Proyecto de Sistemas Informáticos (curso 2006-2007)

Facultad de Informática
Universidad Complutense de Madrid

Desarrollo de un sistema para la integración de nomenclatura y campos textuales sobre genes y proteínas

Tutor: Mónica Chagoyen Quiles

Alumnos: Amaia Begoña Arce Abaitua
Ignacio Baena Moya
Ignacio Díaz Baeza

Se autoriza a la Universidad Complutense a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria, como el código, la documentación y/o el prototipo desarrollado.

Objetive:

Among the areas of principal activities' investigation in biomedic text analysing field we find the recognition of names and abbreviations used to talk about two main entitys: genes and proteins. Many of the proposed methods are based in control vocabularies builded up from information stored in several databases used in bioinformatic.
Designing and developing a system that integrates information about genes/proteins , susceptible of being used for the automatic recognise in science texts. This integration will be achieved by using the id numbers (foreign keys) from the main bioinformatics' data bases (UniProt, NCBI Entrez) building up a datawarehouse that includes the principal features of this genes/proteins. This system will be used by the tools developed in the Department of Architecture and Automatic Engineering.

Objetivo:

Una de las áreas de más actividad investigadora en el análisis de texto biomédico es el reconocimiento de los nombres y abreviaturas utilizadas para referirse a dos entidades biológicas de gran relevancia: genes y proteínas. Muchos de los métodos propuestos se basan en vocabularios control construidos a partir de la información almacenada en distintas bases de datos utilizadas en bioinformática.
Debido a la ambigüedad existente en la nomenclatura de genes y proteínas dentro del mundo de la bioinformática, el objetivo de este proyecto es el de diseñar y desarrollar un sistema que integre información normalizada sobre genes/proteínas, susceptible de ser utilizada para el reconocimiento automático de dicha nomenclatura en textos científicos. La normalización se realizará en base a los identificadores únicos (claves externas) de diversas bases de datos.
Para ello se desarrollará una base de datos centralizada (*datawarehouse*), sobre la que se incorporarán los datos pertinentes de varias bases de datos públicas (ej. UniProt, NCBI Entrez, Gene Ontology), así como los mecanismos de actualización necesarios.
Finalmente se proporcionará la funcionalidad para su acceso programático. Este sistema será consultado por las herramientas de análisis de texto biomédico desarrolladas en nuestro grupo de investigación.

Palabras clave:

- Bioinformática
- Datawarehouse
- ProteinUpload
- ProteinSeek
- Gen
- Proteína
- Base de datos
- SQL
- XML
- Integración de datos

INDICE

INTRODUCCIÓN

Las proteínas son referenciadas en textos utilizando sus nombres o sus símbolos que se utilizan como identificadores no ambiguos en bases de datos.
La identificación de proteínas sin ambigüedad es un paso fundamental para la extracción de la funcionalidad de estas entidades. Por desgracia esto es un proceso difícil, en parte por el uso complicado de los nombres de los genes y las proteínas.
Los genes y proteínas pueden ser referenciados en cualquier texto de diferentes formas: con su nombre completo, con su símbolo y también con variantes tipográficas. Muchos genes también tienen varios sinónimos, o el nombre del gen puede ser ambiguo y referir a palabras que también tienen diferentes significados dependiendo del contexto. Además, se comenta que hay errores en los nombres de genes que pudieron ser introducidos automáticamente por ciertas aplicaciones en bioinformática.

Proteínas y genes se caracterizan dentro de las bases de datos mediante identificadores únicos; cada identificador se asocia con su correspondiente proteína o secuencia de nucleótidos y descripción de su funcionalidad. El reconocimiento automático de entidades como genes y proteínas en textos es insuficiente si no está ligado a su identificador correspondiente de la base de datos. Distinguir entre el uso del nombre de la proteína y el nombre de la familia supone un gran obstáculo en la tarea de encontrar las entidades de las proteínas en el texto, ya que algunos párrafos del texto se referirán a las propiedades generales de las familias de las proteínas y en otros casos a las propiedades de las proteínas individuales.

Uno de los principales retos cuando se relacionan nombres de proteínas con las entradas de la base de datos es distinguir entre proteínas que tienen los mismos nombres pero pertenecen a diferentes genomas – un proceso que se denomina desambiguación entre especies de genes. Esto es especialmente voluminoso en el caso de los genes del ratón y del ser humano; el mismo símbolo de gen es a menudo utilizado en ambas especies y ambos nombres a menudo son mencionados en los mismos pasajes. La compleja naturaleza del nombre de las proteínas y de los genes se refuerza con la naturaleza dinámica del uso del nombre de los genes y la creación de los nombres, el nombre oficial de los genes deberá ser modificado y se crearán nuevos sinónimos. Esto explica por qué una aproximación estática o diccionarios no serán suficientes para resolver el problema.

Con nuestro proyecto, trataremos de ayudar a resolver este problema mediante un buscador de genes y proteínas que nos proporcionará detalles tan fundamentales como sinónimos, palabras asociadas y descripciones precisas de estos. Ayudado previamente por una aplicación encargada de normalizar diversas bases de datos públicas en una sola, de forma que las búsquedas serán más fáciles y rápidas.

1.-FUNDAMENTOS TEÓRICOS

1.1- BASE DE DATOS

Una base de datos es una colección de datos relacionados usada para representar la información. La base de datos es persistente.
Las bases de datos se utilizan para evitar los problemas que suponen el uso de ficheros. Estos problemas son:

- Control de redundancia e inconsistencia de datos.
- Costes de mantenimiento de programas:
    - Dificultad en el acceso. Cada consulta de datos implica generalmente escribir un nuevo programa.
    - Aislamiento de datos. Formatos diferentes y en medios diferentes.
    - Integridad. Para implementar restricciones de integridad es necesario modificar todos los programas que acceden a los datos.
- Atomicidad: Cuando hay un fallo informático (corte de corriente, error de disco, ...) se puede producir una inconsistencia en una transferencia bancaria. La transferencia debe ser una operación atómica (ocurre totalmente o no ocurre).
- Acceso concurrente: dos clientes de un banco quieren retiran fondos de la misma cuenta simultáneamente. Valor leído X, fondos retirados Y, Z. Uno escribe X-Y y el otro X-Z. Al final no queda el valor correcto X-Y-Z.
- Seguridad.

Las bases de datos están gestionadas por un Sistema Gestor de Bases de Datos
Características:

- Los SGBD permiten controlar la redundancia. A veces, por cuestiones de rendimiento, se pueden implementar datos redundantes.
- Seguridad: control de acceso a usuarios y grupos de usuarios a subconjuntos de la base de datos dependiendo de los permisos de cada uno de ellos.
- Representación de relaciones complejas entre datos.
- Imposición de restricciones de integridad.
- Almacenamiento persistente.
- Pueden ser muy grandes
- Las bases de datos son compartidas por usuarios y aplicaciones.
- Copias de seguridad y recuperaciones.
- Provisión de varias interfaces de usuario.
- Eficientes.
- Atomicidad: cuando hace un cambio lo hacen en todos los sitios o en ninguno para evitar errores.
- Control de concurrencia: sólo puede consultar un dato una persona cada vez.
- Acceso rápido: índices.
- Control de cambios

Un modelo de datos es una colección de conceptos que se usan para describir la estructura de una base de datos. La estructura son los tipos de datos las relaciones y las restricciones.
Un esquema es el diseño de una base de datos.
Una instancia son los valores concretos almacenados en una base de datos.

1.2.- SQL

SQL = Structured Query Language
El álgebra relacional es un lenguaje de manipulación de datos que permite modificar y consultar las bases de datos.
Operaciones:

- Entre conjuntos:
  - Unión.
  - Intersección.
  - Diferencia.
- Eliminan partes de una relación:
  - Proyección: elimina columnas.
  - Selección: elimina filas.
- Renombramiento.
- Combinaciones de tuplas:
  - Productos cartesianos.
  - Uniones naturales.
  - Productos.
  - Divisiones.
- Extensiones del álgebra relacional.

SQL es un lenguaje relacional estándar de facto multiconjunto del álgebra relacional.
Al trabajar con multiconjuntos se permite repetir valores. Las ventajas son:

- Eficiencia.
- El usuario puede querer los datos repetidos.
- Funciones de agrupación.

Tipos de dominios en SQL:

- char (n) : es una cadena de caracteres de longitud fija, con una longitud n especificada por el usuario.
- varchar (n): es una cadena de caracteres de longitud variable, con una longitud n especificada por el usuario.
- int: entero.
- smallint: es un subconjunto del dominio de los enteros.
- numeric (p,d): es un número en coma flotante, cuya precisión la especifica el usuario. El número está formado por p dígitos (más el signo) y de esos p dígitos, d pertenecen a la parte decimal
- real: números en coma flotante.
- double precision: números en coma flotante de doble precisión.
- flota (n): número en coma flotante, cuya precisión la especifica el usuario y es de al menos n dígitos.
- date: fecha. Contiene el año, mes y día.
- time: hora del día expresada en horas, minutos y segundos.
- timestamp: combina date y time.

El valor null pertenece a todos los dominios.
Comandos:

- Select.
- Insert.
- Delete.
- Update.

1.2.1.- Select

Utilizado para consultar registros de una base de datos que satisfagan un criterio determinado.
Recupera filas de la base de datos y permite seleccionar una o varias filas o columnas de una o varias tablas.
Esta función no modifica, inserta ni elimina ningún dato.
Sintaxis:

SELECT lista_selección
[FROM tabla_fuente]
[WHERE condición]
[GROUP BY expresión]
[HAVING condición]
[ORDER BY expresión [ASC | DESC]]

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desea seleccionar o manipular. Cláusulas:

- from: utilizada para especificar la tabla de la cual se van a seleccionar los registros.
- where: utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar.
- group by: utilizada para separar los registros seleccionados en grupos específicos.
- having: utilizada para expresar la condición que debe satisfacer cada grupo.
- order by: utilizada para ordenar los registros seleccionados de acuerdo con un orden específico.

La condición where puede ser:

- Comparación entre valores: =, >, >=, <=, <, <>.
- Similitud entre cadenas de caracteres: LIKE.
  - _ : representa un carácter cualquiera.
  - % : cualquier cantidad de caracteres.
- Comprobación de NULL.
- Conjunción (AND) y disyunción (OR) e condiciones.
- Negación de condiciones: NOT.
- EXISTS.

Las funciones de agregado se usan en cláusulas select en grupos de registros para devolver un único valor que se aplica a un grupo de registros. Estas funciones son:

- avg (expr): utilizada para calcular el promedio de los valores de un determinado campo.
- count (expr): utilizada para devolver el número de registros de la selección.
- sum (expr): utilizada para devolver la suma de todos los valores de un campo determinado.
- max (expr): utilizada para devolver el valor más alto de un campo especificado.
- min (expr): utilizada para devolver el valor más bajo de un campo especificado.

Estrategia de ejecución:

1. Genera la combinación de las tablas en la cláusula From.
2. Si existe una cláusula Where, se aplica la condición a las filas resultantes del paso 1 y se conserva sólo las filas que cumplen la condición.

3. Si no hay ninguna cláusula Group by se va al paso 7.
4. Si hay una cláusula Group by, se dividen las filas resultantes en el paso 2 en varios grupos, de modo que todas las filas de cada grupo tengan el mismo valor en todas las columnas de agrupamiento. Si no existe la cláusula Group by, todas las filas estarán en un único grupo.
5. A cada grupo se le aplica la cláusula Having si se ha especificado. Se conservarán los grupos que cumplan la condición.
6. Para cada grupo resultante se genera exactamente una fila de resultados evaluando la lista de selección de la cláusula select en dicho grupo.
7. Si la cláusula select contiene la palabra clave DISTINT, se eliminan las filas duplicadas.
8. Si hay una cláusula Order by, se ordena el resultado del paso 7 según la expresión de ordenación.

1.2.2.- Insert

Utilizado para cargar lotes de datos en la base de datos en una única operación.
Esta consulta puede ser de 2 tipos:

- Insertar un único registro:

  Sintaxis:

  INSERT INTO Tabla ($campo_1$, $campo_2$, ... , $campo_N$)
  VALUES ($valor_1$, $valor_2$, ... , $valor_N$)

  Esta consulta graba en el $campo_1$ el $valor_1$, en el $campo_2$ el $valor_2$ y así sucesivamente. Hay que prestar especial atención a acotar entre comillas simples los valores literales (cadenas de caracteres) y las fechas indicarlas en el formato mm-dd-aa.

- Insertar en una tabla los registros contenidos en otra tabla:

  Sintaxis:

  INSERT INTO Tabla ($campo_1$, $campo_2$, ... , $campo_N$)
  SELECT TablaOrigen.$campo_1$, ... , TablaOrigen.$campo_N$
  FROM TablaOrigen

  En este caso se seleccionarán los campos 1, 2, ... , n de la tabla origen y se grabarán en los campos 1, 2, ... , n de la Tabla.
  La condición select puede incluir la cláusula where para filtrar los registros a copiar.

1.2.3.- Delete

Utilizado para eliminar registros de una tabla de una base de datos.
Crea una consulta de eliminación que elimina los registros de una o más de las tablas listadas en la cláusula From que satisfagan la cláusula Where. Esta consulta elimina los registros completos, no es posible eliminar el contenido de algún campo en concreto.
Sintaxis:

DELETE
[FROM] tabla
[WHERE criterio]

Para eliminar valores en un campo especificado se ha de crear una consulta de actualización que cambie los valores a Null.
Si no se incluye cláusula Where, la instrucción Delete elimina todas las filas de la tabla.
Si se especifica una condición de búsqueda, ésta se aplica a todas las filas de la tabla. Todas las filas en las que el resultado de la condición de búsqueda es true se marca para su posterior eliminación. Todas las filas que están marcadas para eliminarse se deben eliminar al final de la instrucción delete antes de comprobar cualquier restricción de integridad.
En el caso de que infrinja una resticción foreing key, dicha instrucción se cancela, devuelve un error y no elimina ninguna fila.

1.2.4.- Update

Utilizado para modificar los valores de los campos y registros especificados.
Crea una consulta de actualización que cambia los valores de los campos de una tabla especificada basándose en un criterio específico.
Sintaxis:

UPDATE Tabla
SET $Campo_1 = Valor_1$, ... , $Campo_N = Valor_N$
WHERE Criterio

El valor de cada campo puede ser:

- Expresión: una variable, valor literal o expresión que devuelve un valor individual.
- Default: especifica que el valor predeterminado definido para la columna debe reemplazar al valor existente en dicha columna.
- Null: pone el valor a null en los casos que este valor sea permitido.

Si no se especifica cláusula where se actualizan todas las filas de la tabla.
La condición de búsqueda en la cláusula where se evalúa para cada fila de la tabla antes de actualizar una fila de la tabla.
Si una actualización ed la tabla infringe una restricción o una regla, vulnera la condición null de la columna o si el nuevo valor pertenece a un tipo de datos incompatible, la instrucción se cancela, se devuelve un error y no se actualiza ningún registro.

1.3.- MODELO ENTIDAD-RELACIÓN

El modelo entidad-relación se basa en los conceptos de: entidad, tipo de entidad, atributos y relaciones.
Esta información se representa en los diagramas entidad-relación.
Una entidad es el menor objeto con sentido completo en una instancia.
Se dice que un elemento de datos pertenece a una entidad (conjunto)
Los elementos de datos son únicos.
Clasificación de entidades:

- Débiles y fuertes: una entidad débil no tiene ninguna clave. Una entidad fuerte tiene una o más claves.
- Subentidades: todos los objetos de la entidad son a la vez objetos de otra entidad. Se dice que la primera es subentidad de la segunda entidad.

El tipo de entidad es el conjunto de entidades que comparten los mismos atributos (aunque con diferentes valores para ellos).
Las entidades se describen por un conjunto de atributos.
Cada atributo tiene asociado un dominio (conjunto de valores que puede tomar el atributo).
Tipos de atributos:

- Multivalorados o monovalorados: un atributo multivalorado es el que puede contener más de un valor simultáneamente. Un monovalorado sólo tiene un valor.
- Simples y compuestos: en un atributo compuesto se pueden distinguir varias componentes o atributos más pequeños. Y simples en otro caso.
- Clave: conjunto de atributos de una entidad que permiten distinguir entre sí a los objetos de la entidad. Clasificación:
    - Superclave: conjunto de atributos que permite distinguir a todas las entidades de cualquier instancia válida en la base de datos.
    - Clave candidata: superclave que no contiene ningún subconjunto que también sea superclave.
    - Clave primaria: clave candidata seleccionada por el diseñador para distinguir entre las entidades de cada instancia.

Los atributos pueden tener valor NULL, excepto aquellos que sean parte de la clave.
Una relación es un conjunto de la forma $\{(e_1, \dots , e_n) \mid e_1 \in E_1, e_2 \in E_2, \dots , e_n \in E_n \}$ con e_i entidades y E_i conjunto de entidades del mismo tipo.
La cardinalidad es el número de ocurrencias de cada objeto en la relación.
La cardinalidad entre dos entidades P y Q puede ser:

- Uno a uno: a cada instancia de P le corresponde una sola instancia de Q. A cada instancia de Q le corresponde una sola instancia de P.
- Uno a muchos: a cada instancia de P le corresponde varias instancias de Q. A cada instancia de Q le corresponde una sola instancia de P.
- Muchos a muchos: a cada instancia de P le corresponde varias instancias de Q. A cada instancia de Q le corresponde varias instancias de P.

Sea r una relación definida sobre los tipos de entidades $E_1,\dots,E_m$ y sea $E_j \in \{E_1,\dots,E_m\}$.
Se dice que la participación de la entidad $e \in E_j$ en r es n si $e \in E_j$ aparece en n tuplas de la relación.
Se dice que E_j tiene participación total en r si cada entidad $e \in E_j$ se encuentra en alguna tupla de r. En otro caso la participación es parcial.

1.4.- MODELO RELACIONAL

El modelo relacional es un modelo lógico de datos de no muy alto nivel orientado a registros.
El modelo relacional fue creado por Codd a principios de los 70.
Codd propuso que los sistemas de bases de datos deberían presentarse a los usuarios con una visión de los datos organizados en estructuras llamadas relaciones, definidas como un conjunto de tuplas (filas) y no como series o secuencias de objetos, con lo que el orden no es importante. Por tanto, detrás de una relación puede haber cualquier estructura de datos compleja que permita una respuesta rápida a una variedad de consultas.
El usuario de un sistema relacional sólo debe preocuparse por el qué consultar y no el cómo de las estructuras de almacenamiento (modelo físico).
La estructura fundamental del modelo relacional es la relación, es decir, una tabla bidimensional constituida por filas (tuplas) y columnas (atributos). Las relaciones representan las entidades que se consideran interesantes en la base de datos. Cada instancia de la entidad encontrará sitio en una tupla de la relación, mientras que los atributos de la relación representan las propiedades de la entidad.
Las tuplas en una relación son una colección no ordenada de elementos diferentes. Para distinguir una tupla de otra, se recurre al concepto de "clave primaria". La clave primaria es un atributo o conjunto de atributos que permiten identificar unívocamente una tupla en una relación. En una relación puede haber más de una combinación de atributos que identifique unívocamente una tupla. A estas relaciones las llamamos claves candidatas, pero entre éstas se elegirá una sola para utilizarla como clave primaria. Los atributos de la clave primaria no pueden asumir el valor nulo. A esta propiedad se conoce como integridad de las entidades.
Cada atributo de una relación se caracteriza por un nombre y por un dominio. El dominio indica qué valores pueden ser asumidos por una columna de la relación. Los dominios de una base de datos relacional deben ser atómicos, es decir, los valores contenidos en los atributos no se pueden separar en valores de dominios más simples. En resumen, no se admiten valores multivalorados ni compuestos.
Las tablas se normalizan para evitar la repetición innecesaria de datos (redundancia). La normalización ahorra espacio de almacenamiento, optimiza el rendimiento y, al eliminar la redundancia, impide modificaciones parciales o incompletas que podrían dar lugar a inconsistencias.
Hay 4 posibles formas normales, cada una más exigente que la anterior:

- 1ª Forma Normal: se cumple cuando no hay atributos multivalorados. Si existen este tipo de atributos, se pueden eliminar de 2 maneras:
  - Crear una tabla auxiliar para el atributo incluyendo la clave de la tabla origen.
  - Si se conoce la cardinalidad del atributo multivalorado, se puede descomponer en varios atributos.
- 2ª Forma Normal: una relación con clave primaria X está en 2ªFN si la dependencia funcional $X \rightarrow A$ es completa para todo atributo A o es trivial. Ningún atributo puede depender de parte de la clave. Todos los atributos deben depender de toda la clave.
  Una dependencia funcional ($X \rightarrow Y$) es completa cuando no se cumple $X-\{A_i\} \rightarrow Y$ para ningún $A_i \in X$ (no le sobra atributos en la parte izquierda)
- 3ª Forma Normal: Un esquema está en 3ªFN si cumple:
  - Está en 2ªFN

- Si S es el conjunto de dependencias funcionales (DF) del esquema, se tiene que para toda DF $X \rightarrow Y \in S^+$ no trivial, se tiene que X es superclave o Y es parte de una clave candidata (tenemos una DF transitiva).
  S^+ es el cierre del conjunto de dependencias funcionales S.

- 4ª Forma Normal (Forma Normal de Boyce-Codd): Un esquema está en 4ªFN con respecto a un conjunto de DF S si para toda $X \rightarrow Y \in S^+$ no trivial es superclave.

Durante el proceso de normalización surgen nuevas tablas y se eliminan algunas redundantes.

1.5.- PASO DEL MODELO ENTIDAD-RELACIÓN AL MODELO RELACIONAL

Cada entidad se transforma en una tabla con sus mismos atributos. Excepciones:

- Atributos compuestos → Posibles soluciones:
  - Unir todos los atributos compuestos en un solo atributo.
  - Considerar sólo los atributos no compuestos.
- Atributos multivalorados: se transforma en una entidad nueva y una relación entre la entidad nueva y la que la tenía como atributo.
- Entidades débiles: se pueden transformar si están relacionadas con una entidad fuerte con participación exactamente uno. En este caso se añaden a los atributos de la entidad débil los de la clave de la entidad fuerte relacionada. La clave son todos los atributos.

En las relaciones las restricciones de participación se pierden. Cada relación se corresponderá con una tabla cuyos atributos son los de la clave de las entidades relacionadas más los atributos de la relación. La clave será:

- Si todas las entidades participan una cantidad diferente de uno en la relación, la clave en el modelo relacional será la unión de las claves de las entidades que relaciona.
- Si alguno de los tipos de entidad tiene participación exactamente uno, la clave de ese tipo de entidad será la clave de la relación en el modelo relacional.

En ocasiones es necesario renombrar atributos para evitar tener varios atributos con el mismo nombre.

En ocasiones es posible combinar 2 o más tablas en una sola o eliminar tablas que contengan información redundante.

1.6.- RESTRICCIONES DE INTEGRIDAD

1.6.1.- Restricciones de dominio

- Tipo: al dar valores a una columna, restringimos su conjunto de valores válidos a los valores del tipo
- Check: condiciones que debe cumplir una columna
- Restricciones de existencia (NOT NULL): se usa para indicar que un atributo siempre debe tomar un valor.
- Restricciones de unicidad: una columna o columnas cuyos valores no pueden repetirse (claves candidatas). Las claves afectadas deben declararse NOT NULL.

1.6.2.- Integridad referencial

Sean R_1, R_2, dos relaciones. Sea k_1 la clave primaria de R_1. Se dice que un conjunto de atributos A de R_2 es una clave externa de R_2 con respecto a R_1 si para todo par de instancias válidas $r_1 \in R_1$, $r_2 \in R_2$ se cumple que para toda tupla $t \in r_2$, existe una tupla $s \in r_1$ tales que $s[k_1]=t[s]$. Decimos en este caso que tenemos una restricción de integridad referencial de R_2 con respecto a R_1.

1.6.3.- Aserciones

Son predicados que indican condiciones que debe cumplir la instancia de la base de datos en todo momento. Las restricciones de dominio y las restricciones de integridad referencial son casos particulares.
Las aserciones no se indican al crear la tabla sino después de creada la instancia de la base de datos. En el momento de crear la aserción se comprueba si la instancia actual verifica la aserción. A partir de ese momento la aserción queda activa. Si la instancia no la verifica, la aserción no se crea.
Las aserciones se pueden eliminar.
El principal problema de las aserciones es que tienen un gran coste en tiempo para el sistema.

1.6.4.- Disparadores

Es una orden que el sistema ejecuta de manera automática cuando se produce algún tipo de modificación de la instancia de la base de datos. Los disparadores se utilizan para evitar que una instancia llegue a ser no válida y para realizar acciones de forma automática.

1.6.5.- Dependencias funcionales

Una dependencia funcional X→Y indica que un conjunto de atributos Y depende de otro conjunto de atributos X, en el sentido de que una repetición de valores para X implica una repetición de valores para Y.
Sea R una tabla relacional con atributos $\{A_1, \ldots, A_n\}$. Una dependencia X→Y se cumple en una instancia r ϵ R si para toda t, s ϵ r, t[x]=s[x] => t[y]=s[y] (si dos tuplas coinciden en x deben coincidir en y).
La relación "→" cumple las siguientes propiedades:

- Reflexiva: sí. Se cumple siempre que X→X.
- Simétrica: no. X→Y no implica que Y→X.
- Antisimétrica: no. Es posible tener X→Y, Y→X con $X \neq Y$.
- Transitiva: si. Si X→Y, Y→Z entonces X→Z.

Las dependencias triviales son aquellas que se van a cumplir siempre, independientemente de la instancia. Son de la forma X→Y con $Y \subseteq X$.
A partir de un conjunto de DF se pueden definir nuevas DF.
Se llama cierre de un conjunto de dependencias funcionales S, al conjunto de todas las dependencias funcionales S que se deducen del conjunto S. A este nuevo conjunto se le denota S^+ y se calcula mediante los axiomas de Armstrong. Estos axiomas son:

1. Reflexividad: si $Y \subseteq X$ entonces se cumple que X→Y
2. Aumentatividad: si X→Y entonces XZ→YZ donde XZ es la unión de X y Z
3. Transitividad: si X→Y, Y→Z entonces X→Z
4. Autodeterminación: X→X
5. Unión: si X→Y, Y→Z entonces X→YZ
6. Descomposición: si X→YZ entonces X→Y y X→Z
7. Composición: si X→Y, Z→W entonces XZ→YW
8. Pseudotransitividad: si X→Y, YZ→W entonces XZ→W

Para saber si X→Y está en S^+, tenemos 2 métodos:

- Utilizar las reglas del 1 al 8 para deducir X→Y de S
- Calcular explícitamente S^+ como punto fijo de los axiomas 1, 2 y 3 y ver si X→Y ϵ S^+. El inconveniente de este método es que el conjunto S^+ puede ser muy grande.

Dado un conjunto de atributos X, se llama cierre de X con respecto a S (S conjunto de DF) al conjunto X^+ que contiene todos los atributos de A tales que X→{A} se puede deducir de S.
Dada una relación R con un conjunto de dependencias funcionales S. Llamamos recubrimiento mínimo de S al conjunto de dependencias funcionales equivalente y mínimo (no le sobra ninguna DF).

1.6.6.- Dependencias multivaloradas

Sea R el conjunto de atributos de una relación. Decimos que la dependencia multivalorada X→→Y se cumple en una instancia r de la relación cuando dadas dos tuplas t_1, t_2 ϵ r tales que $t_1[x]=t_2[x]$, se tiene que existen otras dos tuplas t_3, t_u ϵ r verificando:

1. $t_1[x]=t_2[x]=t_3[x]=t_4[x]$
2. $t_3[y]=t_1[y]$ y $t_3[z]=t_2[z]$
3. $t_4[y]=t_2[y]$ y $t_4[z]=t_1[z]$

donde z = r\x\y

1.7.- DISEÑO DE BASES DE DATOS RELACIONALES

Etapas en la construcción de una aplicación de bases de datos:

1. Requisitos o requerimientos: en lenguaje natural.
2. Esquema conceptual: descrito con gran precisión. Es el diagrama entidad-relación y las reglas de negocio escritas en lenguaje natural.
3. Esquema lógico: tablas relacionales y restricciones.
4. Esquema físico: ficheros, índices, parámetros.

El paso de la etapa 1 a la 2 se llama diseño conceptual. El de la 2 a la 3, diseño lógico. Y de la 3 a la 4, diseño físico.

1.7.1.- Requisitos

Descripción en lenguaje natural del problema. Fases:

1. Recoger información del usuario/cliente.
2. Identificar conceptos, sinónimos y posibles reglas de negocio.
    a. Concepto = entidades y relaciones que se pueden extraer de los requisitos
    b. Sinónimos = conceptos equivalentes. Se eliminan hasta que no quedan sinónimos.
    c. Reglas de negocio = restricciones que deben cumplir las instancias válidas de una base de datos. Tipos:
        i. Definición de relaciones entre conceptos
        ii. Restricciones de integridad
        iii. Restricciones de eficiencia
3. Añadir información sobre operaciones de la base de datos y frecuencia de las operaciones.
4. Especificación en lenguaje natural mejorado. Se le enseña al cliente y se vuelve a la fase 1 si es necesario.

1.7.2.- Esquema conceptual

Etapas en el diseño del esquema conceptual:

1. Se parte de un análisis de requisitos:
    a. Glosario de términos (diccionario del problema).
    b. Requisitos agrupados según funcionalidad.
2. Paso básico: diagramas con super-entidades.
3. Paso de descomposición: refinar las super-entidades convirtiéndolas en entidades más pequeñas.
4. Análisis de calidad: comprobar si se cumplen los requerimientos. En caso de no cumplirse volver al paso 2.

1.7.3.- Esquema lógico

Etapas en el diseño del esquema lógico:

1. Se obtienen tablas relacionales a partir del diagrama Entidad-Relación.
2. Se determinan las dependencias funcionales.
3. Se determinan las superclaves, las claves candidatas y se escoge una clave primaria por tabla.
4. Se normalizan las tablas.

2.- DISEÑO E IMPLEMENTACIÓN

2.1.-BASES DE DATOS PÚBLICAS

Descarga de las Bases de Datos

1. *Homo Sapiens y Mus Musculus*

Dirección Web: www.uniprot.org

A la izquierda, hacemos clic en "Power Search" y nos aparecerá una ventana en la que tendremos que ir seleccionando las opciones que queremos por el siguiente orden:

Select a library UniProtKB/SwissProt Only

Query line type Species / Organism or Host

Line type details Any Name

Select an operador Exact Match

Enter the query text Homo Sapiens

----------→ Search & View

Nos aparecerá otra ventana de “Result Entry Set Gris View”
En la que haremos clic en “Download Set”
Seleccionaremos

Select a download format XML

Select number of files One file for all proteins

----------→ Add all

Es posible que a la hora de bajarnos el fichero XML, lo tenga que fraccionar en varios ficheros distintos, que posterior mente cargaremos en nuestra base de datos uno a uno.

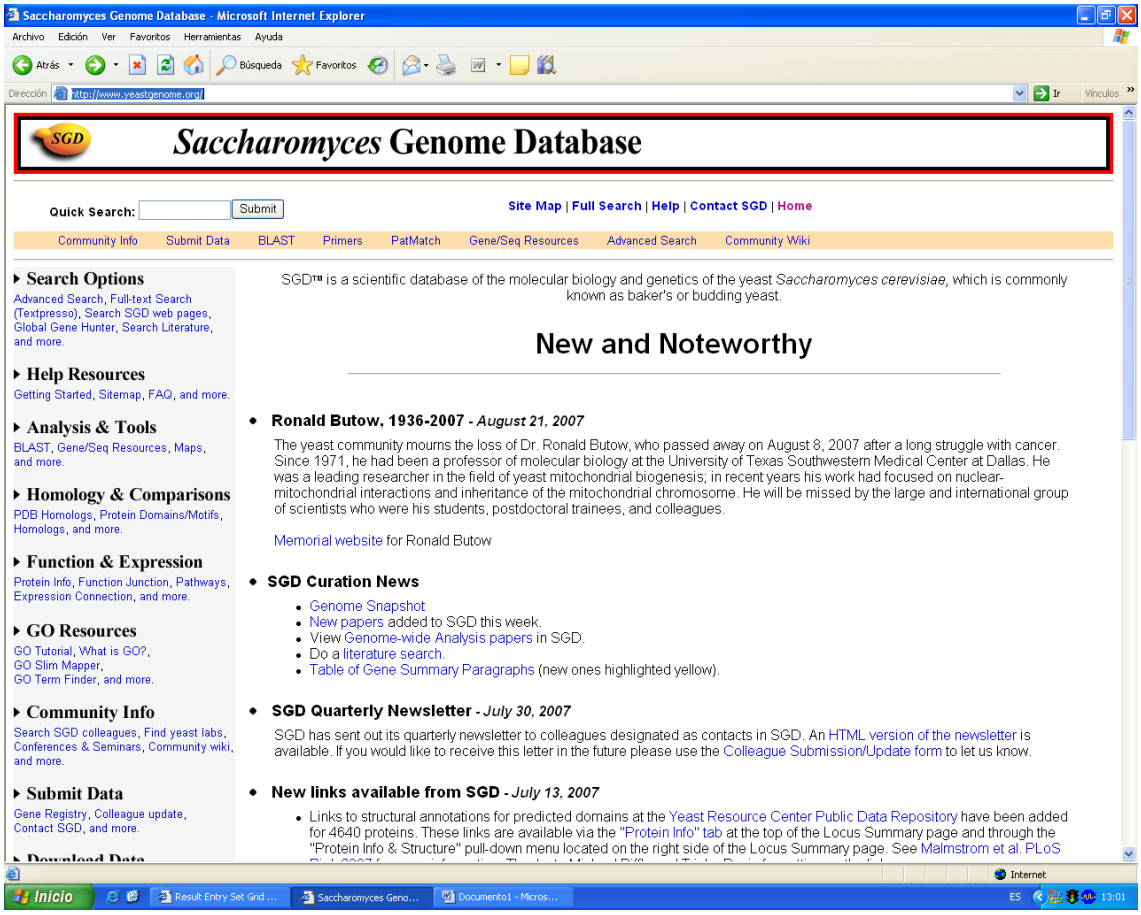
Se repetirá el mismo proceso para el ratón cambiando

Enter the query text Mus Musculus

Ambas, se deberán de descargar en la carpeta que posteriormente utilizaremos para aunar las bases, que en nuestro caso ser **ProteinUpload.**

2. *Saccharomyces*

Dirección Web: www.yeastgenome.org



A la izquierda, haremos clic en el apartado de Download Data en "FTP" que nos enlazará con el

"Índice de ftp://genome-ftp.stanford.edu/pub/yeast/"

En el que hemos de seleccionar el apartado

"gene registry"

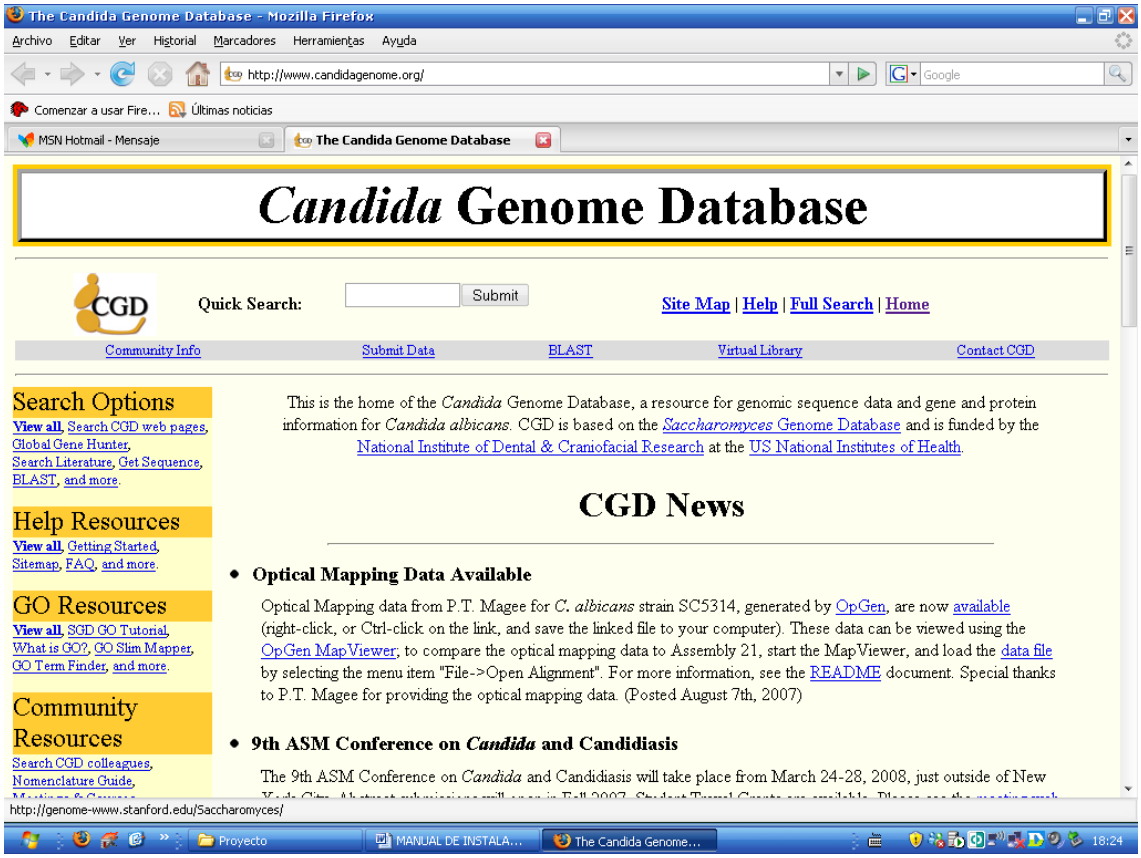
Acto seguido, hemos de hacer clic en el apartado de la nueva página

"registry.genenames.tab"

Y guardar dicho fichero en la carpeta que tenemos asignada para el proyecto.

3. *Candida Albicans*

Dirección Web: www.candidagenome.org



A la izquierda, haremos clic en el apartado de Download Data en "Chromosomal Feature File" que nos enlazará con el archivo que contiene todos los datos.

En este caso, no queremos todas las características de dicha especie, serán solamente seis las que hemos de guardar. Por el contrario, lo que haremos será guardar el archivo completo, y luego nuestra base de datos se encargará de seleccionar los seis apartados que tenemos. Mediante los "Servicios de transformación" de datos que posee el programa, haremos una selección de las características que deseamos formen parte de nuestra base de datos.

Por supuesto, este archivo se descargará en la carpeta del proyecto.

Una vez descargadas las bases de datos, el programa se encargará de cargarlas, creando nuevamente las tablas y procesos.
Todas ellas se han de guardar con los nombres que las propias páginas asignan a los archivos descargados, esto evitará problemas de renombramiento en las búsquedas a la hora de ejecutar la carga del programa.

2.2.- HOMO SAPIENS Y MUS MUSCULUS

2.2.1.- Requisitos

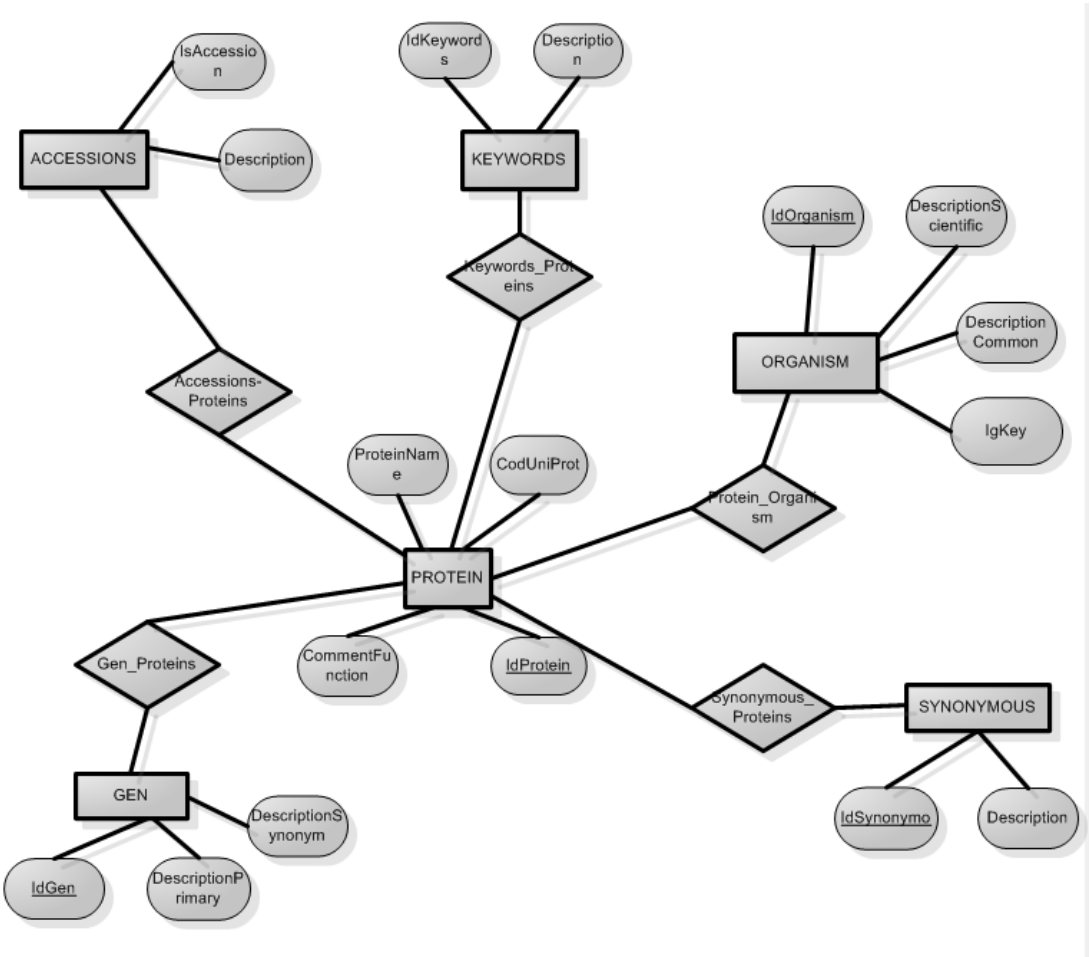
La información que queremos guardar sobre los organismos Homo Sapiens y Mus Musculus es:

- Nombre proteína.
- Nombre gen: descripción primaria y secundaria.
- Sinónimos.
- Número de acceso en la tabla de Uniprot.
- Función.
- Keywords.
- A qué organismo pertenecen.

Es posible que una proteína tenga más de un sinónimo, más de un número de acceso, más de una clave y puede pertenecer a más de un gen.

2.2.2.- Esquema conceptual

Diagrama Entidad-Relación:



2.2.3.- Esquema lógico

A partir del diagrama Entidad-Relación obtenemos las siguientes tablas:

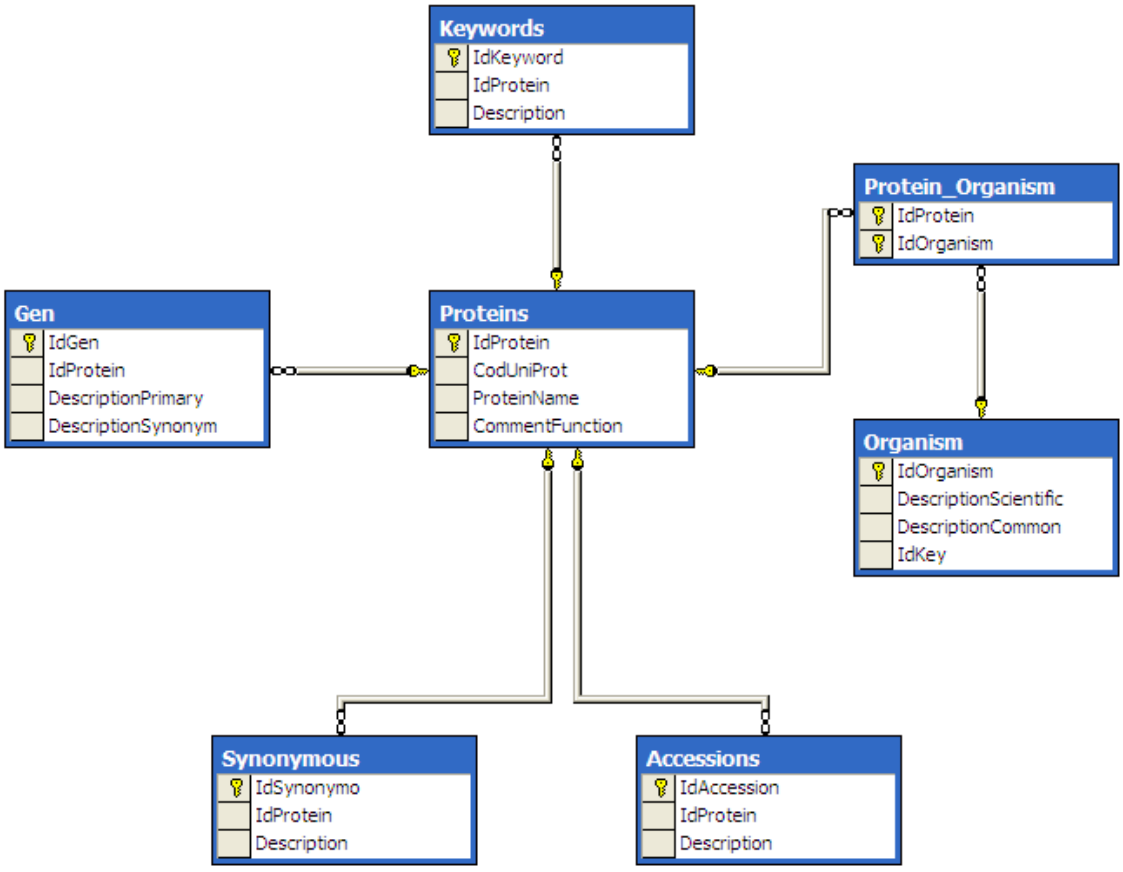
- Proteins (IdProtein, CodUniProt, ProteinName, CommentFunction).
- Protein_Keywords (IdProtein, IdKeywords).
- Keywords (IdKeyword, Description).
- Protein _Gen (IdProtein, IdGen).
- Gen (IdGen, DescriptionPrimary, DescriptionSynonym).
- Protein _Synonymous (IdProtein, IdSynonymo).
- Synonymous (IdSynonymo, Description).
- Protein _Accesions (IdProtein, IdAccession).
- Accessions (IdAccession, Description).
- Protein_Organism (IdProtein, IdOrganism).
- Organism (idOrganism, DescriptionScientific, DescriptionCommon, IdKey).

Se puede simplificar y obtenemos las siguientes tablas:

- Proteins (IdProtein, CodUniProt, ProteinName, CommentFunction).
- Keywords (IdKeyword, IdProtein, Description).
- Gen (IdGen, IdProtein, DescriptionPrimary, DescriptionSynonym).
- Synonymous (IdSynonymo, IdProtein, Description).
- Accessions (IdAccession, IdProtein, Description).
- Protein_Organism (IdProtein, IdOrganism).
- Organism (idOrganism, DescriptionScientific, DescriptionCommon, IdKey).

Dependencias funcionales: sólo tenemos dependencias triviales.

Diagrama:


Keywords
IdKeyword
IdProtein
Description
Protein_Organism
IdProtein
IdOrganism
Gen
IdGen
IdProtein
DescriptionPrimary
DescriptionSynonym
Proteins
IdProtein
CodUniProt
ProteinName
CommentFunction
Organism
IdOrganism
DescriptionScientific
DescriptionCommon
IdKey
Synonymous
IdSynonymo
IdProtein
Description
Accessions
IdAccession
IdProtein
Description

2.3.- SACCHAROMYCES

2.3.1.- Requisitos

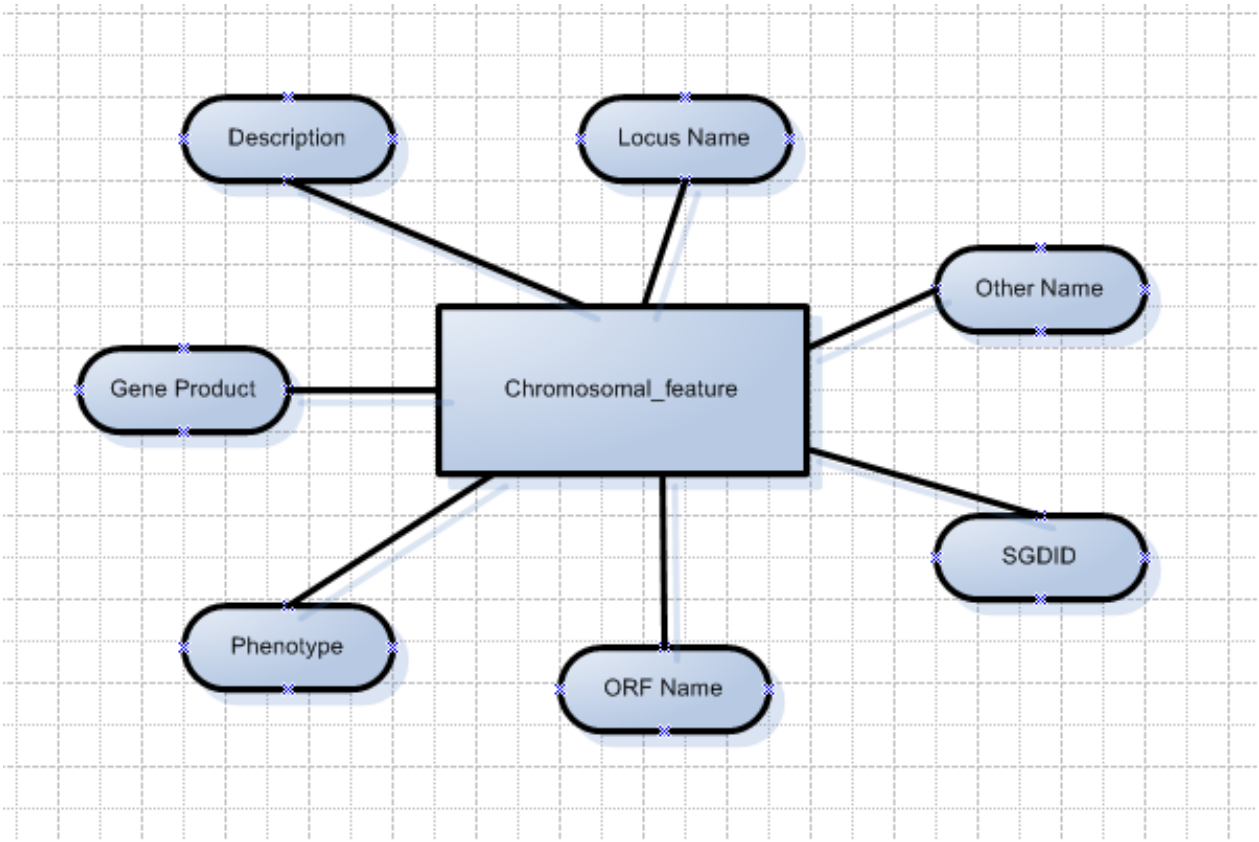
La información que queremos guardar sobre este organismo es:

1. Locus name.
2. Otros nombres (sinónimos).
3. Descripción.
4. Productos genéticos.
5. Fenotipo.
6. Nombre ORF.
7. SGDID.

Es posible que una proteína tenga más de un sinónimo, más de un producto genético y más de un fenotipo.
La descripción, los productos genéticos y el fenotipo pueden tomar el valor NULL.

2.3.2.- Esquema conceptual

Diagrama Entidad-Relación:



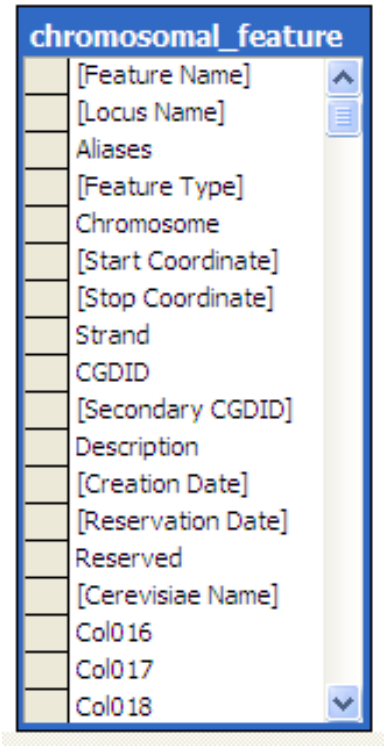
2.3.3.- Esquema lógico

A partir del diagrama Entidad-Relación obtenemos las siguiente tabla:

Chromosomal_feature (Locus Name, Other Name, Description, Gene Product, Phenotype, ORF Name, SDID)

Dependencias funcionales: sólo tenemos dependencias triviales.

Diagrama:



2.4.- CANDIDA

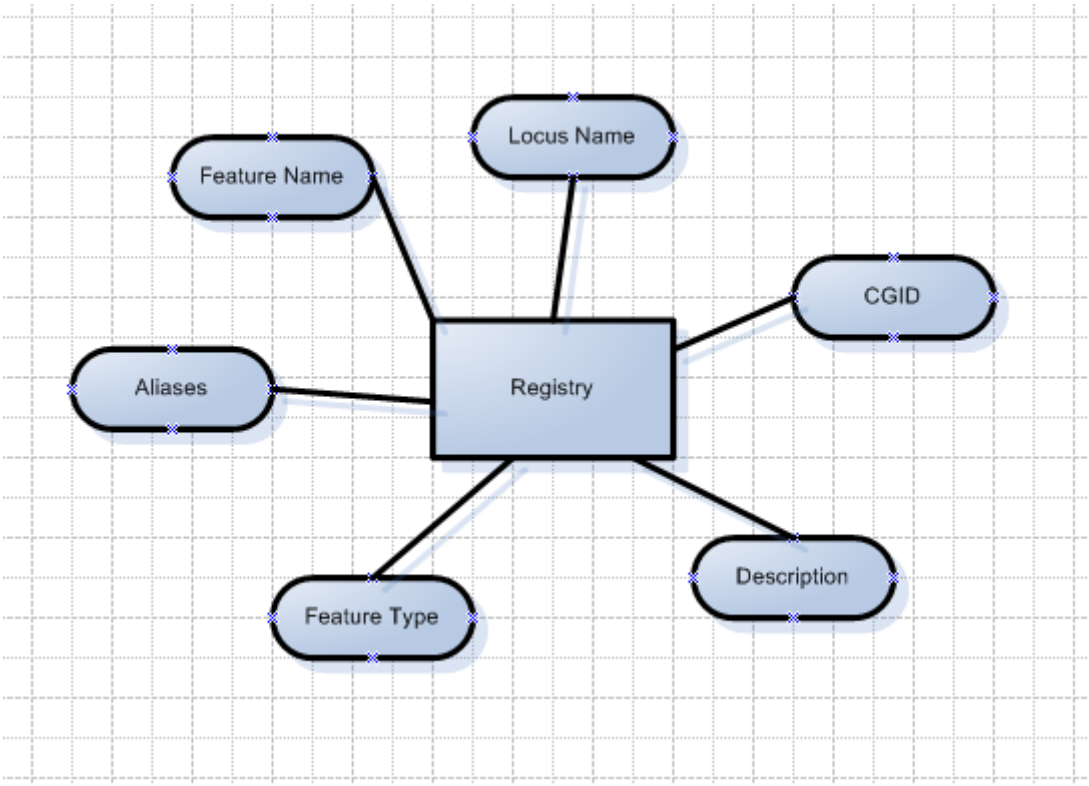
2.4.1.- Requisitos

La información que queremos guardar sobre este organismo es:

1. Feature name.
2. Locus name.
3. Aliases.
4. Feature type.
5. Descripción.
6. CGID

2.4.2.- Esquema conceptual

Diagrama Entidad-Relación:



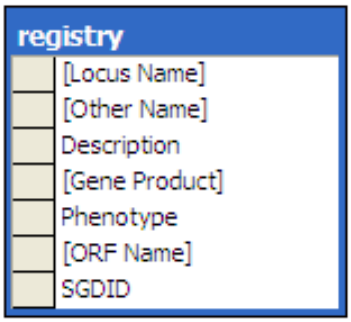
2.4.3.- Esquema lógico

A partir del diagrama Entidad-Relación obtenemos la siguiente tabla:

Registry (Feature Name, Locus Name, Aliases, Feature Type, Description, CGID)

Dependencias funcionales: sólo tenemos dependencias triviales.

Diagrama:


registry
[Locus Name]
[Other Name]
Description
[Gene Product]
Phenotype
[ORF Name]
SGDID

3.- PROCESADO E INTEGRACIÓN DE LA INFORMACIÓN

3.1.- CARGA DE DOCUMENTOS XML

Una vez tenemos nuestros documentos XML descargados de las bases de datos públicas de Uniprot con toda la información referente al ratón y el genoma humano nos disponemos a procesar la información de dichos documentos para cargar nuestra base de datos con toda la información que nos interesa. Para ello, iniciamos la carga, habiendo seleccionado previamente un fichero con toda la información que deseamos cargar. Tanto la selección del fichero como la operación de carga se realizará a través de un sencillo interfaz. En el caso de que al iniciar una carga no se haya seleccionado antes algún fichero, nuestra aplicación mostrará un mensaje de error indicándonos que debemos seleccionar algún fichero.

Una vez seleccionado el fichero que deseamos cargar, podemos proceder a ello. Veamos cómo hemos realizado el procesado de nuestros documentos. En primer lugar creamos un objeto XML. Con este objeto cargamos el fichero seleccionado con la ruta donde se encuentra. Una vez cargado, mediante este objeto podemos comenzar a tratar nuestro documento XML.

Seleccionamos los nodos "entry" que corresponde a cada una de las proteínas. Procesamos cada una de ellas. Antes de nada comprobamos que la proteína tenga, al menos un código. Extraemos su código. A continuación, extraemos el nombre de la proteína así como la función comment. Con su código, su nombre y la función cargamos la proteína en cuestión comprobando previamente que la proteína no existe ya. La información anterior aparece en el documento XML de la siguiente manera (se indica en negrita la información que nos interesa con los nodos para su localización):

```
<entry dataset="Swiss-Prot" created="1986-07-21" modified="2007-07-24"
version="73">
  <accession>P05408</accession>
  <accession>P01164</accession>
  <accession>Q9BS38</accession>
  <name>7B2_HUMAN</name>
  <protein>
    <name>Neuroendocrine protein 7B2 precursor</name>
    <name>Secretogranin-5</name>
    <name>Secretogranin V</name>
    <name>Secretory granule endocrine protein I</name>
    <name>Pituitary polypeptide</name>
    <component>
      <name>N-terminal peptide</name>
    </component>
      ...
<comment type="function">
    <text>Acts as a molecular chaperone for PCSK2/PC2, preventing its
premature activation in the regulated secretory pathway. Binds to
inactive PCSK2 in the endoplasmic reticulum and facilitates its
transport from there to later compartments of the secretory pathway
where it is proteolytically matured and activated. Also required for
```

```
cleavage of PCSK2 but does not appear to be involved in its folding.
Plays a role in regulating pituitary hormone secretion. The C-terminal
peptide inhibits PCSK2 in vitro</text>
  </comment>
...
<location>
      <begin position="186"/>
      <end position="194"/>
    </location>
  </feature>
  <sequence length="197" mass="22552" checksum="82B25F56D382CE57"
modified="2002-08-30" version="2">
MYWSNQITRRLGERVQGFMSGISPQQMGEPEGSWSGKNPGTMGASRLYTL
VLVLQPQRVLLGMKKRGFGAGRWNGFGGKVQEGETIEDGARRELQEESGL
TVDALHKVGQIVFEFVGEPELMDMHVFCTDSIQGTPVESDEMRPCWFQLD
QIPFKDMWPDDSYWFPLLLQKKKFHGYFKFQGQDTILDYTLREVDTV
</sequence>
</entry>
```

Para comprobar que la proteína no existe ya, nos basamos en su código. Creamos un conjunto de datos para la proteína. En dicha estructura se almacenan las proteínas de nuestra base de datos (si las hubiera) con el código de la proteína en cuestión que estamos comprobando. Por tanto, tendremos que acceder a nuestra base de datos haciendo uso además de un procedimiento almacenado en SQL Server creado por nosotros. El procedimiento en cuestión es `"dbo.P_S_Proteina"` que no hace más que seleccionar todos los campos de nuestra tabla Proteins cuyo campo código coincida con el de la proteína que estemos tratando. Previamente establecemos la conexión y la abrimos, indicando además que vamos a ejecutar un procedimiento almacenado y añadiendo los parámetros del procedimiento almacenado, en este caso el código. Rellenamos nuestra estructura creada con la información que devuelve el procedimiento. Por último, desconectamos las conexiones para que no queden colgadas. Una vez realizado todo esto, si nuestra estructura está vacía, podremos estar seguro que la proteína no se encontraba en la base de datos, pasando a continuación a insertarla.

Insertamos la proteína con el código, nombre y función. Es decir, rellenamos la tabla Proteins con todos sus campos, IdProtein (nuestro identificador de la tabla), CodUniprot (el código con el que comprobamos que la proteína no se encontraba ya insertada), ProteinName (contiene el nombre de la proteína) y, por último, CommentFunction. También buscaremos el campo correspondiente al organismo de esa proteina para configurar nuestra tabla Protein_Organism, la cual, asocia cada proteina con su organismo. Para crear ambas tablas, hacemos uso de uno de los procedimientos almacenados: `"dbo.P_I_Proteina".` Antes de nada establecemos la conexión y la abrimos indicando que vamos a ejecutar un procedimiento almacenado. Ejecutamos y recuperamos el registro id del registro recién insertado. Por último, desconectamos las conexiones.

Previo a la inserción de la proteina con el identificador del organismo asociado (si lo hubiera), insertamos el organismo de la proteína (si lo tuviese) en su tabla y con sus valores correspondientes que necesitemos. Veamos el formato en XML de un organismo en una proteina:

```
      ...
```

```
<organism key="1">
    <name type="scientific">Homo sapiens</name>
    <name type="common">Human</name>
    <dbReference type="NCBI Taxonomy" id="9606" key="2"/>
    <lineage>
      <taxon>Eukaryota</taxon>
      <taxon>Metazoa</taxon>
      <taxon>Chordata</taxon>
      <taxon>Craniata</taxon>
      <taxon>Vertebrata</taxon>
      <taxon>Euteleostomi</taxon>
      <taxon>Mammalia</taxon>
      <taxon>Eutheria</taxon>
      <taxon>Euarchontoglires</taxon>
      <taxon>Primates</taxon>
      <taxon>Haplorrhini</taxon>
      <taxon>Catarrhini</taxon>
      <taxon>Hominidae</taxon>
      <taxon>Homo</taxon>
    </lineage>
  </organism>
      ...
```

La información que necesitamos de todo lo anterior para configurar nuestra tabla Organism será `"organism/name[@type = 'scientific']"` (correspondiente al campo DescriptionScientific en nuestra tabla) y `"organism/name[@type = 'common']"` (correspondiente al campo DescriptionCommon en la tabla).

Para hacer la inserción de los valores seleccionados procedemos de igual forma que a la hora de insertar valores en otras tablas. Aquí el procedimiento almacenado a utilizar será: `"dbo.P_I_Organism"`. Para la inserción primero establecemos la conexión y la abrimos, indicamos que vamos a ejecutar un procedimiento almacenado y añadimos los parámetros. Ejecutamos y recuperamos el id del registro recién insertado. Por último, desconectamos las conexiones.

Al igual que hicimos para identificar la información correspondiente a cada una de las proteínas mediante los nodos entry, para cada una de las proteínas identificamos los genes asociados a la proteína que se esté procesando en ese momento. Lo identificamos a través de los nodos "gene". Cada proteína puede tener un número diferente de genes asociados, tal y como podemos observar viendo la relación de tablas de nuestra base de datos. Si tiene más de uno, los procesamos. En el caso de que tuviese synonym, lo insertamos con el gen. Veamos el formato en el que aparece dentro del documento XML un gen:

```
...
<gene>
    <name type="primary">SCG5</name>
    <name type="synonym">SGNE1</name>
  </gene>
...
```

En este caso insertaríamos el nombre del gen (Description Primary en nuestra tabla) y el sinónimo (Description Synonym), además de nuestro identificador interno del gen junto con el identificador interno a la proteina asociada. Para realizar dicha inserción haremos uso de uno de los procedimientos almacenados: `"dbo.P_I_Gen"`. Este procedimiento nos insertará los valores necesarios en nuestra tabla Gen. Al igual que antes, primero establecemos la conexión y la abrimos, indicamos que vamos a ejecutar un procedimiento almacenado y añadimos los parámetros. Ejecutamos y recuperamos el número de registros afectados. Por último, desconectamos las conexiones.

Una vez insertado el organismo nos disponemos a buscar los nodos "keyword" de una proteina en nuestro documento XML y procedemos de la misma forma que en las proteinas, seleccionamos los nodos keyword y procesamos cada keyword. Veamos el formato en el que aparecen en el documento XML:

```
      ...
<dbReference type="Pfam" id="PF05281" key="59">
    <property type="entry name" value="Secretogranin_V"/>
    <property type="match status" value="1"/>
  </dbReference>
  <keyword id="KW-0025">Alternative splicing</keyword>
  <keyword id="KW-0143">Chaperone</keyword>
  <keyword id="KW-0165">Cleavage on pair of basic residues</keyword>
  <keyword id="KW-0903">Direct protein sequencing</keyword>
  <keyword id="KW-0527">Neuropeptide</keyword>
  <keyword id="KW-0597">Phosphorylation</keyword>
  <keyword id="KW-0964">Secreted</keyword>
  <keyword id="KW-0732">Signal</keyword>
  <keyword id="KW-0765">Sulfation</keyword>
  <keyword id="KW-0813">Transport</keyword>
  <feature type="signal peptide">
    <location>
      <begin position="1"/>
      <end position="26"/>
    </location>
  </feature>
      ...
```

Reconocemos cada keyword y lo insertamos en nuestra tabla Keywords, que se compone de tres campos, el identificador de su proteina correspondiente, el identificador del keyword y la descripción. Estos dos últimos campos son los que se observan en cada linea de cada uno de los keywords. A la hora de insertar cada keyword se realiza de la misma forma que en las anteriores inserciones. En este caso el procedimiento almacenado a utilizar es `"dbo.P_I_Keyword"`. Se establece y se abre la conexión, se indica la ejecución del procedimiento almacenado anterior, añadimos parámetros y ejecutamos recuperando el número de registros afectados. Finalmente desconectamos las conexiones.

De forma similar a las proteinas y a los keyword buscamos los nodos accession y realizamos la inserción por cada uno de ellos. La información a procesar aparece registrada de la siguiente forma:

```
<entry dataset="Swiss-Prot" created="1986-07-21" modified="2007-07-24"
version="73">
  <accession>P05408</accession>
```

```
<accession>P01164</accession>
<accession>Q9BS38</accession>
<name>7B2_HUMAN</name>
<protein>
  <name>Neuroendocrine protein 7B2 precursor</name>
  <name>Secretogranin-5</name>
  <name>Secretogranin V</name>
  <name>Secretory granule endocrine protein I</name>
  <name>Pituitary polypeptide</name>
    ...
```

Para cada inserción indicamos el procedimiento almacenado a ejecutar. En este caso el procedimiento es "dbo.P_I_Accession". Establecemos la conexión como en cada inserción, indicamos el procedimiento, añadimos parámetros y ejecutamos recuperando registros afectados. Por último, desconectamos.

Por último, buscamos los nodos "protein/name" para insertar todos los sinónimos de una proteina, y por cada uno realizamos una inserción.

```
...
    <protein>
        <name>Neuroendocrine protein 7B2 precursor</name>
        <name>Secretogranin-5</name>
        <name>Secretogranin V</name>
        <name>Secretory granule endocrine protein I</name>
        <name>Pituitary polypeptide</name>
...
```

Se procede de igual forma que en anteriores ocasiones. Ahora el procedimiento almacenado a llamar será "dbo.P_I_Synonymo".

Referente a las inserciones que hemos realizado para todas las tablas, podemos observar que todas siguen un patrón común. A modo resumen los pasos para realizar una inserción en cualquiera de nuestras tablas son:

- Indicamos el procedimiento almacenado a utilizar. Cada procedimiento fue diseñado junto a las tablas con la herramienta SQL Server.

```
string strProcedimiento = "Propietario.Nombre";
```

- Establecemos la conexión y la abrimos:

```
System.Data.OleDb.OleDbConnection oConn= new
System.Data.OleDb.OleDbConnection();

System.Data.OleDb.OleDbCommand oCmd =
new System.Data.OleDb.OleDbCommand();

RutaAplicacion = Application.StartupPath;
RutaAplicacion="File Name=" + RutaAplicacion + "\\ProteinUpload.udl;";
oConn.ConnectionString = RutaAplicacion;
oConn.Open();

oCmd.Connection = oConn;
```

```
oCmd.CommandText = strProcedimiento;
```

- Indicamos que vamos a ejecutar un procedimiento almacenado:

```
oCmd.CommandType = CommandType.StoredProcedure;
```

- Añadimos los parámetros del procedimiento almacenado:

```
oCmd.Parameters.Add("@nombreCampo",valor);
```

- Ejecutamos y recuperamos los registros:

```
resultado = oCmd.ExecuteNonQuery();
return resultado;
```

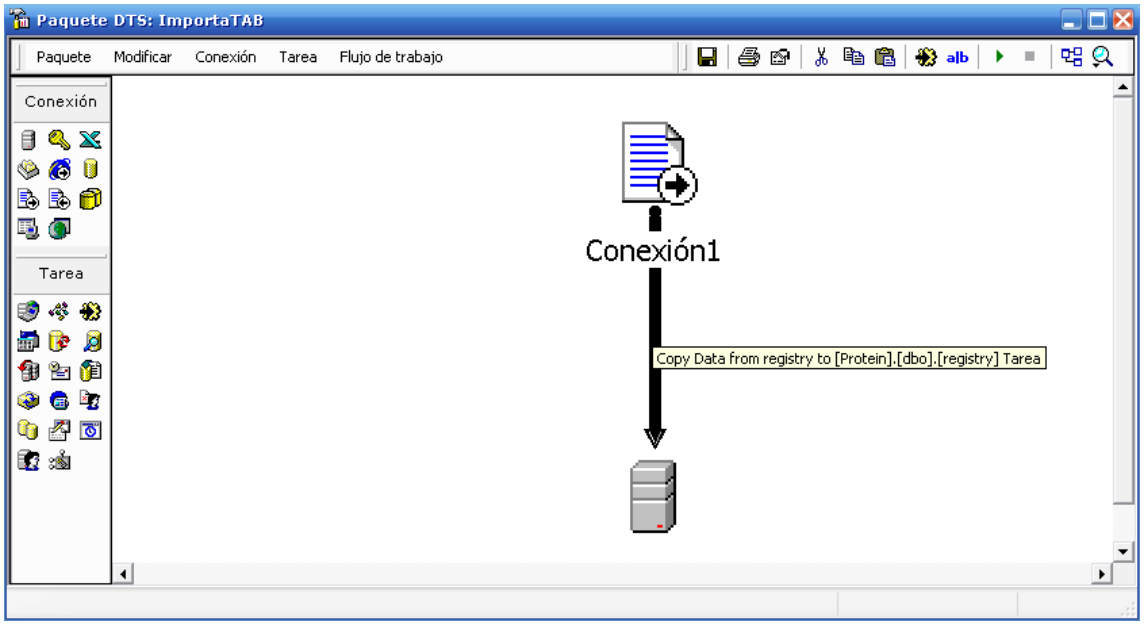
- Por último, desconectamos las conexiones para no dejarlas colgadas:

```
oCmd = null;
oConn.Close();
```

3.2.-CARGA DE DOCUMENTOS TABULADOS

SQL Server

Será aquí donde crearemos las tablas con todas las características de los genes/proteínas de las bases de datos descargadas a las que se hará referencia cuando ejecutemos el programa.
Esta aplicación tiene una particularidad, por la cual nos inclinamos a desarrollar el programa con esta base, y es que posee un "Servicio de transformación de datos" que nos es muy útil para las dos bases tabuladas que nos hemos descargado.
Gracias a este servicio, somos capaces de configurar la transformación de .tab a .sql fácilmente y sin ninguna complicación.



Con el programa, se entregarán dos archivos

- ImportaTAB
- ImportaTAB_Features

Estos dos Scripts serán los que cargaremos en el Servicio de transformación de datos, ya que están creados para transformar cada uno a la base de datos respectiva.

La transformación es bastante simple, los pasos son los siguientes:

Crearemos previamente una tabla destino, que tendrá los campos que queremos asignar a los futuros datos ("locus name", "sinónimos", etc) así, a la hora de hacer la transformación, el fichero origen se mapea directamente con la tabla destino, sin tener que asignar manualmente los datos.

Conexión1, donde seleccionaremos la base de datos origen, y donde especificaremos que tipo de archivo ha de transformar, delimitador de filas y especificar el delimitador de columnas (en nuestro caso Tabulado)

Servicio de Transformación, encargado de relacionar origen y destino.

Conexión2, que será donde se guardará el resultado de la transformación, tabla que hemos creado previamente.

ImportaTAB

Transforma la base de datos de la levadura (*"Saccharomyces"*), asigna a cada columna del fichero tabulado, su respectivo destino en fichero .sql

ImportaTAB_Features

En este caso, la transformación es un poco más elaborada, ya que no deseamos todos los campos del fichero tabulado, sino que únicamente queremos quedarnos con unos pocos. En un principio pensamos relacionar manualmente las columnas del origen que nos eran necesarias con la tabla de pocos elementos que habíamos creado, todo manualmente, quitando columnas y asignado destinos. Pero después nos dimos cuenta que era mucho más sencillo, y a posteriori fácil de usar el asignar todos los datos de origen a la tabla destino, con todas las columnas. Ahora bien, cuando realizamos la búsqueda, y el muestreo de datos, solamente utilizaremos las columnas del destino que nos eran necesarias, las otras 12 las ignoramos.

3.3.- REALIZACIÓN DE BÚSQUEDAS

Un usuario seleccionará un organismo entre aquellos que compongan nuestra base de datos. Nuestra base de datos está formada por cuatro tipos de organismos.

En caso de que un usuario seleccione a través de nuestro interfaz el organismo Candida albicans o Saccharomyces actuaremos de la siguiente forma. Internamente crearemos una estructura que cargará los datos encontrados en el resultado de la proteina que el usuario desee buscar para el organismo en cuestión. Si encuentra algún dato, inicia la generación de la lista de datos. Por cada proteina encontrada que contenga la cadena de caracteres introducida por el usuario para ese organismo generaremos una lista con todos sus datos. Mencionar que se busca por la palabra clave de la proteina, por lo que sólo habrá una, sin embargo, el usuario puede introducir una cadena de caracteres contenida simultáneamente en varios nombres de proteinas, siendo distinto el nombre de esas proteinas.

Para cada elemento de la lista de proteinas encontradas, creamos el nodo principal con el "Locus name" de la proteina. A este nodo le añadimos un subnodo "Campos". Para cada columna del conjunto de datos devuelto de la búsqueda generamos un subnodo dentro del subnodo "Campos". Generamos el subnodo añadiendo el nombre de la columna y valor para la proteina en la que estamos y se lo añadimos al subnodo "Campos". Finalmente, a nuestro nodo principal "Locus Name" le añadimos el subnodo Campos con sus subnodos correspondientes. Una vez hecho esto, lo añadimos a un componente visual creado anteriormente que generará la lista, con el fin de mostrar toda la información en pantalla a través de nuestro interfaz.

La estructura creada para encontrar las proteinas que contengan la cadena de caracteres introducidas como posible proteína para el organismo dado hará una búsqueda en nuestra base de datos para localizar toda la información que necesitamos. Hará una llamada a un procedimiento almacenado de nuestra base de datos en SQL Server. Establece una conexión y abre dicha conexión. A continuación, se indica que vamos a ejecutar un procedimiento almacenado y añadimos los parámetros. Por último, desconectamos las conexiones para no dejarlas colgadas. Si el organismo en cuestión es Candida albicans, ejecutamos el procedimiento almacenado `"dbo.P_S_CandidaFind".` Si es Saccharomyces ejecutamos el procedimiento almacenado `"dbo.P_S_SaccharomycesFind".` Ambos procedimientos no son más que sentencias de selección sobre cada una de las dos tablas de dichos organismos.

Si por el contrario, los organismos para los que queremos encontrar la proteina en cuestión son alguno de los otros dos organismos, de los 4 que estamos tratando, o incluso algún otro con las mismas características que el ratón y el genoma humano (es decir, habría que haberlos cargado previamente en nuestra base de datos con similar información a como fueron cargados esos dos organismos), procederemos de la siguiente forma. Como hicimos en los dos anteriores organismos, generaremos una estructura en la que almacenaremos las proteínas encontradas según el patrón introducido por el usuario. Para buscar las proteínas haremos uso del procedimiento almacenado `"dbo.P_S_ProteinaFind".` Procedemos de la misma forma a como hacemos siempre que ejecutamos un procedimiento almacenado. Establecemos la conexión y la abrimos, ejecutamos el procedimiento almacenado y rellenamos nuestra

estructura con la información que devuelve el procedimiento. Finalmente, desconectamos las conexiones.

Al igual que hicimos para los anteriores organismos, por cada proteína encontrada generamos una lista con sus datos. Generamos el nodo principal con el código y el nombre de la proteína. A continuación creamos el subnodo "Gens". Creamos otra estructura de datos para almacenar los genes de la proteína en la que estamos y los cargamos. Para ello, haremos uso de otro procedimiento almacenado, estableciendo la conexión, ejecutándolo y cerrando dicha conexión. Llamaremos al procedimiento almacenado `"dbo.P_S_ProteinGen"`.

Por cada gen añadimos un subnodo dentro de "Gens". Creamos un subnodo con el nombre y los sinónimos del gen, y se lo añadimos al nodo "Gens". Una vez hecho esto, añadimos el nodo "Gens" con todos sus subnodos al nodo principal.

A continuación, creamos un subnodo "Keywords" y creamos otra estructura de datos para para almacenar los Keywords de la proteína dada. Para cargar la información en nuestra estructura crearemos la conexión y ejecutaremos el procedimiento almacenado `"dbo.P_S_ProteinKeywords"`. Una vez hecho esto, cerraremos la conexión. Para cada "Keywords" añadimos un subnodo con la descripción del Keyword. Finalmente le añadimos el nodo "Keywords" con todos sus subnodos al nodo principal.

Lo siguiente será crear un subnodo "Accesión". Para almacenar toda la información referente a la tabla Accesión creamos una estructura de datos y cargamos en ella los datos Accesión de la proteina. Crearemos una conexión y la abriremos para posteriormente ejecutar el procedimiento almacenado `"dbo.P_S_ProteinAccessions"`, para posteriormente cerrar la conexión. Para cada Accesión añadimos un subnodo con la descripción de la Accesión, añadiendo todo esto al nodo principal.

El siguiente paso será crear un subnodo "Synonymous". Como en las anteriores ocasiones, generamos una estructura de datos para almacenar los Synonymous de la proteina en cuestión, y cargamos dicha información mediante el establecimiento de la conexión y la llamada al procedimiento almacenado `"dbo.P_S_ProteinSynonymous"`, con su posterior cierre de conexión. A este subnodo le asociaremos un subnodo con la descripción, añadiendo el bloque de nodos al nodo principal.

Por último, al igual que hicimos con los dos organismos anteriores, añadimos el nodo principal al componente visual que genera la lista.

4.-MANUAL

4.1.- REQUISITOS

A continuación, enunciamos el SW necesario para poder desarrollar el proyecto de manera eficiente:

- SQL Server Enterprise
- Visual Studio 2003
- Windows XP (o posteriores)
- PC (no existen restricciones de HW)
- Conexión a Internet

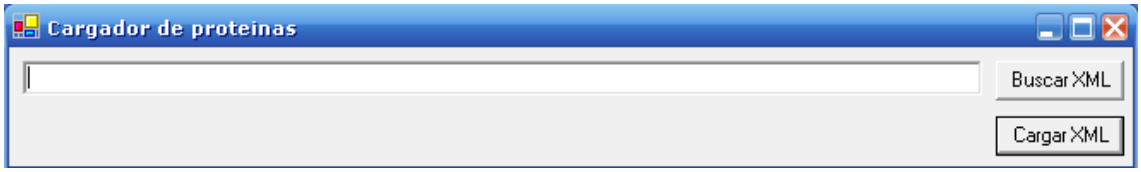
4.2.- CARGA DE LAS BASES DE DATOS

Puesto que nuestro proyecto, se encarga de unificar diversas bases de datos en distintos formatos sobre una misma, hemos de descargarlas a un mismo fichero desde Internet.
Existen 2 localizadas en la misma página (UNIProt) que poseen la misma estructura (XML) y de las cuales tomaremos las mismas características, y otras dos que se encuentran en bases de datos diferentes en un modelo de texto tabulado. Para ello, nos remitimos al apartado **2.1 Bases de datos públicas.**

Una vez descargadas, continuaremos con los siguientes pasos para unificarlas en nuestra base de datos.

- Carga de los archivos XML

Hemos desarrollado un buscador-cargador muy sencillo de usar que nos ayudará a insertar en nuestra base de datos los datos de los organismos que necesitemos.
Se trata de la aplicación **"ProteinUpload"**



A través de esta aplicación, podremos cargar los archivos XML que previamente hemos descargado de la base de datos de UNIProt. Si el archivo completo se fragmento en varias carpetas cuando lo guardamos, tendremos que repetir el proceso de búsqueda-carga tantas veces como fragmentos de la base haya.

Una de las particularidades que tiene el cargador, es que a la hora de cargar, si se encuentra con proteínas que ya existen en nuestra base de datos las descartará y únicamente se quedará con aquellas que no existan.

Otra ventaja del mismo, y a nuestro juicio la principal, es que permite la carga de tantos organismos distintos existan en las bases de datos XML como se quiera. Siempre y cuando estos mantengan el mismo formato que los del *Homo sapiens* y *Mus musculus*, ya que lo que nos importará serán los aspectos de nombre, synomims, accesions…

Una vez ejecutado el programa, este se habrá encargado de crear una base de datos SQL en la que encontraremos los organismos cargados como sus características, cuya relación podemos encontrar en el apartado **2.2.3.-Esquema lógico.**

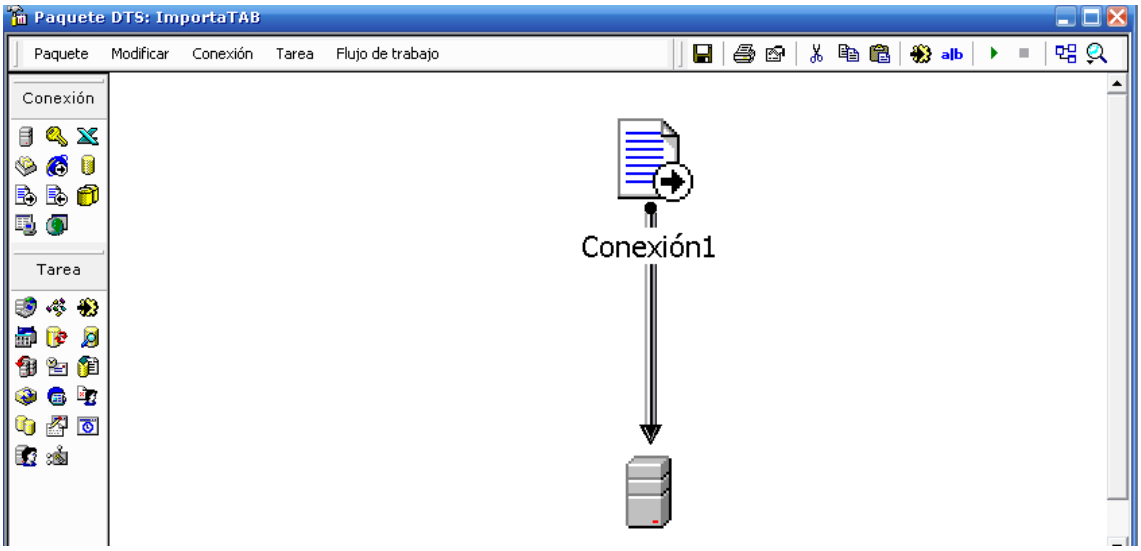
- Carga de archivos tabulados

En este caso, la carga es un poco más complicada de llevar a cabo, ya que en esta ocasión no hemos creado una aplicación que nos ayude. Lo que hemos hecho ha sido aprovechar los servicios que nos proporciona la aplicación SQL Server, concretamente el "Servicio de transformación de archivos".

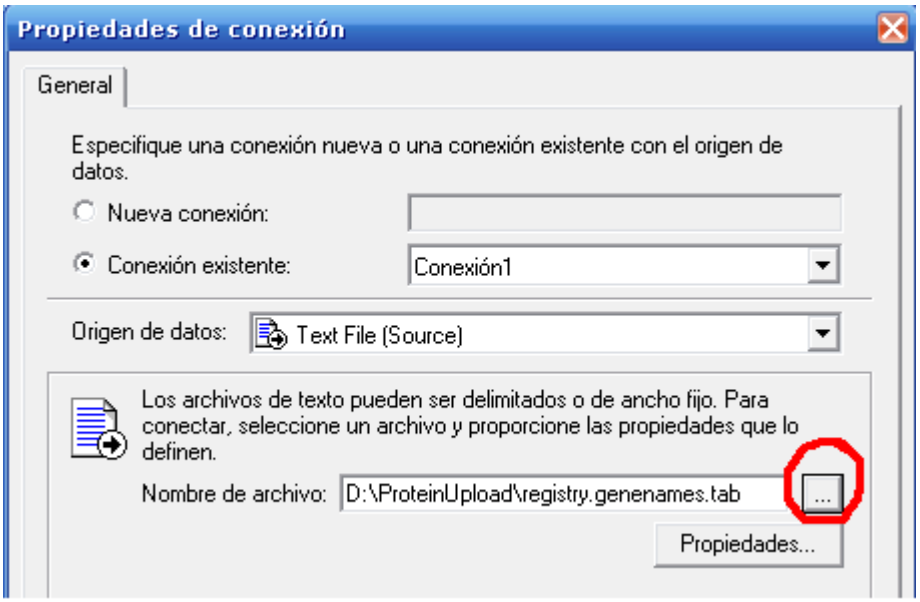
Al tratarse de dos ficheros tabulados con distintos campos, lo que hemos hecho ha sido crear una transformación para cada uno de ellos.
Explicaremos la carga de la levadura,ya que los pasos a seguir en ambos son idénticos.

Una vez abierto el administrador corporativo de SQL Server

Dentro de "Servicios de transformación de datos", deberemos de darle al botón derecho "Abrir paquete" y abriremos el paquete "ImportaTAB" (esta operación hay que hacerla únicamente una vez, las actualizaciones posteriores tendrán el paquete ya incorporado).
Una vez abierto, pinchamos sobre el y nos aparecerá el siguiente gráfico de transformación

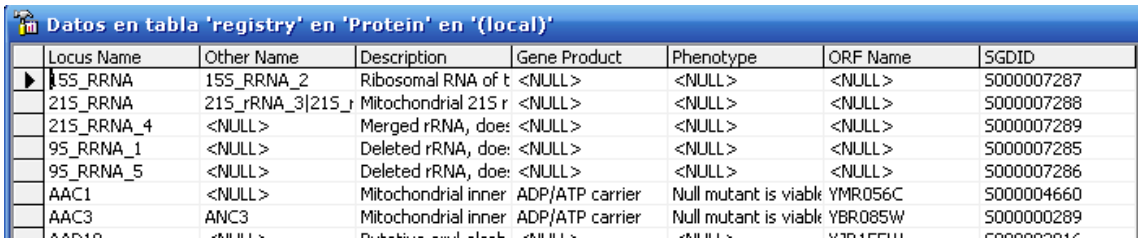


A continuación, haremos doble clic sobre la Conexión1 y nos aparecerá un menú, en el que nos dará la opción de seleccionar el archivo origen, ahí es donde cargaremos el fichero.



Para el destino, lo que hemos hecho ha sido crear una tabla con los campos que deseamos guardar.

Revisamos las características en la flecha de transformación tanto de origen como de destino, y si está todo correcto, ejecutamos y ya tenemos todos los datos cargados en nuestra tabla destino (en este caso "registry"), que nos quedará de este estilo:


Datos en tabla 'registry' en 'Protein' en '(local)'

Locus Name	Other Name	Description	Gene Product	Phenotype	ORF Name	SGDID
15S_RRNA	15S_RRNA_2	Ribosomal RNA of t	<NULL>	<NULL>	<NULL>	S000007287
21S_RRNA	21S_rRNA_3|21S_r	Mitochondrial 21S r	<NULL>	<NULL>	<NULL>	S000007288
21S_RRNA_4	<NULL>	Merged rRNA, does	<NULL>	<NULL>	<NULL>	S000007289
9S_RRNA_1	<NULL>	Deleted rRNA, does	<NULL>	<NULL>	<NULL>	S000007285
9S_RRNA_5	<NULL>	Deleted rRNA, does	<NULL>	<NULL>	<NULL>	S000007286
AAC1	<NULL>	Mitochondrial inner	ADP/ATP carrier	Null mutant is viable	YMR056C	S000004660
AAC3	ANC3	Mitochondrial inner	ADP/ATP carrier	Null mutant is viable	YBR085W	S000000289

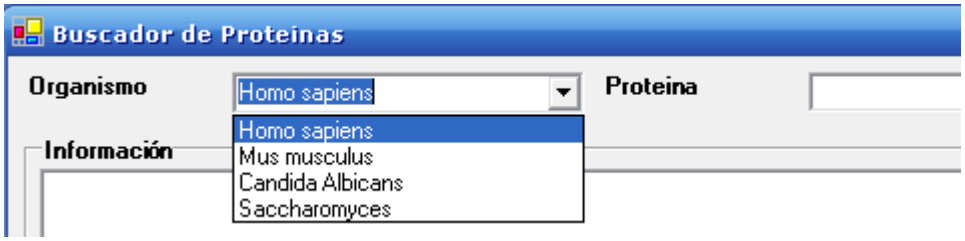
En el caso de la *Candida albicans,* el proceso de carga es similar. Cargando el paquete **ImportaTAB_Features,** y siguiendo los mismos pasos de carga origen-destino que en el anterior:

Seleccionamos como origen el "chromosomal_feature.tab" y como destino tendremos creada ya la tabla de "chromosomal_feature". Esta tabla posee 18 características, de las cuales, a la hora de cargarlas en el programa principal, nos quedaremos con 7, las mismas que en el caso del registry, las demás se omitirán.
De esta forma, nos quedaremos con los mismos campos de ambas tablas, lo que nos resultará útil para las posteriores búsquedas.

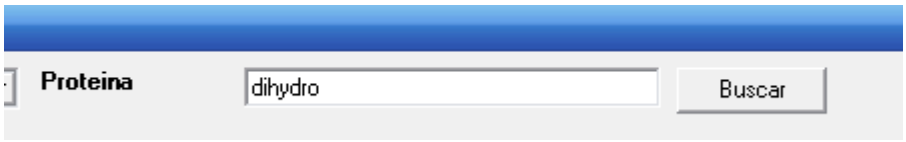
4.3.- USO DE LA APLICACIÓN PROTEINSEEK

Se trata de un interfaz muy simple y fácil de usar en el que dispondremos de dos campos de selección:

El primero, será la selección del "Organismo" en el que queremos buscar. Se trata de un menú desplegable en el que nos aparecerán todos los organismos que poseemos en nuestra base de datos.
Gracias a la ventaja que tenemos, la carga de los XML incorpora directamente el organismo a este menú: a medida que incorporamos, estos van apareciendo en dicho menú.

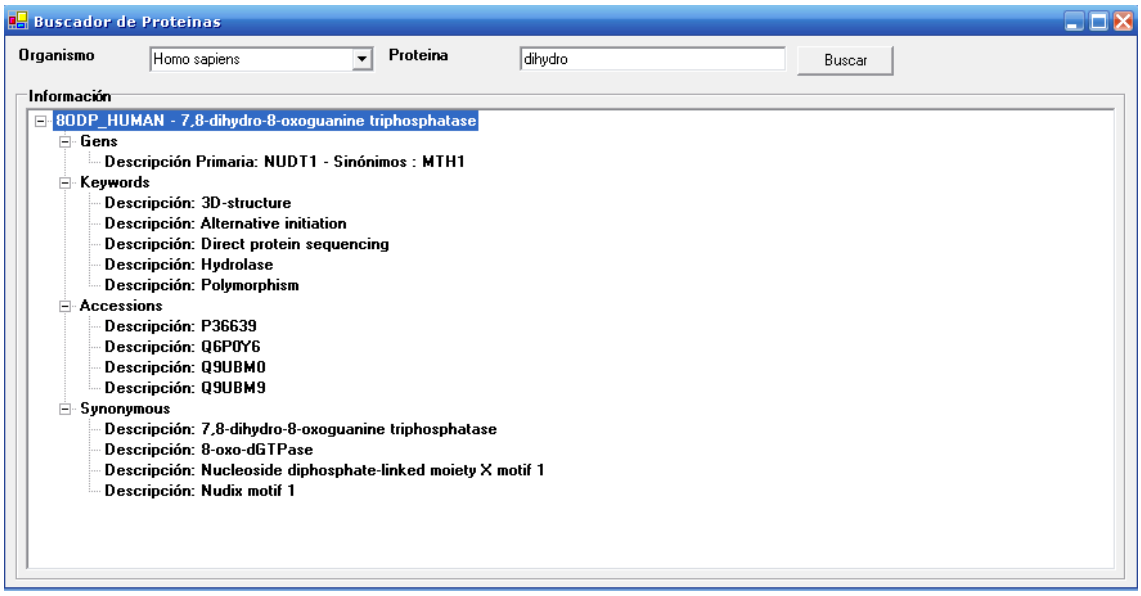


Una vez seleccionado el organismo, a continuación tendremos el campo "Proteína". En este, podemos poner el nombre completo o una parte de este, ya que la búsqueda no la hace por nombre exacto, sino que la hace en aquellos que contienen dicho fragmento en su nombre principal (XML) y locus name (Tabulados).



Elegimos la opción de buscar una vez seleccionados ambos campos, y si existe alguna proteína que en su nombre principal contenga "dihydro", nos mostrará todos los resultados obtenidos, tantas como existan en nuestra base. Si por el contrario no existe ninguna, no observaremos ninguna acción al respecto.

En este caso, observamos que tenemos una proteína dentro del organismo *"Homo sapiens"*. Si existieran por el contrario cierto número de ellas, nos las mostraría todas:

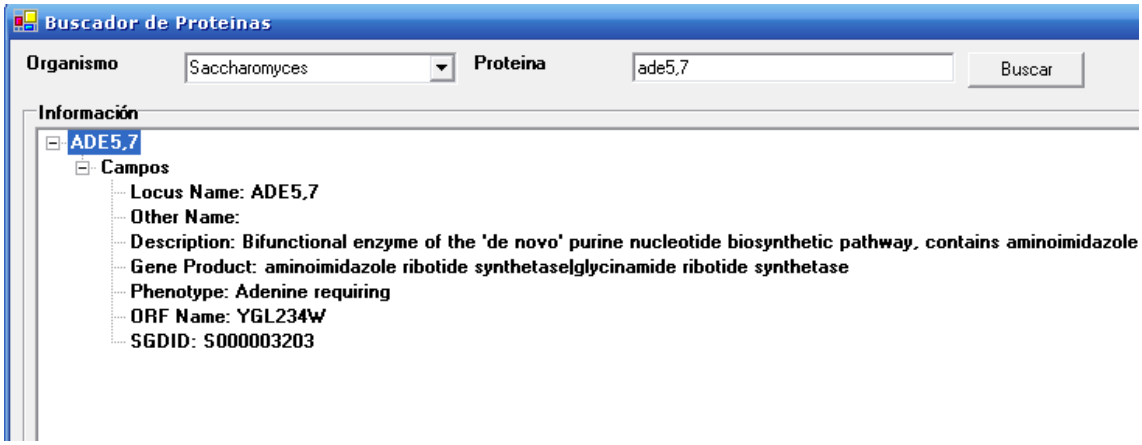


Campos del resultado (XML)

- Identificador de la proteína en la base UniProt seguido de su nombre principal.
- Gen asociado y sinónimos de este.
- Keywords o palabras clave asociados a la búsqueda de dicha proteína.
- Accessions o nombres con los que también se puede acceder a dicha proteína.
- Sinónimos de la misma, comenzando por su nombre principal.

En el caso de la búsqueda dentro de los organismos *"Candida albicans"* y *"Saccharomyces"* los campos del resultado difieren ligeramente de los mostrados anteriormente.

Buscaremos dentro de la base de datos de "*Saccharomyces*" aquella que posee el nombre **ADE5,7** por poner un ejemplo.



Campos del resultado ("*Saccharomyces*")

- Nombre principal
- Sinónimos
- Descripción
- Gen asociado
- Fenotipo
- ORF asociado
- Identificador dentro de la base de datos original (SGID)

Campos del resultado ("*Candida albicans*")

- Nombre característico (Feature name)
- Nombre principal (Locus name)
- Sinónimos (Aliases)
- Característica principal (Feature type)
- Identificador dentro de la base de datos original (CGID)
- Descripción

5.- PROYECTOS FUTUROS

Ampliación en el número de organismos en nuestra base de datos Protein:

Gracias a la estructuración de nuestro cargador para bases de datos XML, es posible ampliar el número de organismos existentes. Esto quiere decir, que si nos interesa incluir algún organismo más de la base de datos pública UniProt, seguiremos los mismos pasos que con el *"Homo sapiens"* y *"Mus musculus"*, descarga y posterior carga, siempre y cuando la información que se desea incluir sea la misma que en los otros dos casos.

Otro tipo de búsquedas:

Nuestra aplicación ProteinSeek, realiza las búsquedas a través de su nombre principal. Una posible mejora sería poder ampliar este campo de búsqueda y realizarlas a través de otro tipo de información, bien sea sinónimos, palabras clave, etc.

Migración a otras plataformas:

El proyecto está elaborado para ser utilizado bajo el sistema operativo Windows, utilizando para ello la herramienta SQL Server, si se desease utilizar bajo otro sistema operativo, ya sea Linux, una posible opción sería migrar toda esta base de datos a otras herramientas como por ejemplo MySQL.

Descarga de las bases de datos:

Ya que a la hora de descargarnos las bases de datos públicas, supone una tarea un tanto pesada, se podría elaborar una nueva aplicación cuya finalidad sería enlazar directamente estas bases de datos y cargarlas en la nuestra (Complicado de elaborar) de una forma directa.

6.- CONCLUSIONES

Una vez descargadas las bases de datos públicas, y por medio de la aplicación ProteinUpload y el Servicio de transformación de datos de SQL Server integradas en nuestra base de datos "Protein", ya tenemos el programa listo para realizar tantas búsquedas como se deseen realizar.

De este modo, el resultado de nuestro sistema será consultado por parte de las herramientas de análisis de texto biomédico, y junto al proyecto de Desarrollo de un sistema de búsqueda para la base de datos bibliográfica Medline, supondrá una ayuda en el contexto de las aplicaciones de clasificación y minería de textos en biomedicina.

Como resultado, tratamos de mermar el problema existente en la ambigüedad de la nomenclatura dentro de la bioinformática, aportando un método sencillo de consulta, que facilitará las labores de reconocimiento, relaciones, similitudes y diferencias de las proteínas y genes incluidos en la base de datos a partir de su nombre principal.

7.- BIBLIOGRAFÍA

http://www.inegi.gob.mx/inegi/contenidos/espanol/prensa/Contenidos/Articulos/tecnologia/relacional.pdf

http://www.clikear.com/manuales/sql/index.asp

http://technet.microsoft.com/es-es/library/ms174149.aspx

http://technet.microsoft.com/es-es/library/ms174122.aspx

http://technet.microsoft.com/es-es/library/ms174018.aspx

http://technet.microsoft.com/es-es/library/ms174633.aspx

Professional SQL Server 2005 programming / Robert Vieira (Indianápolis, IN : Wiley Technology Pub, 2006

Así es Microsoft Visual Studio.Net / Microsoft Corporation (Madrid, McGraw Hill D.L. 2001)