.jar`, y debe ser almacenado en el directorio "lib". Véase la figura B.5 .

Ahora bien, dado que el sistema se desarrolló en JAVA con el `JDK versión 1.6.0_01`, véase la figura B.6, el cual puede ser descargado desde la página `http://java.sun.com/javase/downloads/index.jsp`. Una vez que se ha descargado el software se deben crear dos variables de ambiente en el sistema operativo, las cuales son utilizadas para personalizar el manejo de los programas y la comunicación entre ellos, sus nombres y valores se describen a continuación.

`JAVA_HOME`

con el valor:

`C:\Archivos de programa\Java\jdk1.6.0_01;C:\Archivos de programa\Java\jdk1.6.0_01\lib\tools.jar`

y

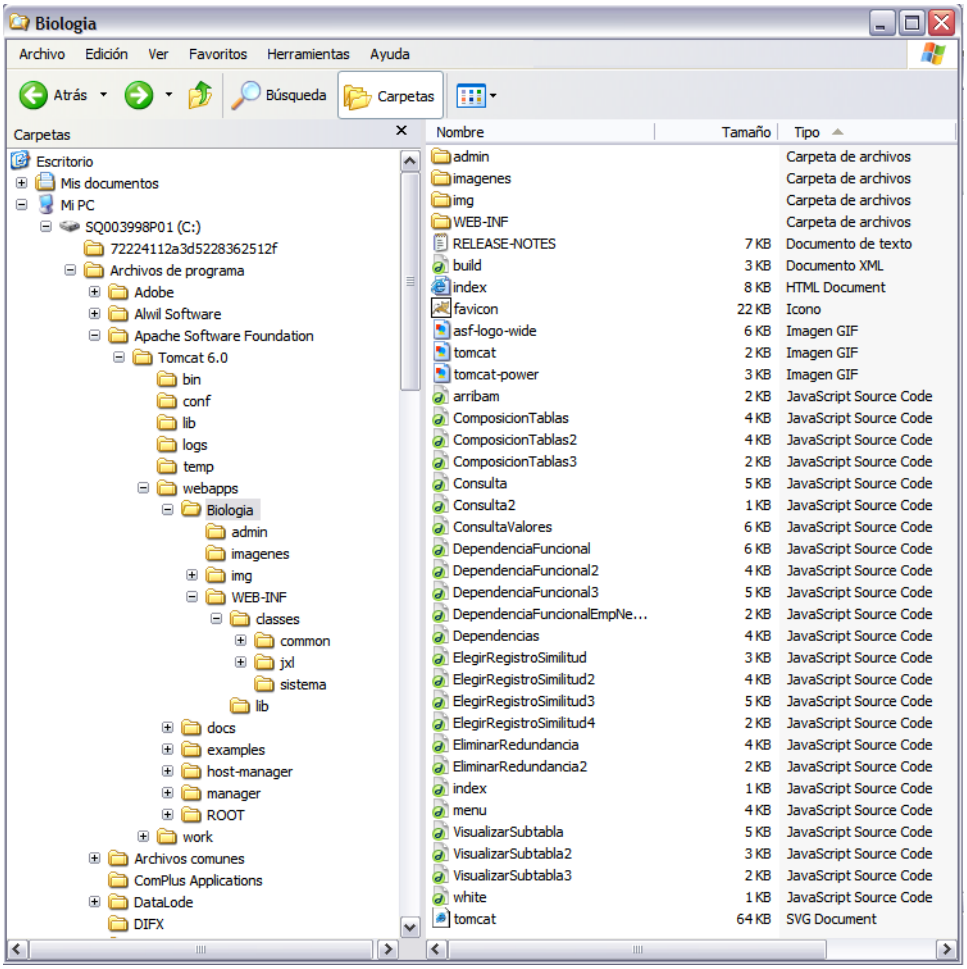
`CLASSPATH`

con el valor:

`%CLASSPATH%;C:\postgresql-8.2-505.jdbc3.jar;C:\Archivos de programa\PostgreSQL\8.2\jdbc ;c:\Archivos de programa\Apache Software Foundation\Tomcat 6.0\lib\servlet-api.jar;c:\Archivos de programa\Apache Software Foundation\Tomcat 6.0\lib\jsp-api.jar; c:\Archivos de programa\Java\jdk1.6.0_01\jre\lib.`

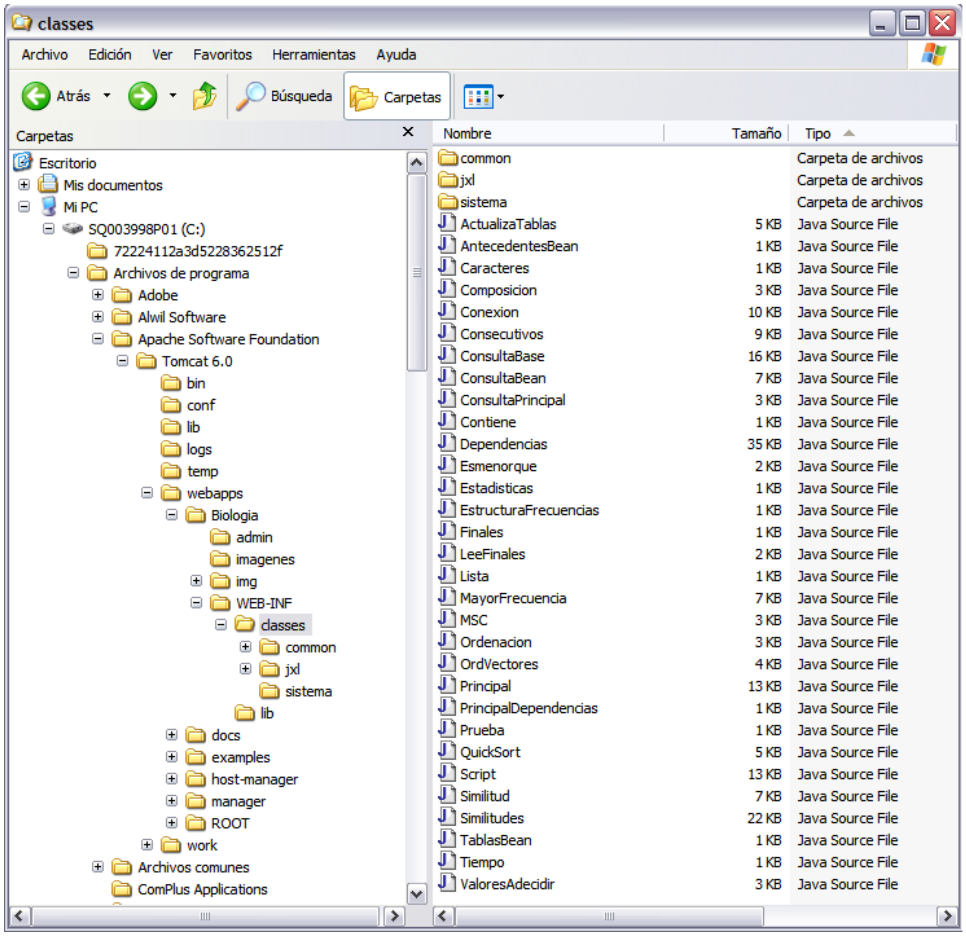
Finalmente, el manejador de base de datos PostgreSQL, el cual puede ser descargado desde la página `http://www.postgresql.org/`. La versión utilizada para este sistema es `PostgreSQL 8.2`.

Una vez instalado PostgreSQL el Script "ScriptINICIO.txt" debe ser ejecutado de lo contrario el sistema no tendrá un correcto funcionamiento. El contenido de este archivo puede ser visto en la figura B.7.



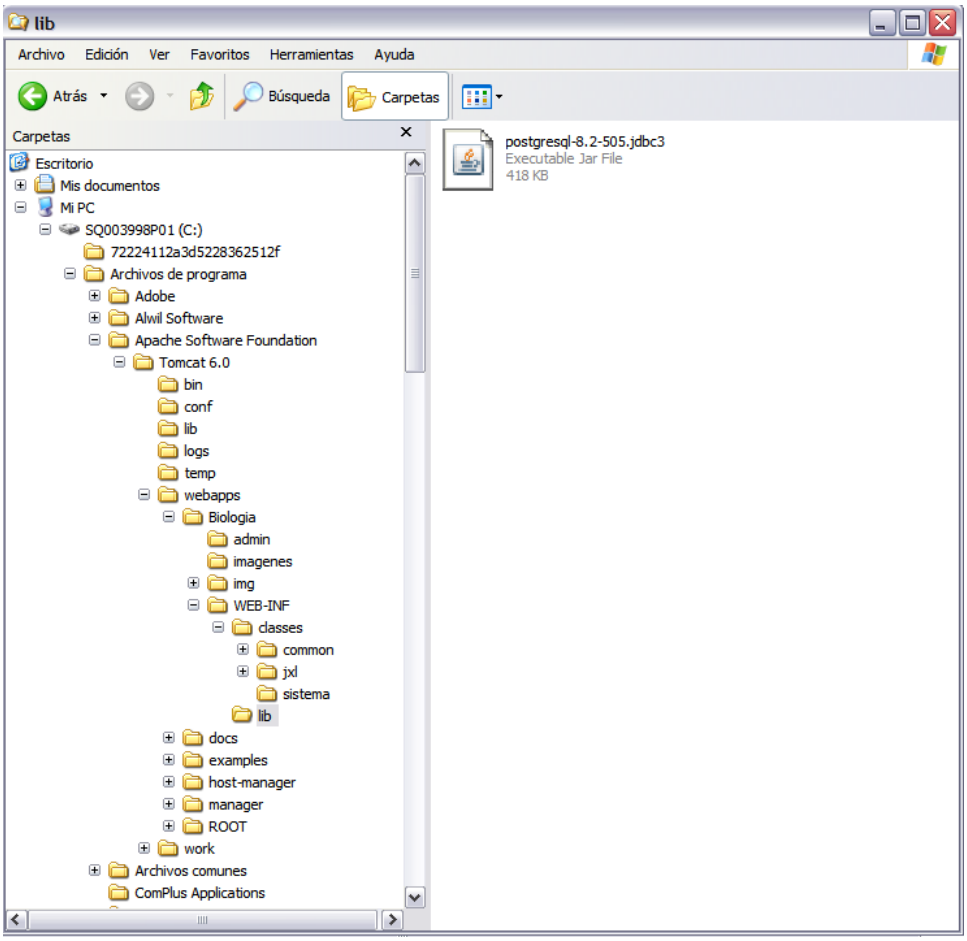
La figura muestra la estructura de directorios que debe llevar el sistema para su ejecución. También muestra el directorio principal llamado “Biologia” el cual contiene todos los programas realizados en el sistema. Como puede verse dentro de “Biologia” se encuentran todos los “JSPs” programados en el sistema.

Figura B.3: Estructura de directorios para el funcionamiento del sistema.



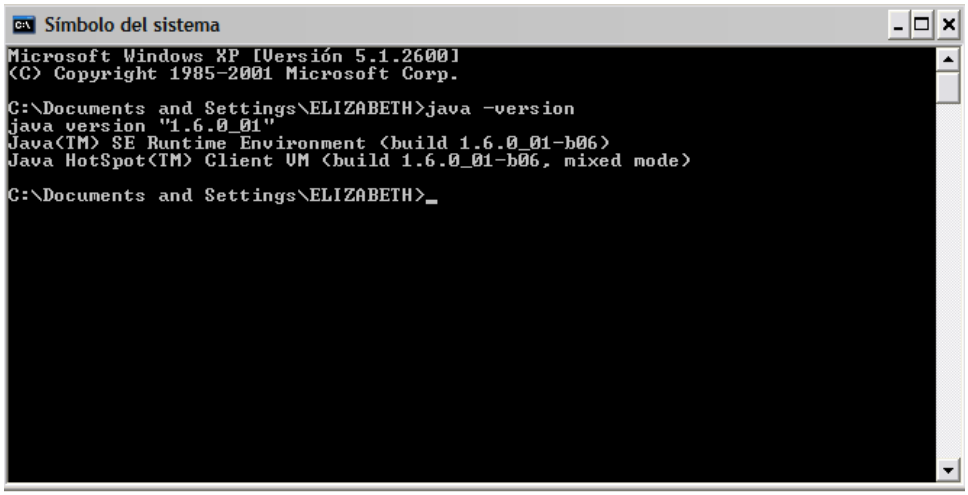
La figura muestra el contenido del directorio “classes” en el cual se almacenan todos los programas realizados en Java.

Figura B.4: Estructura de directorios para el funcionamiento del sistema.



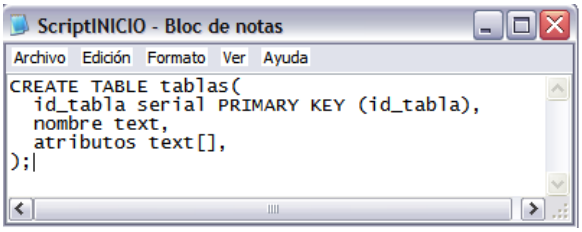
La figura muestra el directorio "lib" el cual contiene el JDBC de Posgres para comunicar la base de datos con el lenguaje Java.

Figura B.5: Directorio donde se almacena el JDBC de Posgres.



Prompt de WINDOWS donde se ejecuta la instrucción `java -version` una vez que se tiene intalado JAVA

Figura B.6: Versión del JDK instalado.



Script de incio para la ejecución del sistema una vez instalado POSTGRESQL.

Figura B.7: Contenido del "Script " de inicio.

Referencias

[1] V. Curcin, M. Ghanem, Y. Guo, M. Kohler, A. Rowe, J. Syed, and P. Wendel. Discovery net: Towards a grid of knowledge discovery. In *Proceedings of KDD-2002. The Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Canada, July 2002.

[2] A. Budanitsky and G. Hirst. Semantic distance in wordnet: An experimental, application-oriented evaluation of five measures. In *Workshop on WordNet and Other Lexical Resources, in the North American Chapter of the Association for Computational Linguistics (NAACL-2000)*, Pittsburgh, PA, June 2001.

[3] Francisco Couto, Mario Silva, and Pedro Coutinho. Improving information extraction through biological correlation. In Tobias Scheffer and Ulf Leser, editors, *Data Mining and Text Mining for Bioinformatics, Proceedings of the European Workshop Held in Conjunction with ECML/PKDD*, pages 8–13, Dubrovnik, Croatia, September 2003.

[4] S. Altschul, W. Gish, W. Miller, E. Myers, and D. Lipman. Basic local alignment search tool. *Journal of Molecular Biology*, pages 403–410, 1990.

[5] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1995.

[6] Ullman Jeffrey. *Principles of Database Systems*. Computer Science Press, USA, 1980.

[7] Abraham Silberschatz et all. *DataBase System Concepts*. WCB McGraw Hill, USA, 1999.

[8] C.J.Date. *Database In Depth, Relational Theory for Practitioners*. O'REILLY, USA, 2005.

[9] Frederick H. Lochovsky Dionysios C.Tsichritzis. *Data Base Management Systems*. ACADEMIC PRESS, USA, 1977.

[10] Kenmore S.Brathwaite. *Relational Database, Concepts, Design, and Administration*. McGraw Hill, USA, 1991.

[11] Jesús M.Milán-Franco. *Bases de datos y sistemas de información*. `http://www.fdi.ucm.es/profesor/milanjm/bdsi0304`, 2004.

[12] Nancy Greenberg and Instructor Guide PriyaÑathan. *Introduction to Oracle9i:SQL*. ORACLE, USA, 2001.

[13] Thomas Haigh. A veritable bucket of facts, origins of the data base management systems. In *SIGMOD Record, Vol.35 No.2*, Wisconsin, EUA, 2006.

[14] Mario Piattini y Oscar Díaz. *Advanced Database Technology and Design*. Artech House, Boston, London, 2000.

[15] ODBC3. `http://sestud.uv.es/manual.esp/gestion/gestion3.htm`. 2008.

[16] JDBC1. `http://java.sun.com/products/jdbc/faq.html`. 2008.

[17] Acharya Tinku Mitra Sushmita. *Data Mining, Multimedia, Soft Computing, and Bioinformatics*. Wiley-Interscience, USA, 2003.

[18] Micheline Kamber Jiawei Han. *Data Mining:Concepts and Techniques*. Morgan Kaufmann Publishers, USA, 2001.

[19] Eibe Frank Ian H.Witten. *Data Mining, Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers, USA, 2005.

[20] Dolf Zantinge Pieter Adriaans. *Data Mining*. Addison-Wesley, England, 1996.

[21] University of Massachusetts Elaine Johansen Mange y Arthur P. Mange. *Basic Human Genetics*. Sinauer Associates, Inc., Sunderland, Massachusetts, 1994.

[22] Bruce S. Weir. *Genetic Data Analysis II, Methods for Discrete Population Genetic Data*. Sinauer Associates, Inc.Publishers, Sunderland, Massachusetts, USA, 1996.

[23] GenBank. `http://www.psc.edu/general/software/packages/genbank/genbank.html`. 2006.

[24] Swiss-Prot. `http://ca.expasy.org/sprot/userman.html`. 2006.

[25] Medline. `http://www.seeiuc.com/forma/bases.htm`. 2006.

[26] Intefaz Medline. `http://medline.cos.com`. 2007.

[27] Fco. Javier Ceballos Sierra. *Java 2, Curso de Programación*. Alfaomega, España, 2000.

[28] PostgreSQL. `http://www.postgresql.org/about/`. 2007.

[29] Perdita Stevens y Rob Pooley. *Utilización de UML en Ingeniería del Software con Objetos y Componentes*. Addison Wesley, España, 2003.

[30] Guillermo Morales-Luna. *Análisis y diseño de algoritmos.* Sección de Computación, CINVESTAV-IPN, México, 2003.

[31] What is Unicode? `http://www.unicode.org/standard/WhatIsUnicode.html`. 2007.

[32] ¿Qué son los metadatos? `http://antares.inegi.gob.mx/metadatos/metadat1.htm`. 2008.