

UNIVERSIDAD DE MÁLAGA  
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA



UNIVERSIDAD DE MÁLAGA

**APLICACIÓN GESTORA DE BASES DE DATOS**

GRADO EN INGENIERÍA DE LA SALUD

MÁLAGA, 2018

Proyecto Bases de Datos Biológicas  
Ingeniería de la Salud

**Formato de Publicación de la Escuela Técnica Superior de Ingeniería  
Informática de Málaga**

Autores:

José Rodríguez Maldonado  
Paloma Domínguez Sánchez  
Naira Chiclana García

## Resumen

En este documento se trata un proyecto de la asignatura de Bases de Datos Biológicas que realizaron alumnos del grado de Ingeniería de la Salud de la Escuela Técnica Superior de Ingeniería Informática de Málaga.

En él se explican los procedimientos seguidos para la creación de bases de datos relacionales, en XML y No relaciones usando los gestores correspondiente a dichas bases. Además se comparan los tiempo de ejecución de diferentes consultas en los diferentes gestores y versiones de la bases.

Se incluye además los pasos seguidos para la realización de una interfaz gráfica que gestione estas consultas.

# Indice de contenidos

1. Introducción ??
  - 1.1. Contexto ??
  - 1.2. Motivación ??
  - 1.3. Objetivos del proyectos ??
  - 1.4. Estructura del documento ??
2. Tecnologías utilizadas en SGBD Relacionales y diseño de consultas ??
  - 2.1. Tecnologías utilizadas en SGBD Relacionales ??
    - 2.1.1. MariaDB ??
    - 2.1.2. MySQL ??
    - 2.1.3. Postgree ??
  - 2.2. Diseño de consultas ??
    - 2.2.1. Consultas simples ??
    - 2.2.2. Consultas avanzadas ??
3. Versiones optimizadas ??
  - 3.1. Motores de búsqueda ??
    - 3.1.1. MyISAM ??
    - 3.1.2. InnoDB ??
  - 3.2. Índices ??
  - 3.3. Conclusión de los gestores relacionales ??
    - 3.3.1. Evaluación de tiempo??
4. Tecnologías utilizadas en SGBD XML y diseño de consultas ??
  - 4.1. Tecnologías utilizadas en SGBD XML ??
    - 4.1.1. Docker ??
    - 4.1.2. ExitDB ??
    - 4.1.3. XQuery ??
  - 4.2. Diseño de consultas XML-XQuery ??
    - 4.2.1. Reducción db en XML ??
    - 4.2.2. Creación de consultas para db reducida ??
  - 4.3. Conclusión gestor XML ??
    - 4.3.1. Tiempos consultas ??
5. Tecnologías utilizadas en SGBD no relacionales y diseño de consultas NOSQL ??

- 5.1. Tecnologías utilizadas en SGBD no relacionales ??
  - 5.1.1. MongoDB ??
  - 5.1.2. JSON ??
  - 5.1.3. JavaScript ??
- 5.2. Diseño de consultas NoSQL ??
  - 5.2.1. Migración de datos a MongoDB ??
  - 5.2.2. Consultas para MongoDB a través de JavaScript ??
- 5.3. Conclusión gestor NoSQL ??
  - 5.3.1. Tiempos consultas ??
- 6. Clases Java ??
  - 6.1. Conexión con Bases de datos ??
    - 6.1.1. Connection Postgree ??
    - 6.1.2. Connection MariaDB ??
    - 6.1.3. Connection MySQL ??
    - 6.1.4. Connection XML ??
    - 6.1.5. Connection MongoDB ??
  - 6.2. Conexión BD-Interfaz??
    - 6.2.1. Petition Controller ??
- 7. Implementación de la interfaz gráfica ??
  - 7.1. Tecnologías usadas ??
    - 7.1.1. Angular JS ??
    - 7.1.2. JavaScript ??
    - 7.1.3. HTML ??
    - 7.1.4. CSS ??
  - 7.2. Diseño interfaz ??
- 8. Integración Back-end Front-end ??
  - 8.1. Tecnologías usadas ??
    - 8.1.1. Spring-Boot ??
    - 8.1.2. Spring Tool Suite??
    - 8.1.3. Java ??
    - 8.1.4. Maven ??
  - 8.2. Proceso integración ??
- 9. Conclusiones ??
- 10. GitHub ??
- 11. Webgrafía ??

## Indice de ilustraciones

- Figura 1: homo\_sapiens\_core en Firezilla ??
- Figura 2: Relaciones entre tablas de homo\_sapiens\_core ??
- Figura 3: 4 versiones de la base de datos ??
- Figura 4: Interfaz ExistDB en contenedor Docker ??
- Figura 5: Introducción de bd en eXistDB ??
- Figura 6: Consulta XML 1 ??
- Figura 7: Consulta XML 2 ??
- Figura 8: Consulta XML 3 ??
- Figura 9: Consulta XML 4 ??
- Figura 10: Consulta XML 5 ??
- Figura 11: Consulta 1 MongoDB ??
- Figura 12: Consulta 2 MongoDB ??
- Figura 13: Consulta 3 MongoDB ??
- Figura 14: Primer prototipo interfaz ??
- Figura 15: Prototipo final interfaz ??
- Figura 16: Tiempos devueltos por interfaz para las 4 BD ??
- Figura 17: Importación Angular ??
- Figura 18: Estructura de carpetas con spring-boot ??
- Figura 19: Eliminar node modules ??
- Figura 20: Primera prueba de integración ??
- Cuadro 1: tiempos de gestores relacionales ??

# 1. Introducción: Definición inicial del proyecto

En este capítulo se va a proporcionar al lector una visión general sobre el proyecto desarrollado. Para ello se ha dividido en los siguientes apartados: Contexto, motivación, objetivos establecidos y estructura de la memoria.

## 1.1. Contexto

El proyecto desarrollado se encuentra en el campo de las Bases de Datos y sus gestores. Vamos a comenzar con una explicación del concepto y algunas de sus posibles aplicaciones.

Una base de datos es un conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su posterior uso. En este sentido; una biblioteca puede considerarse una base de datos compuesta en su mayoría por documentos y textos impresos en papel e indexados para su consulta. Actualmente, y debido al desarrollo tecnológico de campos como la informática y la electrónica, la mayoría de las bases de datos están en formato digital, siendo este un componente electrónico, por tanto se ha desarrollado y se ofrece un amplio rango de soluciones al problema del almacenamiento de datos. Existen programas denominados sistemas gestores de bases de datos, abreviado *SGBD* (del inglés Database Management System o DBMS), que **administran y gestionan la información que contiene una base de datos**. A través de él se maneja todo acceso a la base de datos con el objetivo de servir de interfaz entre ésta, el usuario y las aplicaciones. Las propiedades de estos SGBD, así como su utilización y administración, se estudian dentro del ámbito de la informática.

En cuanto a las posibles aplicaciones de los *SGBD*, podemos encontrar la gestión de empresas e instituciones públicas; pero también, y más en el campo que nos concierne, son ampliamente utilizadas en entornos científicos con el objeto de almacenar la información experimental.

Existen diferentes modelos de gestores de bases de datos, en este documento se tratan principalmente bases de datos **relacionales**, aunque también se hace uso de un modelo *XML* y **no relacional**.

Las bases de datos relacionales son utilizadas en la actualidad para representar problemas reales y administrar datos dinámicamente. Tras ser postulados sus fundamentos en 1970 por Edgar Frank Codd, de los laboratorios IBM en San José (California), no tardó en consolidarse como un nuevo paradigma en los modelos de base de datos. Su idea fundamental es el uso de relaciones". Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados "*tuplas*". La información puede ser recuperada o almacenada mediante consultas" que ofrecen una amplia flexibilidad y poder para administrar la información.

El lenguaje más habitual para construir las consultas a bases de datos relacionales es **SQL** *Structured Query Language* o Lenguaje Estructurado de Consultas, un estándar implementado por los principales motores o sistemas de gestión de bases de datos relacionales.

## 1.2. Motivación

Como se ha comentado anteriormente, las bases de datos tienen numerosas aplicaciones, pero en este caso se ha optado por el ámbito de las Bases de Datos Biológicas para la definición del proyecto. En este ámbito, las bases de datos son de gran ayuda pues permiten recoger gran cantidad de información ordenada y diferenciada de los numerosos experimentos científicos que tienen lugar a nivel de la Genómica, Transcriptómica y Metabolómica.

En los últimos años, debido a la rápida evolución de las técnicas experimentales de alto rendimiento (Secuenciación del ADN, Cristalografía de rayos X, Microarreglo de ADN) se generó un crecimiento exponencial en la cantidad de datos biológicos (secuencias genómicas y de proteínas, estructuras de proteínas, expresión génica, mutaciones, etc) que generaron la necesidad de contar con formas eficientes de almacenar la información.

Esta aplicación es muy interesante para todos los grupos de investigación a nivel mundial, pues permiten un uso compartido de la información que ayuda al desarrollo de la investigación de una forma mas eficiente y dinámica.

Además de lo comentado, el almacenamiento de estos datos se realiza de manera histórica, por lo que, no solo tenemos en cuenta los datos actuales, sino que podemos realizar comparaciones y estadísticas teniendo en cuenta datos referentes a otros años. Por lo tanto, además de poder realizar análisis más precisos, nos permite evaluar si las decisiones que se tomaron en un determinado momento de la historia fueron oportunas. Así que, también es posible evitar cometer los mismos errores que se pudieron cometer al realizar experimentos en un determinado momento de la historia.

Para este propósito es necesario realizar una evaluación de los sistemas de almacenamiento. Este proyecto también comprende este aspecto, ya que el rendimiento, capacidad de almacenamiento y rapidez de consulta son factores determinantes para conseguir la solución óptima.



### 1.3. Objetivos del proyecto

El objetivo principal del proyecto es desarrollar una aplicación capaz de analizar, comparar y estudiar la velocidad de consulta de diferentes sistemas de gestión de bases de datos idóneas para una base de datos. En él se hace uso de tres SGBD relacionales:

**MariaDB**, **MySQL** y **Postgres**, un SGBD *XML* (**ExistDB**) y un SGBD no relacional (**MongoDB**.)

Una vez realizada la consulta y el estudio del tiempo requerido de estos sistemas de gestión de bases de datos, se procederá a la elección de uno de ellos, teniendo en cuenta el análisis realizado y el rendimiento obtenido, y se obtendrá la consulta realizada.

### 1.4. Estructura del documento

El presente documento se encuentra dividido en varios capítulos, recogiendo toda la información relacionada con el desarrollo de este proyecto. A continuación, se ofrece una breve descripción de cada uno de ellos.

- **CAPÍTULO 1 – INTRODUCCIÓN.**

En este capítulo se establece el contexto que abarca el desarrollo del proyecto, describiendo su temática, objetivos que se desean alcanzar y las motivaciones que han llevado a su desarrollo.

- **CAPÍTULO 2 – TECNOLOGÍAS UTILIZADAS EN SGBD RELACIONES Y DISEÑO DE CONSULTAS.**

Durante este capítulo se describen las tecnologías utilizadas en el desarrollo de los sistemas gestores de MariaDB, MySQL y Postgres; así como las características principales de cada una de ellas. Además, se explican las razones por las que se ha decidido el uso de esa tecnología.

- **CAPÍTULO 3 – VERSIONES OPTIMIZADAS**

Este capítulo describe la metodología seleccionada para optimizar los gestores utilizados en el capítulo anterior.

- **CAPÍTULO 4 – TECNOLOGÍAS UTILIZADAS EN SGBD XML Y DISEÑO DE CONSULTAS.**

Durante este capítulo se describen las tecnologías utilizadas en el desarrollo del sistema gestor ExistDB y las consultas XML

- **CAPÍTULO 5 – TECNOLOGÍAS UTILIZADAS EN SGBD NO RELACIONALES Y DISEÑO DE CONSULTAS NOSQL**

En este capítulo se describen las tecnologías utilizadas en el desarrollo del sistema gestor MongoDB y las consultas NoSQL

- **CAPÍTULO 6 - CLASES JAVA**

Todas las clases usadas para conectarse con los gestpres y realizar las consultas.

- **CAPÍTULO 7 – IMPLEMENTACIÓN DE LA INTERFAZ GRÁFICA**

Usada para meter los datos con los que se creará á consulta y visualizar los resultados.

- **CAPÍTULO 8 -INTEGRACIÓN BACK-END FRONT-END**

Durante este capítulo se describe la evaluación realizada a los sistemas de almacenamiento, detallando las pruebas de almacenamiento realizadas y explicando los resultados obtenidos.

- **CAPÍTULO 9 – CONCLUSIONES Y TRABAJOS FUTUROS.**

Durante este capítulo se aportaran las conclusiones obtenidas a lo largo del desarrollo del proyecto, teniendo en cuenta los objetivos y el todo el proceso de desarrollo. Además, se describen los conocimientos aprendidos a lo largo del proyecto y algunos aspectos a tener en cuenta en trabajos futuros.

- **CAPÍTULO 10 – WEBGRAFÍA**

Referencias a documentos y webs que se han usado para realizar la redacción de este documento.

## 2. Tecnologías utilizadas en SGBD Relacionales y diseño de consultas

### 2.1. Tecnologías utilizadas en SGBD Relacionales: Despliegue en 3SGBD

En este apartado se explican los sistemas gestores de Bases de Datos escogidos, el proceso de instalación de cada uno de ellos y su puesta en marcha de la base de datos **Ensembl** (sin índices, ni optimizaciones).

Gestores elegidos:

- MySQL
- MariaDB
- PostgreSQL

*Debido a dificultades técnicas decidimos cambiar Oracle por Postgree.*

A continuación se explica el proceso de instalación de cada gestor y puesta en marcha de la base de datos:

1. Descargar la base de datos *Ensembl* y organismo *homo\_sapiens\_core\_92\_38*.

Lo haremos usando *FireZilla* con la dirección

<http://www.ensembl.org/info/data/ftp/index.html>

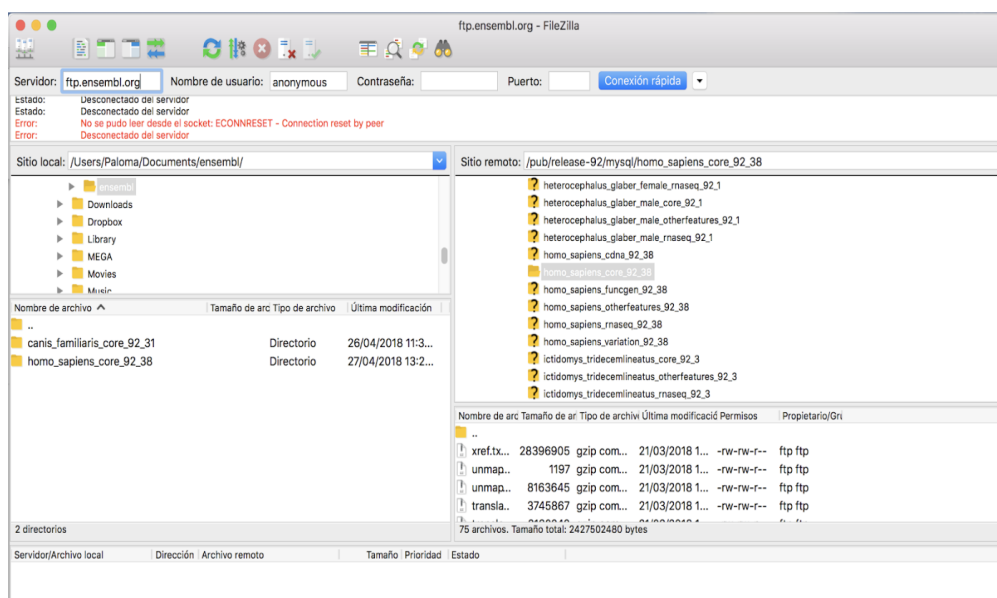


Figura 1: h  
omo\_sapiens\_core en Firezilla

2. Crear usuario y contraseña para los gestores.

Procedimiento para cada gestor:

### 2.1.1. MariaDB

MariaDB es un sistema de gestión de bases de datos derivado de *MySQL* con licencia GPL (General Public License). Es desarrollado por Michael (Monty) Widenius (fundador de *MySQL*), la fundación *MariaDB* y la comunidad de desarrolladores de software libre. Tiene una alta compatibilidad con *MySQL* ya que posee las mismas órdenes, interfaces, APIs y bibliotecas, siendo su objetivo poder cambiar un servidor por otro directamente.

1. Entrar en el gestor a través de terminal

```
$mysql -u root -p
```

```
Enter password:
```

```
Welcome to the MariaDB monitor. Commands end with ; or \g.
```

```
Your MariaDB connection id is 8
```

```
Server version: 10.2.13-MariaDB Homebrew
```

```
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.
```

```
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
```

```
MariaDB [(none)]>
```

2. Ver base de datos existentes:

```
MariaDB [(none) ]> show databases;
```

```
+-----+
| Database          |
+-----+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+-----+
```

3. Crear nueva bd

```
MariaDB [(none)] > create database bdb_sisa;
```

4. Entrar en directorio que contenga los archivos e insertar estructura de tablas:

```
alu-239-204:homo_sapiens_core_92_38 nairachiclana$ mysql -u root -p
bdb_sisa< homo_sapiens_core_92_38.sql
```

5. Comprobar que se han introducido las tablas correctamente:

```
MariaDB [(none)]> use bdb_sisa;
MariaDB [bdb_sisa]> show tables;
```

```
+-----+
| Tables_in_homo_sapiens_core_91_38 |
+-----+
| alt_allele |
| alt_allele_attrib |
| alt_allele_group |
| analysis |
| analysis_description |
| assembly |
| assembly_exception |
| associated_group |
| associated_xref |
| attrib_type |
| coord_system |
| data_file |
| density_feature |
| density_type |
| ..... |
```

Ver datos de alguna de las tablas:

```
MariaDB [homo_sapiens_core_91_38]> describe alt_allele;
```

```
+-----+-----+-----+-----+-----+
| Field | Type | Null | Key | Default |
+-----+-----+-----+-----+-----+
| alt_allele_id | int(10) unsigned | NO | | |
| alt_allele_group_id | int(10) unsigned | NO | | |
| gene_id | int(10) unsigned | NO | | |
+-----+-----+-----+-----+-----+
```

6. Volcar los datos en las tablas creadas:

(Debemos estar dentro del directorio de la carpeta que contiene el archivo con los datos)

```
alu-239-204:homo_sapiens_core_92_38 nairachiclana$
mysqlimport -u root -p --fields-terminated-by='\t'
--fields_escaped_by=\\ bdb_sisa -L *.txt
```

Para ello usaremos la clase en *Java* que se crea la conexión con el gestor y le aplicaremos la consulta:

### 2.1.2. MySQL

Se usan los mismos comandos que para *MariaDB* :

```
mysql -u root -p

create database bdb_sisa;

mysql -u root -p bdb_sisa< homo_sapiens_core_92_38.sql

mysqlimport -u root -p --fields-terminated-by='\t'
--fields-escaped-by=\\ bdb_sisa -L *.txt
```

### 2.1.3. PSQL

Priemero crearemos la base de datos en otro gestor (*MariaDB*) y luego las exportaremos a *Postgres*.

1. Exportar bd creadas en *MariaDB* a .mysql

```
$mysqldump --compatible=postgresql
--default-character-set=utf8 -r bdb_sisa.mysql -u root -p
bdb_sisa --max_allowed_packet=512M
```

Esta orden devuelve la base de datos `bdb_sisa.sql` en un archivo `bdb_sisa.mysql` compatible con *Postgres*.

En principio con esto ya deberíamos obtener una base de datos dentro del .mysql que pudiera importar bien dentro de postgres, pero daba errores. Todos errores de sintaxis que *psql* no era capaz de reconocer por no haber traducido bien las sentencias. Para arreglar este **fallo**, he recurrido al uso de un script de python que cambia la sintaxis de MySql/*MariaDB* a PSQL.

```
$python db_converter.py bdb_sisa.mysql bdb_sisa.psql
```

El archivo `bdb_sisa.psql` ya si tiene una versión legible por *psql*.

Con todo esto todavía seguíamos encontrando algún fallo a la hora de importar las BD, al importar empezaba bien, pero llegado a cierto punto obteníamos el siguiente fallo:

```
ERROR: invalid input value for enum
coord_system_attrib: "default_version,sequence_level"
```

```
LINE 1: ...T INTO "coord_system" VALUES (1,1,'contig', NULL, 4, 'default_v...
```

La única forma en la que hemos podido solventarlo ha sido cambiando los tipos de las variables de la base de datos. En vez de mantener *coord system* como un *enumerado* (en vista de que el script no hacía bien estos cambios), lo cambiamos por un *VARCHAR(250)* haciendo los cambios directamente sobre la tabla en MySQL Workbench. Utilizamos la siguiente sentencia:

```
ALTER TABLE coord_system CHANGE attrib attrib VARCHAR(250);
```

Haciendo este cambio sobre todas las bases de datos, se corrige el error y ya se pueden crear las bases de datos e introducirle los datos del archivo psql correcto (*bdb\_sisa.psql* ).

2. Crear las bases de datos:

```
create database bdb_sisa;
```

3. Comprobar que se ha creado la bd:

```
$postgres=# \l
```

```
bdb_sisa      | postgres | UTF8      | es_ES.UTF-8 | es_ES.UTF-8 |
```

4. Entrar a la bd y comprobar que se han creado las tablas:

```
$postgres=# \c bdb_sica;
```

```
psql (10.3 (Ubuntu 10.3-1), server 9.6.8)
```

```
You are now connected to database "bdb_sisa" as user "postgres".
```

```
$bdb_sisa=# \dt
```

```
public | alt_allele | table | postgres
public | alt_allele_attrib | table | postgres
public | alt_allele_group | table | postgres
public | analysis | table | postgres
.....
```

5. Importar los datos dentro:

```
sudo -i -u postgres psql bdb_sisa < bdb_sisa.psql
```

## 2.2. Diseño de consultas

En esta sección se trabaja con el diseno de las consultas que darían soporte a la Web de esa base de datos biológica.

Dado que solo tenemos un organismo en la base de datos, nuestras consultas están basadas en dicho organismo. A continuación se adjunta una imagen donde se pueden ver claramente las relaciones entre las tablas, a partir de la cual se han diseñado las consultas.

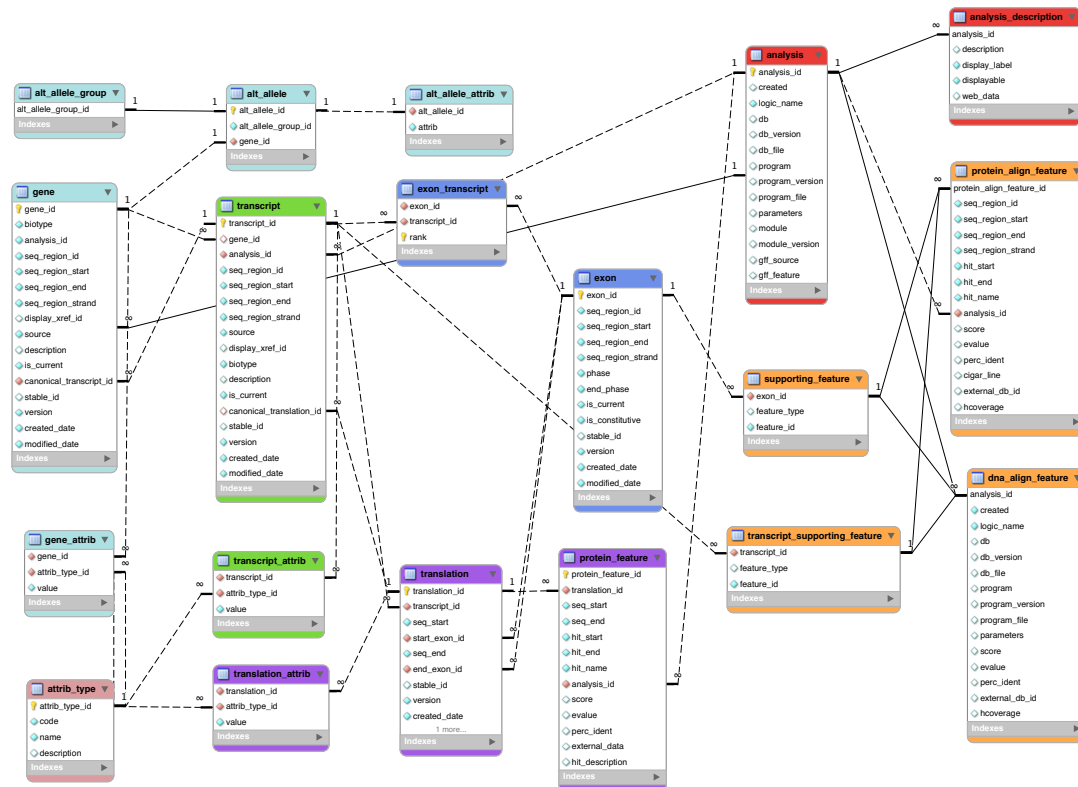


Figura 2: Relaciones entre tablas de homo\_sapiens\_core



### 2.2.1. Consultas simples

- Consulta para buscar un trozo de secuencia:

```
SELECT * FROM homo_sapiens_core_91_38.dna
WHERE sequence LIKE '%AGGTGTTA%';
```

- Consulta sobre genes:

```
SELECT * FROM homo_sapiens_core_91_38.gene
```

- Consultas de primers:

```
SELECT left_primer FROM homo_sapiens_core_91_38.marker
WHERE marker_id = 1;
```

- Para hacer consultas acerca de en qué cromosoma se encuentra cierto primer:

```
SELECT chromosome_name FROM homo_sapiens_core_91_38.marker_map_location;
```

- Consulta acerca de péptidos:

```
SELECT * FROM homo_sapiens_core_91_38.peptide_archive
WHERE peptide_seq LIKE '% GTRLPAERLQ%';
```

### 2.2.2. Consultas avanzadas

- Consulta sobre la descripción de los genes cuyos identificadores coinciden con los alelos correspondientes:

```
SELECT description
FROM alt_allele al, gene g WHERE al.gene_id = g.gene_id
```

- Consulta de identificadores de genes iguales en las tablas de atributo y grupo:

```
SELECT g.gene_id
FROM gene g, gene_attrib ga, attrib_type aty
WHERE g.gene_id = ga.gene_id AND ga.attrib_type_id = aty.attrib_type_id
```

- Consulta sobre los identificadores de los exones y genes:

```
SELECT gene_id, ex.exon_id
FROM transcript tr, exon_transcript extr, exon ex
WHERE tr.transcript_id = extr.transcript_id
AND extr.exon_id = ex.exon_id
```

- Consulta sobre el identificador del exon, fase de terminación y biotipo, ordenada:

```
SELECT ex.exon_id, end_phase, biotype
FROM transcript tr, exon_transcript extr, exon ex
WHERE tr.transcript_id = extr.transcript_id
AND extr.exon_id = ex.exon_id AND biotype='miRNA'
ORDER BY exon_id;
```

- Consulta agrupada:

```
SELECT alt_allele, alt_allele_group_id, attrib
FROM alt_allele al, alt_allele_group g, alt_allele_attrib at
WHERE al.alt_allele_id = g.alt_allele_group_id
AND al.alt_allele_id = at.alt_allele_id
GROUP BY attrib
```

### 3. Versiones optimizadas

En esta sección del document se modifican los **índices** y **motores de almacenamiento**. Para ello vamos a crear 4 versiones de nuestra base de datos en cada gestor, cuyas características serán las definidas a continuación:

- V1: bdb\_sisa  
Sin índices ni almacenamiento (Sin índices + Motor de búsqueda *MyISAM*)
- V2: bdb\_cisa  
Con índices y sin almacenamiento (Índices + Motor de búsqueda *MyISAM*)
- V3: bdb\_sica  
Sin índices pero con almacenamiento (Sin índices + Motor de búsqueda *InnoDB*)
- V4: bdb\_cica  
Con índices y con almacenamiento (Índices + Motor de búsqueda *InnoDB*)

#### 3.1. Motores de búsqueda

En *MySQL* existen diferentes motores de almacenamiento. Cada motor de almacenamiento trabaja con un tipo de tabla. Los dos tipos de tablas más importantes son *MyISAM* e *InnoDB*. Aunque *MyISAM* es la que viene por defecto, recomendamos trabajar con *InnoDB* ya que este es el único tipo de tabla que admite **transacciones**. Enumeraremos de manera breve las propiedades de estos dos tipos de tabla:

##### 3.1.1. MyISAM

- Dentro de cada carpeta de base de datos existen tres tipos de archivos por cada tabla, *frm* (formato), *MYD* (datos) y *MYI* (índices).
- Al no manejar transacciones es más **rápido** que *InnoDB*

##### 3.1.2. InnoDB

- Necesitan de un archivo (o varios) situado en el directorio de datos (por defecto). Estos archivos forman el espacio de tablas y por defecto todas las bases de datos almacenan sus datos e índice en el espacio de tablas. Debajo de cada carpeta de base de datos existe un tipo de archivo por cada tabla, *frm* (formato). Se puede especificar que **cada tabla guarde sus datos en un archivo diferente**. En este caso en la carpeta de la base de datos habrá otro tipo de archivo por cada tabla, *idb* (datos e índices). Aún así se sigue utilizando el espacio de tablas para guardar información sobre los diccionarios de datos que utiliza el servidor.

- Admite **transacciones**.
- Soporte de **claves externas e integridad referencial**

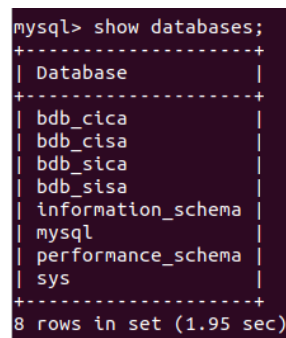
MyISAM viene por defecto en las tablas de la base de datos. Para las versiones con almacenamiento optimizado (bdb\_sica y bdb\_cica), cambiaremos el motor a InnoDB con la consulta:

```
"ALTER TABLE" + table + "ENGINE=InnoDB";

//Guardar los nombres de todas las tablas
selectSql = "select * from information_schema.tables where
    table_schema = 'bdb_cica'";
List<String> nombreTablas = new ArrayList<>();
result = statement.executeQuery(selectSql);
while (result.next()) {
    nombreTablas.add(result.getString(3));
}

//Realizar consulta a todas las tablas
for (int i = 0; i < nombreTablas.size(); i++) {
    String table = nombreTablas.get(i);
    System.out.println(table);
    selectSql2 = "ALTER TABLE " + table + " ENGINE=InnoDB;";
    statement2 = connection.createStatement();
    statement.executeUpdate(selectSql2);
}
while (result2.next()) { }
```

Comprobamos que los cambios se han aplicado correctamente:



```
mysql> show databases;
+-----+
| Database |
+-----+
| bdb_cica |
| bdb_cisa |
| bdb_sica |
| bdb_sisa |
| information_schema |
| mysql |
| performance_schema |
| sys |
+-----+
8 rows in set (1.95 sec)
```

Figura 3: 4  
versiones de la base de datos

- bdb\_sisa

```
MariaDB [bdb_sisa]> select table_name,engine from information_schema.tables
where table_schema='bdb_sisa';
```

```
+-----+-----+
| table_name | engine |
+-----+-----+
```

table_name	engine
alt_allele	MyISAM
alt_allele_attrib	MyISAM
alt_allele_group	MyISAM
analysis	MyISAM
analysis_description	MyISAM
.....	

#### ■ bdb\_cisa

MariaDB [bdb\_cisa]> select table\_name,engine from information\_schema.tables where table\_schema='bdb\_cisa';

table_name	engine
alt_allele	MyISAM
alt_allele_attrib	MyISAM
alt_allele_group	MyISAM
analysis	MyISAM
analysis_description	MyISAM
.....	

#### ■ bdb\_sica

MariaDB [bdb\_sica]> select table\_name,engine from information\_schema.tables where table\_schema='bdb\_sica';

table_name	engine
alt_allele	InnoDB
alt_allele_attrib	InnoDB
alt_allele_group	InnoDB
analysis	InnoDB
analysis_description	InnoDB
.....	

#### ■ bdb\_cica

MariaDB [bdb\_cica]> select table\_name,engine from information\_schema.tables where table\_schema='bdb\_cica';

table_name	engine
alt_allele	InnoDB
alt_allele_attrib	InnoDB
alt_allele_group	InnoDB
.....	

## 3.2. Índices

En el esquema de tablas y relaciones mostrado en el apartado ??, vemos como cada tabla tiene una clave primaria que funciona como índice por defecto. Para aumentar aún más la eficiencia (reducir el tiempo de búsqueda) crearemos nuevos índices que optimicen las consultas.

```
SELECT ex.exon_id, end_phase, biotype
FROM transcript tr, exon_transcript extr, exon ex
WHERE tr.transcript_id = extr.transcript_id
      AND extr.exon_id = ex.exon_id
      AND biotype='miRNA'
ORDER BY exon_id, ex.exon_id, end_phase, biotype;
```

s Donde se pregunta por biotype, atributo no primario, éste lo transformaremos en índice de la manera siguiente:

```
CREATE INDEX biotype ON transcript(biotype);
```

Finalmente, tras haber creado las cuatro versiones de nuestra base de datos (con índices y almacenamiento) en cada uno de los tres gestores, comparamos los tiempos de consulta en cada versión basándonos en la siguiente consulta. Mediremos los tiempos para la consulta

```
SELECT ex.exon_id, end_phase, biotype
FROM transcript tr, exon_transcript extr, exon ex
WHERE tr.transcript_id = extr.transcript_id
      AND extr.exon_id = ex.exon_id
      AND biotype='miRNA'
```

## 3.3. Conclusión de los gestores relacionales

### 3.3.1. Evaluación de tiempo

Cuadro 1: Tiempos

	MySQL(Docker)	MariaDB	Postgree
bdb_sisa	11.96s	2.489s	3.965s
bdb_sica	2.59s	3.273s	4.012s
bdb_cisa	2.02s	2.405s	2.02s
bdb_cica	2.005s	2.417s	3.973

## 4. Tecnologías utilizadas en SGBD XML y diseño de consultas

En este apartado vamos a trabajar con un SGBD *XML* denominado *ExistDB*, para el cual se diseñan un modelo de datos *XML* y se generan los datos en *XML* para nuestra base de datos que se almacenan en el SGBD seleccionado.

Se añade al final de este procedimiento las consultas realizadas en SQL en *XQuery*.

Hacemos uso *Docker*, un proveedor de contenedores que aborda cada aplicación a través de contenedores virtuales.

Para ello se ha cogido una de las bases de datos que teníamos subida a *MariaDB*, de la cual hemos exportado exclusivamente tres tablas para basarnos en una consulta.

### 4.1. Tecnologías utilizadas en SGBD XML

#### 4.1.1. Docker

Como contenedor en el que se ha guardado la imagen de *ExitDB*.

Creación de la imagen de *Docker* :

##### 1. Instalación imagen MySQL

###### 1.1. Instalación imagen MySQL

```
$docker pull mysql:5.7.17
```

###### 1.2. Ejecutar la imagen

```
$docker run --name mysql-container  
-e MYSQL_ROOT_PASSWORD = secret -p  
3306:3306 -d mysql:5.7.17
```

Con este comando estamos ejecutando la imagen *mysql* con el tag 5,7,1. Se crea un contenedor con el nombre *container-name* y expone el puerto 3306 (local y virtual) para que podamos conectarnos con un cliente a la base de datos. Además, cuando se crea la instancia de *MySQL*, ésta se crea por defecto con el usuario *root* y la password la que hayamos establecido en *MYSQL\_ROOT\_PASSWORD*.

##### 2. Instalación imagen ExistDB

###### 2.1. Descargar la imagen del gestor:

```
$docker pull evolvedbinary/exist-db:exist-4.1.0
```

###### 2.2. Ejecutar la imagen del gestor (en el puerto 9080):

```
$docker run -d -p 9080:9080 -e EXIST_ADMIN_PASSWORD=name -v  
localPath:/opt/exist_data/export/ davidgaya/existdb:latest|
```

2.3. Comprobamos que se ha creado bien la imagen: `$docker ps -a`

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
evolvedbinary/exist-db	eXist-4.1.0	8aaf9e12f4b3	4 weeks ago	956MB
mysql	mysql	9546ca122d3a	13 months ago	407MB

Ahora podemos acceder a *existDB* en la dirección

`http://localhost:9080`.

En *collections* podemos seleccionar el script .xml creado y subirlo a *eXistDB*, desde donde realizaremos posteriormente las consultas correspondientes.

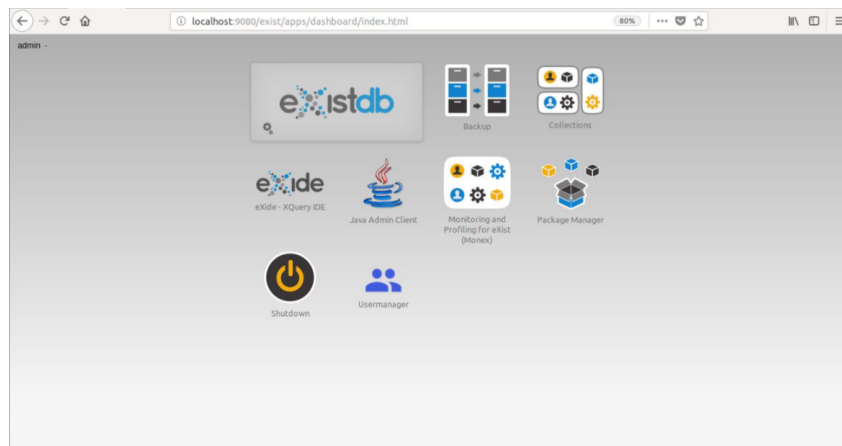


Figura 4: I  
nterfaz ExitsDB

#### 4.1.2. ExitDB

Como gestor de la base de datos.

Es un gestor de base de datos que se puede utilizar sobre bases de datos no relacionales y bases de datos en XML. En nuestro caso, lo utilizaremos para cargar una base de datos en XML. Los problemas principales que hemos encontrado al utilizar este gestor de base de datos a diferencia del resto es que no soporta bases de datos en XML mayores de 4 GB.

Soporta tanto XML, como JSON y HTML.

#### 4.1.3. XQuery

Para realizar las consultas a la base de datos.

A diferencia del resto de gestores de bases de datos eXistDB proporciona su propia forma para realizar las consultas a las bases de datos, XQuery. No lo hemos probado para otro tipo de bases de datos, pero para el caso de XML, nos hemos visto obligado a reducir el tamaño de las bases de datos a tamaños inferiores a 10 Mb para poder realizar consultas a través de XQuery.



## 4.2. Diseño de consultas XML-XQuery

Las consultas que hemos realizado a nuestra base de datos han sido implementadas en XQuery. La idea inicial para realizar las consultas en *XQuery* era la de traducir las consultas que ya teníamos diseñadas.

Por ejemplo:

```
SELECT *  
FROM exon  
WHERE transcript_id = "862540"
```

se podría traducir en XQuery a:

```
for $x in doc("/db/pruebecita.xml")//  
table_data/row/field[@name="transcript_id"]  
where data($x)="862540"  
return $x
```

Pero nos encontramos con un error `GC overhead limit exceeded`. Esto se debía a que habíamos pasado el límite de tiempo que tenía *eXistDB* para realizar una query. Las soluciones que tratamos de abordar para este problema fueron diversas. Desde acceder a los ajustes de *eXistDB* para cambiar la configuración hasta probar una infinidad de consultas. Finalmente, la solución pasó por reducir la base de datos.

### 4.2.1. Reducción bd en XML

Como hemos comentado anteriormente, uno de los dos grandes problemas que nos hemos encontrado al trabajar con *eXistDB* es el tamaño de la base de datos que teníamos. Partimos de la base de datos de *homo\_sapiens* que ocupa cerca de 9GB y nos vimos forzada a reducirla hasta un fichero como el siguiente:

#### ■ Proceso reducción

1. Guardar con `dump` las tablas que nos interesan (transcript, exon y exon\_transcript):

```
mysqldump -u root -p  
bdb_sica transcript exon exon_transcript > bdb_sica_3.sql
```

2. Guardamos el dump generado (`bdb_sica_3.sql`) dentro de una base de datos xml.

```
sudo mysql -u root -p  
bdb_sica_xml < bdb_sica_3.sql
```

3. Incluimos a esta base de datos los datos, de las tablas que no estén no se incluirán los datos.

```
mysqlimport -u root -p
--fields-terminated-by="\t" --fields_escaped_by=\\ bdb_sisa -L *txt
```

4. Ahora volvemos a hacer un dump de la base de datos esta vez con la etiqueta de `-xml` para que nos genere el exportable en XML (`bdb_cisa_3.xml`).

```
sudo mysql -u root -p
bdb_cisa_xml < bdb_cisa_3.xml
```

Este es el proceso para reducir la base de datos a solo tres tablas. Después de probar estos ficheros que aún habiendo sido reducidos seguían ocupando demasiado espacio, nos vimos forzados a recortar la cantidad de datos de cada tabla para poder realizar las queries. Este proceso lo realizamos a mano hasta llegar a reducir el fichero a lo que se ve a continuación:

## ■ Código reducido BD

```
<?xml version="1.0"?>
<mysqldump xmlns:xsi="http://www.w3.org/2001/XMLSchema
-instance">
<database name="bdb_xml">
  <table_structure name="transcript">
    <field Field="transcript_id" Type="int(10)
      unsigned" Null="NO" Key="PRI" Extra="
      auto_increment" Comment="" />
    <field Field="gene_id" Type="int(10) unsigned"
      Null="YES" Key="MUL" Extra="" Comment="" />
    <field Field="biotype" Type="varchar(40)" Null="NO"
      Key="" Extra="" Comment="" />
    <field Field="description" Type="text" Null="YES"
      Key="" Extra="" Comment="" />
  </table_structure>
  <table_data name="transcript">
    <row>
      <field name="transcript_id">862540</field>
      <field name="gene_id">243749</field>
      <field name="biotype">protein_coding</field>
      <field name="description" xsi:nil="true" />
    </row>
    <row>
      <field name="transcript_id">862531</field>
      <field name="gene_id">243689</field>
      <field name="biotype">retained_intron</field>
      <field name="description" xsi:nil="true" />
    </row>
    <row>
      <field name="transcript_id">862532</field>
      <field name="gene_id">243689</field>
      <field name="biotype">retained_intron</field>
      <field name="description" xsi:nil="true" />
    </row>
```

```

</row>
<row>
  <field name="transcript_id">862465</field>
  <field name="gene_id">243679</field>
  <field name="biotype">processed_transcript</field>
  <field name="description" xsi:nil="true" />
</row>
<row>
  <field name="transcript_id">862466</field>
  <field name="gene_id">243679</field>
  <field name="biotype">protein_coding</field>
  <field name="description" xsi:nil="true" />
</row>
<row>
  <field name="transcript_id">862467</field>
  <field name="gene_id">243731</field>
  <field name="biotype">retained_intron</field>
  <field name="description" xsi:nil="true" />
</row>
</table_data>
</database>
</mysqldump><?xml version="1.0"?>
<mysqldump xmlns:xsi="http://www.w3.org/2001/XMLSchema
-instance">
<database name="bdb_xml">
  <table_structure name="transcript">
    <field Field="transcript_id" Type="int(10)
      unsigned" Null="NO" Key="PRI" Extra="
      auto_increment" Comment="" />
    <field Field="gene_id" Type="int(10) unsigned"
      Null="YES" Key="MUL" Extra="" Comment="" />
    <field Field="biotype" Type="varchar(40)" Null="NO
      " Key="" Extra="" Comment="" />
    <field Field="description" Type="text" Null="YES"
      Key="" Extra="" Comment="" />
  </table_structure>
  <table_data name="transcript">
    <row>
      <field name="transcript_id">862540</field>
      <field name="gene_id">243749</field>
      <field name="biotype">protein_coding</field>
      <field name="description" xsi:nil="true" />
    </row>
    <row>
      <field name="transcript_id">862531</field>
      <field name="gene_id">243689</field>
      <field name="biotype">retained_intron</field>

```

```

    <field name="description" xsi:nil="true" />
</row>
<row>
    <field name="transcript_id">862532</field>
    <field name="gene_id">243689</field>
    <field name="biotype">retained_intron</field>
    <field name="description" xsi:nil="true" />

</row>
<row>
    <field name="transcript_id">862465</field>
    <field name="gene_id">243679</field>
    <field name="biotype">processed_transcript</field>
    <field name="description" xsi:nil="true" />
</row>
<row>
    <field name="transcript_id">862466</field>
    <field name="gene_id">243679</field>
    <field name="biotype">protein_coding</field>
    <field name="description" xsi:nil="true" />
</row>
<row>
    <field name="transcript_id">862467</field>
    <field name="gene_id">243731</field>
    <field name="biotype">retained_intron</field>
    <field name="description" xsi:nil="true" />
</row>
</table_data>
</database>
</mysqldump>

```

Una vez reducida, la subimos así a *eXistDB*:

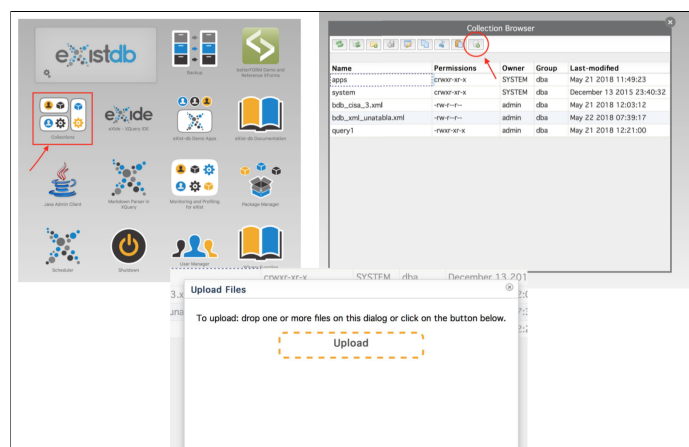


Figura 5: Introducción de bd en eXistDB

#### 4.2.2. Creación de consultas para db reducida

Los principales problemas que tuvimos para realizar las consultas surgieron a raíz del desconocimiento de este lenguaje así como el tamaño de la base de datos que estábamos usando. Como hemos comentado anteriormente en el apartado de reducción de la DB Las queries que hemos realizado son las siguientes:

- Consulta que devuelve el *transcript\_id* dado:

```
for $x in doc("/db/pruebecita.xml")//  
table_data/row/field[@name="transcript_id"]  
where data($x)="862540"  
return $x
```



Figura 6: C  
onsulta XML 1

- Consulta que devuelve el *gene\_id* para un *transcript\_id* dado:

```
for $x in doc("/db/pruebecita.xml")//  
table_data/row/field[@name="transcript_id"]  
where data($x)="862540"  
return $x/../../field[@name="gene_id"]
```

- Consulta que devuelve todos los datos de la tabla para un *transcript\_id* dado:

```
for $x in doc("/db/pruebecita.xml")//  
table_data/row/field[@name="transcript_id"]  
where data($x)="862540"  
return $x/..
```

```
1 xquery version "3.0";
2
3 for $x in doc("/db/pruebecita.xml")//table_data/row/field[@name="transcript_id"]
4 where data($x)="862540"
5 return $x/../../field[@name="gene_id"]
6
7
8
9
```

\_\_new\_\_1

XML Output ☐ Live Preview

```
1 <field name="gene_id">243749</field>
```

Figura 7: Consulta XML 2

```
1 xquery version "3.0";
2
3 for $x in doc("/db/pruebecita.xml")//table_data/row/field[@name="transcript_id"]
4 where data($x)="862540"
5 return $x/..
6
7
8
9
10
```

\_\_new\_\_1

XML Output ☐ Live Preview

```
1 <row>
  <field name="transcript_id">862540</field>
  <field name="gene_id">243749</field>
  <field name="biotype">protein_coding</field>
  <field xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="description" xsi:nil="true"/>
</row>
```

Figura 8: Consulta XML 3

- Consulta que devuelve la información de todas las tablas presentes filtrando por *transcript\_id*

```
for $x in doc("/db/pruebecita.xml")//
table_data/row/field[@name="transcript_id"]
```

```
return $x/..
```

```
1 xquery version "3.0";
2
3 for $x in doc("/db/pruebecita.xml")//table_data/row/field[@name="transcript_id"]
4 return $x/..
5
6
```

\_\_new\_\_1

XML Output ☐ Live Preview

```
1 <row>
  <field name="transcript_id">862540</field>
  <field name="gene_id">243749</field>
  <field name="biotype">protein_coding</field>
  <field xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="description" xsi:nil="true"/>
</row>
2 <row>
  <field name="transcript_id">862531</field>
  <field name="gene_id">243689</field>
  <field name="biotype">retained_intron</field>
  <field xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="description" xsi:nil="true"/>
</row>
3 <row>
  <field name="transcript_id">862532</field>
  <field name="gene_id">243689</field>
  <field name="biotype">retained_intron</field>
  <field xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="description" xsi:nil="true"/>
</row>
4 <row>
  <field name="transcript_id">862465</field>
  <field name="gene_id">243679</field>
  <field name="biotype">processed_transcript</field>
```

Figura 9: Consulta XML 4

- Consulta que devuelve los biotipos en funcion de *gene\_id* :

```
for $x in doc("/db/pruebecita.xml")//
table_data/row/field[@name="gene_id"]
return $x/../../field[@name="biotype"]
```

```
1 xquery version "3.0";
2
3 for $x in doc("/db/pruebecita.xml")//table_data/row/field[@name="gene_id"]
4 return $x/../../field[@name="biotype"]
5
6
```

\_\_new\_\_1

XML Output ☐ Live Preview

```
1 <field name="biotype">protein_coding</field>
2 <field name="biotype">retained_intron</field>
3 <field name="biotype">retained_intron</field>
4 <field name="biotype">processed_transcript</field>
5 <field name="biotype">protein_coding</field>
6 <field name="biotype">retained_intron</field>
```

Figura 10: Consulta XML 5

## 4.3. Conclusión gestor XML

### 4.3.1. Tiempos

- ```
for $x in doc("/db/pruebecita.xml")//  
    table_data/row/field[@name="transcript_id"]  
    where data($x)="862540"  
    return $x
```

2,018s

- ```
for $x in doc("/db/pruebecita.xml")//  
    table_data/row/field[@name="transcript_id"]  
    where data($x)="862540"  
    return $x/../../field[@name="gene_id"]
```

2,022s

- ```
for $x in doc("/db/pruebecita.xml")//  
    table_data/row/field[@name="transcript_id"]  
    return $x/..
```

2,027s

- ```
for $x in doc("/db/pruebecita.xml")//  
    table_data/row/field[@name="gene_id"]  
    return $x/../../field[@name="biotype"]
```

2,025s

- ```
for $x in doc("/db/pruebecita.xml")//  
    table_data/row/field[@name="transcript_id"]  
    where data($x)="862540"  
    return $x/.
```

2,023s

Tras diseñar las bases de datos , podríamos llegar a una conclusión sobre si los datos son más adecuados para el modelo XML o el modelo relacional. Aunque, en nuestro caso, no podemos llegar a una conclusión objetiva puesto que las consultas no son las mismas.

Hay ciertas características que si podemos observar tras experimentar con ambos tipos:



En primer lugar, los datos XML son jerárquicos; mientras que los datos relacionales se representan en un modelo de relaciones lógicas. Además, los datos XML son autodescriptivos; mientras que los datos relacionales, no.

Un documento XML contiene no sólo los datos, sino también la codificación de los datos que explica lo que son. Un solo documento puede contener distintos tipos de datos. En el modelo relacional, el contenido de los datos se define en su definición de columna. Todos los datos de una columna deben tener el mismo tipo de datos.

Tras estas diferencias, hay otros factores que pueden influir en la decisión de qué modelo utilizar: Por ejemplo la máxima flexibilidad frente al máximo rendimiento para la recuperación de datos. Quizás si nos guiamos por estos factores, un máximo rendimiento es más importante en la aplicación que estamos diseñando, por lo que nos seguiríamos decantando por las relacionales.



## 5. Tecnologías utilizadas en SGBD no relacionales y diseño de consultas

### 5.1. Tecnologías utilizadas en SGBD no relacionales

Los servicios de *MySQL* están consumiendo muchos recursos de la máquina en su mayoría de CPU y IO de disco. Para la parte de mongo hemos cogido la BD previamente introducidas en *MariaDB* y las hemos pasado a mongo convirtiéndolas en .csv.

#### 5.1.1. MongoDB

MongoDB es un sistema de gestión de bases de datos *NoSQL* de código abierto y orientado a documentos. Fue desarrollado en 2007 por la compañía 10gen (renombrada actualmente a MongoDB inc), y está disponible para los sistemas operativos Windows, Linux, OS X y Solaris.

A diferencia de los sistemas de datos relacionales, MongoDB no guarda los datos en tablas, sino que en lugar de tablas, guarda los datos en documentos *BSON*. Los documentos *BSON* son similares a los documentos *JSON* pero con un esquema dinámico, con esto se consigue que las integraciones con datos de distintas aplicaciones, sea sencilla y rápida.

Las características principales de *MongoDB* son:

- **Consultas Ad Hoc:** Soporta búsqueda por campos, consultas de rangos y expresiones regulares.
- **Indexación:** Permite crear índices para cualquier campo y también ofrece la posibilidad de crear índices secundarios.
- **Replicación:** La replicación es un mecanismo de seguridad que permite tener acceso al sistema de información en casos de que surja algún problema con el servidor. *MongoDB* soporta el tipo de replicación primario-secundario, este tipo de replicación consiste en que existe un nodo primario y el resto secundarios. El nodo primario puede ejecutar comandos de lectura y escritura, los secundarios replican los datos del primario y solo se pueden usar para lectura o para copias de seguridad. En caso de que el nodo primario caiga, los nodos secundarios tienen la capacidad de elegir a un nuevo nodo primario. A este sistema de nodos primario-secundarios, se le denomina “replica set”.
- **Balanceo de carga:** *MongoDB* ofrece la posibilidad de escalar de forma horizontal empleando el concepto de “shard”. Esto permite que los datos sean distribuidos en distintos servidores, balanceando la carga entre ellos, de manera que ninguno de ellos tenga sobrecarga de datos. De esta manera, se pueden incorporar nuevos servidores proporcionando mucha flexibilidad.

- **Almacenamiento:** Es posible utilizar *MongoDB* como un sistema de archivos, permitiendo utilizar la ventaja de capacidad proporcionada por el balanceo de carga, así como la replicación de datos para el almacenamiento mediante múltiples servidores. Esto permite que los datos puedan ser distribuidos y replicados varias veces, proporcionando un sistema eficiente, con tolerancia a fallos y balanceos de carga.
- **Agregación:** Proporciona un framework de agregación que permite realizar operaciones similares a las operaciones *GROUPBY* de SQL. Además, *MongoDB* también ofrece la función *MapReduce* que puede ser utilizada para el procesamiento por lotes de datos y las operaciones de agregación.
- **Ejecución de *JavaScript* desde el lado del servidor:** Tiene la capacidad de realizar consultas usando JavaScript, enviándolas directamente a la base de datos.

### 5.1.2. JSON

Formato en el que se guardan los datos de las bases de datos.

### 5.1.3. JavaScript

Para hacer las consultas.

## 5.2. Diseño de consultas NOSQL

### 5.2.1. Migración de datos a MongoDB

Para ello, hemos usado un programa que hemos encontrado en el siguiente repositorio de Github:

<https://github.com/lovette/mysql-to-mongo>  
<https://github.com/lovette/mysql-to-mongo>

Dispondremos de las siguientes herramientas:

```
my2mo-fields
my2mo-export
my2mo-import
```

1. **Crear un entorno** donde trabajar cómodamente:

```
$mkdir -p /opt/my2mo/csvdata
$cd /opt/my2mo
```

*(Directorio para hacer migraciones)*

2. Creación del **Schema** (Estructura de la bd sin los datos):

```
$mysqldump --no-data [Base de datos] > [Schema Base de datos].sql
```

### 3. Migración de los campos de las tablas:

Crear la estructura de datos para ser importada a MongoDB:

```
$sudo bash my2mo-fields.sh [Shema Base de datos].sql
```

```
Generating tables and fields from schema.sql...
```

```
...country ...ip2nation ...language
```

```
...offer
```

```
...source ...updater ...updater_country ...updater_offer ...updater_user ...v
```

```
7 fields 2 fields 5 fields
```

```
15 fields 2 fields
```

```
18 fields
```

```
3 fields
```

```
2 fields 13 fields
```

```
4 fields
```

```
Found 10 tables
```

```
Output saved to /srv/mgo
```

```
Tables saved to import.tables Field files saved to fields/*.fields
```

### 4. Creación del archivo de importación:

```
$sudo bash my2mo-export.sh [directorio]/ [BASE DE DATOS]
```

```
Generating SQL to export 10 tables...
```

```
export.sql saved to /srv/mgo
```

```
Data files will be saved to /srv/mgo/csvdata on the
```

```
MySQL server, make sure this directory exists, and is empty
```

### 5. Creación de los **ficheros csv** para la importación

Al ejecutar el siguiente comando importamos los datos de la BD MySQL hacia ficheros csv de cada tabla:

```
$mysql -uroot -p < export.sql
```

### 6. Migración de **csv a MongoDB**:

```
$my2mo-import.sh [directorio] [NOMBRE BASE DE DATOS]
```

```
Importing 10 tables into Mongo database 'toolbox'...
```

```
...country
```

```
...ip2nation
```

```
...language
```

```
...offer
```

```
...source
```

```
...updater
```

```
...updater_country
...updater_offer
...updater_user
...version
Done!
```

7. Comprobamos que se ha importado correctamente:

```
$mongo
MongoDB shell version: 2.4.14 connecting to: test
$ > use toolbox;
switched to db toolbox
$ > show collections;
country ip2nation language
offer
source
updater updater_country updater_offer updater_user version
```

### 5.2.2. Consultas para MongoDB a traves de JavaScript

- Consulta que coge de la bd la tabla *alt\_allele* y devuelve toda la fila en la que la columna *alt\_allele\_id* es 96950

```
> db.alt_allele.find({alt_allele_id: 96950})
```

```
> db.alt_allele.find({alt_allele_id: 96950})
{ "_id" : ObjectId("5b0d7878d512e75c69011a38"), "alt_allele_id" : 96950, "alt_allele_group_id" : 28938, "gene_id" : 194002 }
```

Figura 11: C  
onsulta 1 MongoDB

- Devuelve todos los valores de la tabla *alt\_allele*

```
> db.alt_allele.find()
```

```
> db.alt_allele.find({alt_allele_id: 94325})
{ "_id" : ObjectId("5b0d7878d512e75c69011a4b"), "alt_allele_id" : 94325, "alt_allele_group_id" : 28184, "gene_id" : 193992 }
> db.alt_allele.find()
{ "_id" : ObjectId("5b0d7878d512e75c69011a38"), "alt_allele_id" : 96950, "alt_allele_group_id" : 28938, "gene_id" : 194002 }
{ "_id" : ObjectId("5b0d7878d512e75c69011a39"), "alt_allele_id" : 97653, "alt_allele_group_id" : 29085, "gene_id" : 194006 }
{ "_id" : ObjectId("5b0d7878d512e75c69011a3a"), "alt_allele_id" : 97655, "alt_allele_group_id" : 29086, "gene_id" : 194022 }
{ "_id" : ObjectId("5b0d7878d512e75c69011a3b"), "alt_allele_id" : 97604, "alt_allele_group_id" : 29060, "gene_id" : 193983 }
{ "_id" : ObjectId("5b0d7878d512e75c69011a3c"), "alt_allele_id" : 95711, "alt_allele_group_id" : 28734, "gene_id" : 194043 }
{ "_id" : ObjectId("5b0d7878d512e75c69011a3d"), "alt_allele_id" : 97658, "alt_allele_group_id" : 29087, "gene_id" : 194046 }
{ "_id" : ObjectId("5b0d7878d512e75c69011a3e"), "alt_allele_id" : 100998, "alt_allele_group_id" : 30392, "gene_id" : 194055 }
{ "_id" : ObjectId("5b0d7878d512e75c69011a3f"), "alt_allele_id" : 100063, "alt_allele_group_id" : 29971, "gene_id" : 194047 }
{ "_id" : ObjectId("5b0d7878d512e75c69011a40"), "alt_allele_id" : 94181, "alt_allele_group_id" : 28070, "gene_id" : 194059 }
{ "_id" : ObjectId("5b0d7878d512e75c69011a41"), "alt_allele_id" : 97660, "alt_allele_group_id" : 29088, "gene_id" : 194074 }
{ "_id" : ObjectId("5b0d7878d512e75c69011a42"), "alt_allele_id" : 96606, "alt_allele_group_id" : 28868, "gene_id" : 194114 }
{ "_id" : ObjectId("5b0d7878d512e75c69011a43"), "alt_allele_id" : 97651, "alt_allele_group_id" : 29084, "gene_id" : 193963 }
{ "_id" : ObjectId("5b0d7878d512e75c69011a44"), "alt_allele_id" : 98715, "alt_allele_group_id" : 29370, "gene_id" : 193976 }
{ "_id" : ObjectId("5b0d7878d512e75c69011a45"), "alt_allele_id" : 94179, "alt_allele_group_id" : 28115, "gene_id" : 193975 }
{ "_id" : ObjectId("5b0d7878d512e75c69011a46"), "alt_allele_id" : 95718, "alt_allele_group_id" : 28735, "gene_id" : 194164 }
{ "_id" : ObjectId("5b0d7878d512e75c69011a47"), "alt_allele_id" : 95703, "alt_allele_group_id" : 28733, "gene_id" : 193987 }
{ "_id" : ObjectId("5b0d7878d512e75c69011a48"), "alt_allele_id" : 99250, "alt_allele_group_id" : 29582, "gene_id" : 194169 }
{ "_id" : ObjectId("5b0d7878d512e75c69011a49"), "alt_allele_id" : 95730, "alt_allele_group_id" : 28736, "gene_id" : 194172 }
{ "_id" : ObjectId("5b0d7878d512e75c69011a4a"), "alt_allele_id" : 96259, "alt_allele_group_id" : 28811, "gene_id" : 194129 }
{ "_id" : ObjectId("5b0d7878d512e75c69011a4b"), "alt_allele_id" : 94325, "alt_allele_group_id" : 28184, "gene_id" : 193992 }
Type "it" for more
> db.alt_allele.find({alt_allele_id: 96950})
{ "_id" : ObjectId("5b0d7878d512e75c69011a38"), "alt_allele_id" : 96950, "alt_allele_group_id" : 28938, "gene_id" : 194002 }
```

Figura 12: C  
onsulta 2 MongoDB

- ```
> db.dna.find()
```

Figura 13: Consulta 3 MongoDB

### 5.3.1. Tiempos consultas

## Ventajas de una base de datos NoSQL:

- ### Desventajas de una base de datos NoSQL:

- No todas las bases de datos NoSQL contemplan la atomicidad de las instrucciones y la integridad de los datos.
- Soporte multiplataforma. Aún quedan muchas mejoras en algunos sistemas para que soporten sistemas operativos que no sean Linux.

## **NoSQL vs SQL:** Cuándo utilizar qué tipo de base de datos

- Cuando los datos deben ser consistentes sin dar posibilidad al error utilizar una base de datos relacional. SQL.
- Análisis de grandes cantidades de datos en modo lectura. NoSQL



## 6. Clases Java

### 6.1. Conexión Bases de datos

- Connection Postgree

```
package com.example.demo;
import java.util.ArrayList;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Arrays;
import java.util.Date;
import java.util.List;
import java.util.concurrent.TimeUnit;
import java.sql.*;

public class ConnectionPostgree {

    //Definimos la URL con la que vamos a trabajar

    List<Float> tiempos = new ArrayList<Float>();

    public List<Float> HacerConsulta(String consulta)
    {

        List<String> dbList = Arrays.asList("bdb_cica"
            , "bdb_cisa", "bdb_sica", "bdb_sisa");
        for (int i = 0; i < dbList.size(); i++) {
            String connectionString = "jdbc:postgresql
                ://localhost:5432/" + dbList.get(i) + "?
                user=postgres&password=name&
                autoReconnect=true&useSSL=false";
            Connection connection = null;
            Statement statement = null;
            ResultSet result = null;
            String selectSql = "";

            try {
                //Iniciamos el driver
                Class.forName("org.postgresql.
                    Driver");
                //Abrimos la conexión
                connection = DriverManager.
                    getConnection(connectionString);
```

```

        //Crea un objeto
        SQLServerStatement para enviar
        instrucciones SQL a la base de
        datos.
        statement = connection.
            createStatement();
        //Guardamos la consulta que
        queremos en una variable
        long start = System.
            currentTimeMillis();
        Thread.sleep(2000);
        selectSql = consulta;
        //Ejecuta la instrucción SQL
        especificada y devuelve una sola
        SQLServerResultSet objeto.
        result = statement.executeQuery(
            selectSql);
        while (result.next()) {
        }
        long elapsedTimeMillis = System.
            currentTimeMillis() - start;
        float elapsedTimeSec =
            elapsedTimeMillis / 1000F;
        tiempos.add(elapsedTimeSec);
    }

    catch (Exception e) { e.printStackTrace()
        ;}

    finally {
        if (result != null) try { result.close
            ();} catch (Exception e) {}
        if (statement != null) try { statement
            .close(); } catch (Exception e) {}
        if (connection != null) try {
            connection.close(); } catch (
            Exception e) {}
    }

    }
    return tiempos;
}
}

```

#### ■ Connection MariaDB

```

import java.sql.*;

public class ConnectionMariaDB {

```

```

List<Float> tiempos = new ArrayList<Float>();

public List<Float> HacerConsulta(String consulta) {

    List<String> dbList = Arrays.asList("bdb_cica", "
        bdb_cisa", "bdb_sica", "bdb_sisa");
    for (int i = 0; i < dbList.size(); i++) {

        String connectionString = "jdbc:mysql://
            localhost:3306/" + dbList.get(i) + "?user=root
            &password=name&autoReconnect=true&useSSL=false
            ";
        Connection connection = null;
        Statement statement = null;
        ResultSet result = null;
        String selectSql = "";

        try {
            Class.forName("com.mysql.jdbc.Driver");
            connection = DriverManager.getConnection(
                connectionString);
            statement = connection.createStatement();
            long start = System.currentTimeMillis();
            Thread.sleep(2000);
            selectSql = consulta;
            result = statement.executeQuery(selectSql);
            while (result.next()) {
            }
            long elapsedTimeMillis = System.
                currentTimeMillis() - start;
            float elapsedTimeSec = elapsedTimeMillis /
                1000F;
            tiempos.add(elapsedTimeSec);

        }
        catch (Exception e) { e.printStackTrace();}

        finally {
            if (result != null) try { result.close
                ();} catch (Exception e) {}
            if (statement != null) try { statement
                .close(); } catch (Exception e) {}
            if (connection != null) try {
                connection.close(); } catch (
                Exception e) {}
        }
    }
}

```

```

    }
    return tiempos;
}
}

```

- Connection MySQL (Con Docker, en puerto 3307)

```

import java.sql.*;

public class ConnectionMysql {

    List<Float> tiempos =new ArrayList<>() ;

    public List<Float> HacerConsulta(String consulta) {

        List<String> dbList = Arrays.asList("bdb_cica"
            , "bdb_cisa", "bdb_sica", "bdb_sisa");

        for(int i=0;i<dbList.size();i++){
            String connectionString = "jdbc:mysql://
                localhost:3307/"+dbList.get(i)+"?user=
                root&password=name&autoReconnect=true&
                useSSL=false" ;
            Connection connection = null;
            Statement statement = null;
            ResultSet result = null;
            String selectSql="";

            try{
                Class.forName("com.mysql.jdbc.Driver");
                connection = DriverManager.getConnection(
                    connectionString);
                statement = connection.createStatement();
                long start = System.currentTimeMillis();
                Thread.sleep(2000);
                selectSql = consulta;
                result = statement.executeQuery(selectSql);
                while (result.next()) {}
                long elapsedTimeMillis = System.
                    currentTimeMillis() - start;
                float elapsedTimeSec = elapsedTimeMillis/1000F;
                tiempos.add(elapsedTimeSec);

            }
            catch (Exception e) { e.printStackTrace();}

            finally {

```

```

        if (result != null) try { result.close();}
            catch (Exception e) {}
        if (statement != null) try { statement.
            close(); } catch (Exception e) {}
        if (connection != null) try { connection.
            close(); } catch (Exception e) {}
    }
}
return tiempos;
}
}

```

## ■ Connection XML

```

import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import javax.xml.transform.OutputKeys;

public class ConnectionXML {

    //Definimos la URL con la que vamos a trabajar
    String connectionString = "xmldb:exist://
        localhost:9080/exist/xmlrpc" ;

    //Inicializamos los parámetros necesarios
    private ResourceSet result;
    String selectSql="";
    List<Float> tiempos = new ArrayList<Float>();
    public List<Float> HacerConsulta(String consulta)
    {

        try {
            //Iniciamos el driver
            Class<?> cl = Class.forName("org.exist.
                xmldb.DatabaseImpl");
            //Abrimos la conexión
            Database database = (Database) cl.
                newInstance();
            database.setProperty("create-database", "
                true");
            DatabaseManager.registerDatabase(database)
                ;

            System.out.println(consulta);

            // get root-collection

```

```

        Collection col =
            DatabaseManager.getCollection(
                connectionString + XmlldbURI.
                ROOT_COLLECTION, "admin", "name"
            );
        // get query-service
        EXistXQueryService service =
            (EXistXQueryService) col.
                getService("XQueryService", "1.0
                ");

        // set pretty-printing on
        service.setProperty(OutputKeys.INDENT, "
            yes");
        service.setProperty(OutputKeys.ENCODING, "
            UTF-8");

        CompiledExpression compiled = service.
            compile(consulta);
        //Guardamos la consulta que queremos en
            una variable
        long start = System.currentTimeMillis();
        Thread.sleep(2000);
        result = service.execute(compiled);

        long elapsedTimeMillis = System.
            currentTimeMillis() - start;
        float elapsedTimeSec = elapsedTimeMillis /
            1000F;
        tiempos.add(elapsedTimeSec);
    } catch ( Exception e ) {
        e.printStackTrace();
    }

    return tiempos;
}
}

```

#### ■ Connection MongoDB CAMBIAR

```

import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

import javax.xml.transform.OutputKeys;

public class ConnectionXML {

```

```

//Definimos la URL con la que vamos a trabajar
String connectionString = "xmldb:exist://
    localhost:9080/exist/xmlrpc" ;

//Inicializamos los parámetros necesarios
private ResourceSet result;
String selectSql="";
List<Float> tiempos = new ArrayList<Float>();
public List<Float> HacerConsulta(String consulta)
{

    try {
        //Iniciamos el driver
        Class<?> cl = Class.forName("org.exist.
            xmldb.DatabaseImpl");
        //Abrimos la conexión
        Database database = (Database) cl.
            newInstance();
        database.setProperty("create-database", "
            true");
        DatabaseManager.registerDatabase(database)
            ;

        System.out.println(consulta);

        // get root-collection
        Collection col =
            DatabaseManager.getCollection(
                connectionString + XmldbURI.
                    ROOT_COLLECTION, "admin", "name"
            );
        // get query-service
        EXistXQueryService service =
            (EXistXQueryService) col.
                getService("XQueryService", "1.0
                    ");

        // set pretty-printing on
        service.setProperty(OutputKeys.INDENT, "
            yes");
        service.setProperty(OutputKeys.ENCODING, "
            UTF-8");

        CompiledExpression compiled = service.
            compile(consulta);
        //Guardamos la consulta que queremos en
            una variable
    }
}

```

```

        long start = System.currentTimeMillis();
        Thread.sleep(2000);
        result = service.execute(compiled);

        long elapsedTimeMillis = System.
            currentTimeMillis() - start;
        float elapsedTimeSec = elapsedTimeMillis /
            1000F;
        tiempos.add(elapsedTimeSec);
    } catch ( Exception e ) {
        e.printStackTrace();
    }

    return tiempos;
}
}

```

## 6.2. Conexión BD-Interfaz

- PetitionController

```

package com.example.demo;
import com.example.demo.ConnectionPostgree;

import org.json.*;

import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Map.Entry;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.
    RequestBody;
import org.springframework.web.bind.annotation.
    RequestMapping;
import org.springframework.web.bind.annotation.
    RequestMethod;
import org.springframework.web.bind.annotation.
   .ResponseBody;
import org.springframework.web.bind.annotation.
    RestController;

```



```

@RestController
public class PetitionController {

    @RequestMapping(value="/api/hi", method =
        RequestMethod.GET)
    @ResponseBody
    public String getFoosBySimplePath() {
        return "Get some Foos";
    }

    @RequestMapping(value="/api/hi", method =
        RequestMethod.POST)
    @ResponseBody
    public List<?> getFoosBySimplePath1(@RequestBody
        String body) throws JSONException {

        JSONArray jsonarray = new JSONArray(body);

        //jsonarray guarda una petición en cada pos (2
        //peticiones-> length2,...)
        Map <String, String> consultaGestor = new HashMap<
            String, String>();
        String groupBy="";
        for (int i = 0; i < jsonarray.length(); i++) {

            JSONObject petition = jsonarray.getJSONObject(i)
                ;

            int num_col = petition.getInt("colNum");
            int num_tab = petition.getInt("tabNum");
            int num_cond = petition.getInt("condNum");
            if(petition.has("groupBy")) groupBy = petition.
                getString("groupBy");
            String gestor = petition.getString("bd");

            List<String> listCol = new ArrayList<String>();
            for (int col=1; col<=num_col; col++) {
                String columna = petition.getString("nom" +
                    col);
                listCol.add(columna);
            }
        }
    }
}

```

```

List<String> listTab = new ArrayList<String>();
for (int tab=1; tab<=num_tab; tab++) {
    String tabla = petition.getString("tab" + tab)
    ;
    listTab.add(tabla);
}

List<String> listCond = new ArrayList<String>();
for (int cond=1; cond<=num_cond; cond++) {
    String condicion = petition.getString("cond" +
        cond);
    listCond.add(condicion);
}

System.out.println("JSONOBJECT " + i + ":" +
    petition + "\n" );
System.out.println("josnobj " + i + " NUM TABLAS
    : " + num_tab + " NUM COL: " + num_col + "
    NUM COND: " + num_cond + "\n");

System.out.println("columnas: " + listCol + "\n"
    );
System.out.println("tablas: " + listTab + "\n");
System.out.println("condiciones: " + listCond +
    "\n");

//CREAR CONSULTA

//Lista columnas
String select="";
for(int s=0; s<num_col;s++) {
    if(s==num_col-1) select = select + listCol.get
        (s);
    else select = select + listCol.get(s) + ", ";
}

//Lista tablas
String from="";
for(int f=0; f<num_tab; f++) {
    if(f==num_tab-1) from = from + listTab.get(f);
    else from = from + listTab.get(f) + ", ";
}

//Lista condiciones
String where="";
for(int w=0; w<num_cond; w++) {
    if (w==num_cond-1)
        where = where + listCond.get(w);
}

```

```

        else
            where = where + listCond.get(w) + " AND ";
    }

    //Crear consulta
    String consulta="";

    //Consulta Mongo db (no relational)
    if(gestor.equals("mongodb")) {
        consulta = "db." + from + ".find({" + where + "
            });";
        consultaGestor.put(consulta, gestor);
    }
    //Consulta xml

    else if(gestor.equals("XML")) {
        String consulta4 = "for $x in doc(\"/db/
            pruebecita.xml\")//\n" +
            "    table_data/row/field[@name=\"
            transcript_id\"]\n" +
            "    where data($x)=\"862540\"\n" +
            "    return $x/.";
        String consulta3 = "for $x in doc(\"/db/
            pruebecita.xml\")//\n" +
            "table_data/row/field[@name=\"gene_id\"]\n"
            +
            "return $x/../../field[@name=\"biotype\"]";
        String consulta2 = "for $x in doc(\"/db/
            pruebecita.xml\")//\n" +
            "table_data/row/field[@name=\"transcript_id
            \"]\n" +
            "return $x/..";
        String consulta1 = "for $x in doc(\"/db/
            pruebecita.xml\")//table_data/row/field[@name
            =\"transcript_id\"]\n where data($x)
            =\"862540\" +
            \"return $x/../../field[@name=\"gene_id\"]";
        consulta = "for $x in doc(\"/db/pruebecita.xml
            \")//table_data/row/field[@name=\"gene_id\"]\n
            " +
            "return $x/../../field[@name=\"biotype\"]";
        consultaGestor.put(consulta4, gestor);
    }

    //Consulta relational
    else {
        consulta = "SELECT " + select + " FROM " +
            from + " WHERE " + where;
    }

```

```

        if(!groupBy.equals("")) consulta = consulta +
            " GROUP BY " + groupBy + ";";
        else consulta = consulta + ";";
        consultaGestor.put(consulta, gestor);
    }
}

List<Object> tiemposConsulta = new ArrayList<Object>
    >();

for(Entry<String, String> entry: consultaGestor.
    entrySet()) {

    System.out.println("KEY:" + entry.getKey() + "\n");
    System.out.println("VALUE" + entry.getValue()
        + "\n");

    if(entry.getValue().equals("mongodb")) {
        ConnectionNoSQL cnn = new ConnectionNoSQL
            ();
        tiemposConsulta.addAll(cnn.HacerConsulta(
            entry.getKey()));
        System.out.println("Case nosql: " + entry.
            getValue() + "\n");
    }
    else if (entry.getValue().equals("mariadb")) {
        ConnectionMariaDB cnn_mdb = new
            ConnectionMariaDB();
        List<Float> cnn = new ArrayList<>();
        cnn.addAll(cnn_mdb.HacerConsulta(entry.getKey())
            );
        String petit = "\nBDB_CICA: " + cnn.get(0)+ "\n\
            nBDB_CISA: "+ cnn.get(1) + "\nBDB_SICA: "+cnn.
            get(2)
            + "\nBDB_SISA: "+ cnn.get(3);
        tiemposConsulta.add(petit);
        System.out.println("Case mariadb" + entry.
            getValue()+"\n"+ petit + "\n");

    }
    else if (entry.getValue().equals("psql")){
        ConnectionPostgree cnn_ps = new
            ConnectionPostgree();
        List<Float> cnn = new ArrayList<>();
        cnn.addAll(cnn_ps.HacerConsulta(entry.getKey()))
            ;
    }
}

```

```

String petit = "\nBDB_CICA: " + cnn.get(0)+ "\nBDB_CISA: "+ cnn.get(1) + "\nBDB_SICA: "+cnn.get(2)
            + "\nBDB_SISA: "+ cnn.get(3);
tiemposConsulta.add(petit);
System.out.println("case psql : "+ entry.getValue()+ petit + "\n");
}
else if (entry.getValue().equals("mysql")){
    ConnectionMysql cnn_mys = new ConnectionMysql();
    List<Float> cnn = new ArrayList<>();
    cnn.addAll(cnn_mys.HacerConsulta(entry.getKey()));
    String petit = "\n BDB_CICA: " + cnn.get(0)+ "\n BDB_CISA: "+ cnn.get(1) + "\n BDB_SICA: "+ cnn.get(2)
            + "\n BDB_SISA: "+ cnn.get(3);
    tiemposConsulta.add(petit);
    System.out.println("case mysql: " + entry.getValue()+ petit + "\n");
}
else if (entry.getValue().equals("XML")){
    ConnectionXML cnn_xml = new ConnectionXML();
    tiemposConsulta.addAll(cnn_xml.HacerConsulta(entry.getKey()));
    System.out.println("case XML: " + entry.getValue() + "\n");
}

else{
    System.out.println("no ha funcionao");
    System.out.println(entry.getValue());
}

}
return tiemposConsulta;
}
}

```



## 7. Interfaz gráfica

Nuestra interfaz ha sido desarrollada en AngularJS, un framework de JavaScript. Es una aplicaciones web de una sola página. Sigue el Modelo Vista Controlador (MVC).

### 7.1. Tecnologías usadas

#### 7.1.1. Angular JS

*AngularJS* es un framework de *JavaScript* de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo Vista Controlador (MVC), en un esfuerzo para hacer que el desarrollo y las pruebas sean más fáciles.

Nos ha aportado una serie de funcionalidades extra a nuestro proyecto que nos han facilitado mucho el trabajar con los datos y realizar las peticiones.

#### 7.1.2. JavaScript

*JavaScript* lo hemos utilizado para implementar las diferentes funcionalidades de la interfaz.

#### 7.1.3. HTML

*HTML* nos ha servido para generar la estructura visible de nuestra página web.

#### 7.1.4. CSS

*CSS* aunque poco, porque hemos usado principalmente *Bootstrap* (una librería con diseños de CSS ya hechos), también los hemos aplicado directamente para modificar levemente algún diseño.

## 7.2. Diseño interfaz

En primer lugar ha sido necesario diseñar una **primera aproximación** para la interfaz, tenía el siguiente aspecto:

The screenshot shows a web application titled 'Gestor de peticiones'. It has two main sections: 'Nueva búsqueda' and 'Actualizar búsqueda'. Both sections have input fields for 'Nombre organismo' (with 'Pinus' entered), 'Tipo de búsqueda' (with 'Normal' selected), 'Búsqueda extendida', and 'Dominio de búsqueda' (with 'Assembly', 'Sequence', and 'Counting set' selected). There are also radio buttons for 'Gestor de BD' (MariaDB, Postgres, MySQL) and 'Tipo de búsqueda' (MariaDB, Postgres, MySQL). Below these are 'Crear' and 'Actualizar' buttons. A green message box says 'Petición eliminada'. At the bottom, there is a table titled 'Datos introducidos' with columns: Nombre, Tipo, BD, Búsqueda extendida, Dominio de búsqueda, and Acciones. The table has two rows of data. Below the table are buttons for 'Realizar búsqueda' and 'Limpiar'.

Nombre	Tipo	BD	Búsqueda extendida	Dominio de búsqueda	Acciones
Pinus	Optimizada	MySQL	Pinaster	Non-Coding	Editar Borrar
Pinus	Optimizada	MariaDB	Pinaster	Non-Coding	Editar Borrar

Figura 14: Primer prototipo interfaz

Conforme hemos ido avanzando en el proyecto, hemos visto posibles mejoras que podríamos hacerle a la interfaz y la hemos seguido cambiando hasta tener el siguiente resultado:

The screenshot shows a web application titled 'Encuentra como hacer la mejor petición'. It has a header with 'Home' and 'Buscador' links. The main content area has a title 'Encuentra como hacer la mejor petición' and a description. Below the description is a 'Buscador' button. The application is divided into three main sections: 'Columnas', 'Tablas', and 'Condiciones'. Each section has input fields for 'Número de [columnas/tablas/condiciones]' and 'Nombre de [columna/tabla/condición]'. There are also buttons for 'Group by' and 'Crear consulta'. Below these sections is a table titled 'Datos introducidos' with columns: Numero de Columnas, Columnas, Numero de tablas, Tablas, Numero de condiciones, Condiciones, and Group by. At the bottom are buttons for 'Send petitions' and 'Clean'.

Numero de Columnas	Columnas	Numero de tablas	Tablas	Numero de condiciones	Condiciones	Group by
--------------------	----------	------------------	--------	-----------------------	-------------	----------

Figura 15: Prototipo final interfaz



Esta interfaz ha sido diseñada utilizando **Angular 4** y pretende facilitar el uso de nuestro programa, así como, gestionar una cola de búsqueda en la que podemos ir diciendo las columnas en las que queremos buscar, las tablas que queremos buscar, las condiciones en las que queremos buscar el group By. Esto genera una lista de peticiones (que se irán visualizando en la tabla inferior) y son enviadas a través de un método post hacia el **backEnd**.

Después de realizar la consulta correspondiente, nos devolverá por pantalla el tiempo que ha tardado cada una de las consultas creadas.

El código de la implementación de la interfaz está adjuntado dentro del proyecto. Destacar los scripts *app.component.html* y *app.component.ts*, donde podemos encontrar el código *HTML* y *JavaScript* implementado en la interfaz respectivamente.

Datos introducidos							
Numero de Columnas	Columnas	Numero de tablas	Tablas	Numero de condiciones	Condiciones	Group by	Tiempos
1	*	1	alt_allele	1	gene_id=258544		BDB_CICA: 2.334 BDB_CISA: 2.108 BDB_SICA: 2.233 BDB_SISA: 2.013
							<div>Editar</div> <div>Borrar</div>
1	*	1	alt_allele	1	gene_id=258545		BDB_CICA: 2.001 BDB_CISA: 2.001 BDB_SICA: 2.002 BDB_SISA: 2.002
							<div>Editar</div> <div>Borrar</div>
1	*	1	alt_allele	1	gene_id=258546		BDB_CICA: 2.471 BDB_CISA: 2.137 BDB_SICA: 2.001 BDB_SISA: 2.046
							<div>Editar</div> <div>Borrar</div>
<div>Send petitions</div> <div>Clean</div>							

Figura 16: Tiempos consultas para las 4 DB



## 8. Integración Back-end Front-end

Hasta este momento solo teníamos una interfaz cuyos datos no llegaban a ningún sitio, y un *back – end* donde había que meter los datos a mano, y estos tampoco eran devueltos más que a la pantalla de la consola.

Con esta integración podremos escribir datos en la interfaz, que llegarán al código, este se conectará con el gestor correspondiente, y volveremos a ver sus resultados devueltos en la interfaz. Así, podremos tratar únicamente con la interfaz web, siendo el código una caja negra.

### 8.1. Tecnologías usadas

#### 8.1.1. Spring-Boot 2.0.2

*Spring – Boot* lo hemos utilizado para ejecutar nuestra aplicación y como una aplicación de *Spring* en la que tenemos ya integrando la interfaz y el servidor

#### 8.1.2. Spring Tool Suite 3.8.1

*Spring – Tool – Suite* lo hemos utilizado como herramienta para realizar la integración de ambas partes del proyecto.

#### 8.1.3. Java 1.8

Lenguaje del *Back – End*. En el hemos realizado el código que realiza las conexiones *Front – End-Back – end*.

#### 8.1.4. Maven

Contiene todas las librerías y dependencias en un archivo *pom.xml*, permitiéndonos exportar el proyecto, siendo usable sin la consecuente instalación de estas. Las dependencias requeridas han sido:

- *org.exist – db*
- *org.springframework.boot*
- *org.json*
- *org.slf4j*
- *postgresql*
- *mysql*
- *org.mongodb*
- *com.google.code.gson*

## 8.2. Proceso integración

Partiendo de que tenemos que tener instalado **Node.js** y **@angular/cli**. Pasamos a hacer el setUp del proyecto de Spring. En primer lugar, tenemos comprobar en el script pom.xml, que tenemos la siguiente dependencia.

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

En el proyecto vamos a crear un controlador muy simple, en

```
/source/main/java/com/example/demo
```

```
.

package com.javasampleapproach.restful.controller;

import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class WebRestController {

@RequestMapping("/api/hi")
public String hi() {
return "Hello World from Restful API";
}
}
```

Después de esto, y teniendo nuestro proyecto de angular hecho, pasaremos a integrarlo. Para ello solo tendremos que abrir **SpringToolSuite** y ir a **Import // General // Projects from Folder or Archive** y aceptar.

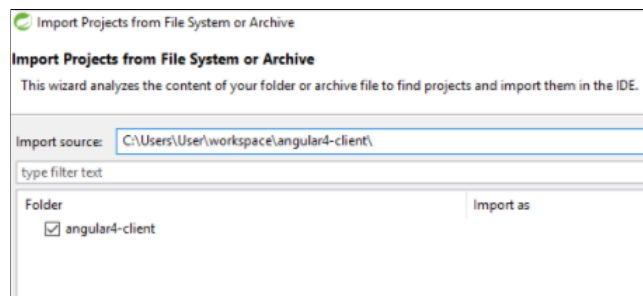


Figura 17: Importación Angular

Con esto ya tendremos el proyecto de **Angular** importado, teniendo algo tal que así:

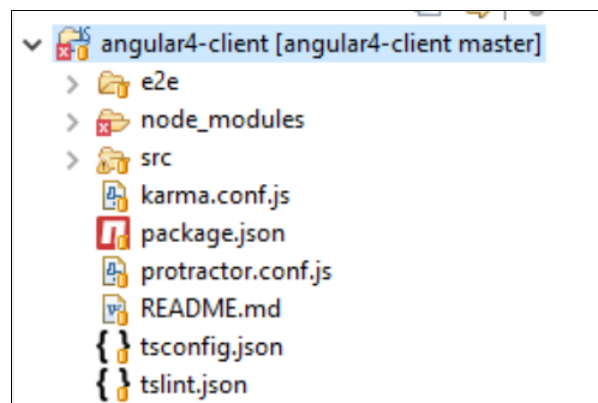


Figura 18: Estructura de carpetas Spring-boot

Lo siguiente que vamos a hacer será quitar la carpeta de *node\_modules* ya que no es necesaria ahora. Para ello, vamos a las **Propiedades** del proyecto y elegimos **Resource // Resource // Filter**. Escribimos lo siguiente:

Ahora ya habremos removido **Node\_modules** del **SpringToolSuite**.

Llegados a este punto tenemos un proyecto de **Spring** en el que tenemos el Back-End, y otro proyecto de **Angular** ya importado en el que tenemos implementado el Front-End. Que trabajan independientemente en los puertos 8080 y 4200.

El objetivo ahora es de establecer una dirección para el cliente y que las peticiones al servidor vayan a diferentes direcciones /api.

Para ello primero tenemos que crear en nuestro proyecto de **Angular**. Un archivo llamado **proxy.config.json** con lo siguiente.

```
{
  "/api": {
    "target": "http://localhost:8080",
    "secure": false
  }
}
```

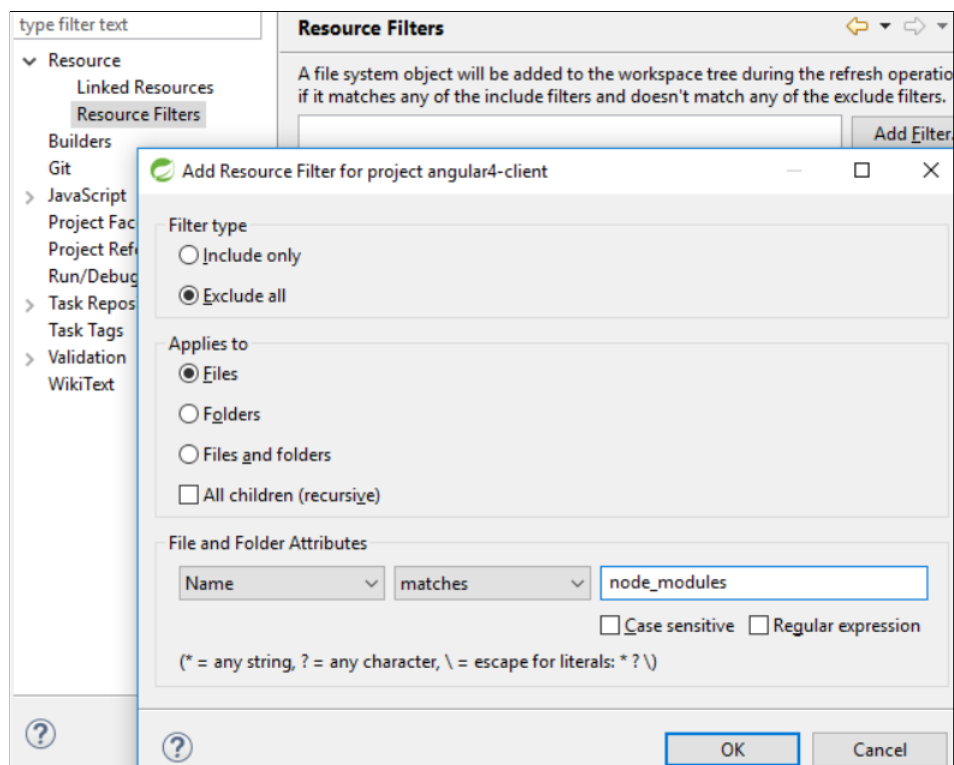


Figura 19: Eliminación node-modules

```
}
}
```

Además vamos a editar el fichero **package.json**, dejándolo tal que así:

```
...
"scripts": {
  "ng": "ng",
  "start": "ng serve --proxy-config proxy.conf.json",
  "build": "ng build",
  "test": "ng test",
  "lint": "ng lint",
  "e2e": "ng e2e"
},
...
```

Construimos y corremos el proyecto **RestfulService** con SpringBoot en el puerto 8080. Y corremos el cliente en el puerto 4200. En este momento, haciendo una petición a

`http://localhost:4200/api/hi`

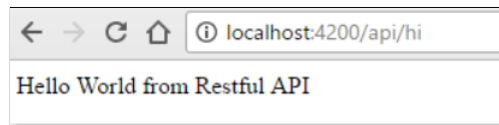


Figura 20: Primera prueba backend en integración

obtenemos el siguiente resultado:

Para final solo queda ejecutar dentro de la carpeta de la interfaz el comando

```
ng build -prod
```

Obtenemos como resultado una carpeta que se llama **dist**. Copiando los archivos que aparezcan en esta carpeta dentro de la ruta del servidor **/src/main/resources/static**. Ya podemos contruir y ejecutar nuestro **SpringBoot server**.

- Build:

```
mvn clean install
```

Es necesario volver a ejecutar este comando y copiar la carpeta **dist** cada vez que queramos realizar un cambio en la interfaz.

- Run:

```
mvn spring-boot:run
```

Para hacer cambios en el backEnd basta con volver a correr la app.

Como resultado ya podremos ejecutar nuestro servidor en la dirección **http://localhost:8080/api/hi** tendremos el servidor. Y en la dirección **http://localhost:8080** tendremos la interfaz.





## 9. Conclusiones



## 10. GitHub

<https://github.com/nairachiclana/Gestor-BDB>

## 11. Webgrafía