

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE
INGENIERÍA DE INFORMÁTICA



UNIVERSIDAD DE MÁLAGA

COMPARACIÓN DE TIEMPOS DE EJECUCIÓN DE CONSULTAS EN DIFERENTES
GESTORES DE BASES DATOS RELACIONALES Y NO RELACIONALES

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:

Naira Chiclana García

José Rodríguez Maldonado

Paloma Domínguez Sánchez

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.

Abstract

| | |
|--|------------|
| Resumen | iii |
| Abstract | v |
| Índice | vi |
| Índice de Tablas | ix |
| Índice de Figuras | xi |
| | |
| 1 Definición inicial del proyecto | 17 |
| 2 Despliegue en 3 SGBDs | 18 |
| 3 Diseño de consultas | 25 |
| | |
| 4 Versiones optimizadas | 28 |
| 5 Despliegue en SGBD XML | 32 |
| 6 Desarrollo Front-End y Back-End | 40 |
| 7 Desarrollo del Back-End | 45 |
| 8 Flujo de funcionamiento | 47 |
| 9 Desarrollo en SGBD NoSQL | 48 |
| | |
| Referencias | 33 |

ÍNDICE DE TABLAS

Tabla 1–1. Tiempos de ejecución de las consultas

32

ÍNDICE DE FIGURAS

Tabla 1–1. Relaciones entre tablas Homo Sapiens

25

1. DEFINICIÓN INICIAL DEL PROYECTO

En este apartado se definen sobre qué Base de Datos Biológica se va a trabajar. Concretamente se ha escogido la base de datos Ensembl.

Ensembl es una base de datos de genomas de metazoos permanentemente actualizada gracias al proyecto común desarrollado por [EMBL - EBI](#) y [Sanger Institute](#)

Su base de datos da acceso gratuito a todo el genoma humano que actualmente es de dominio público y a una gran colección de genes de diferentes especies y su "anotaciones". Además de la descripción de las características identificadas en el DNA reflejada en las llamadas anotaciones, también describe otras características interesantes como SNPs, repeats y homologías.

A parte del acceso a esta gran base de datos, permite una utilización del software de su proyecto para analizar y extraer la información de interés.

2. DESPLIEGUE EN 3 SGBD

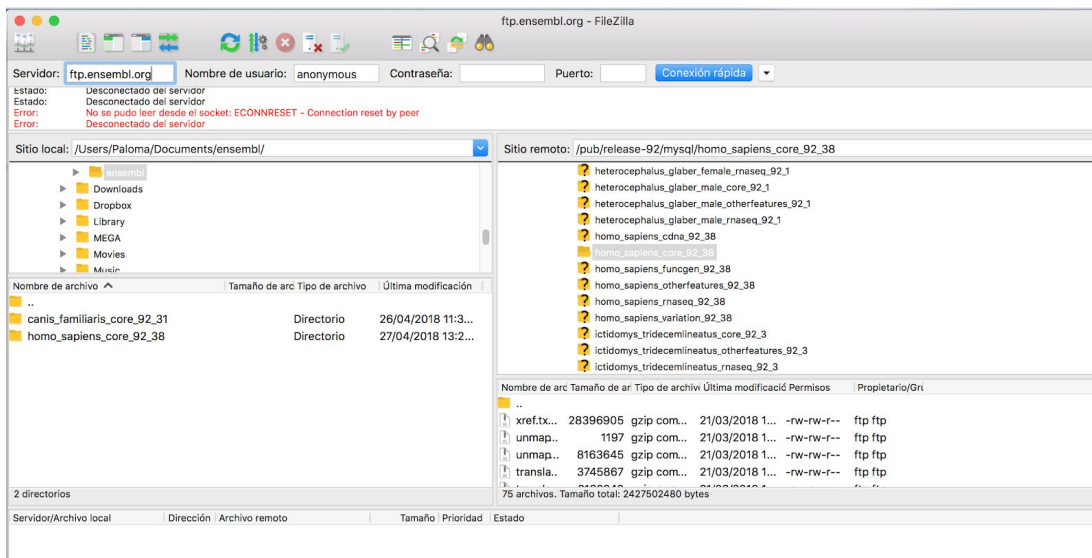
En este apartado se explica los sistemas gestores de Bases de Datos escogidos, el proceso de instalación de cada uno de ellos y su puesta en marcha de la base de datos Ensembl (sin índices, ni optimizaciones):

En un primer barrido se escogieron estos tres gestores:

- MySQL
- PostgreSQL
- MariaDB

Debido a dificultades técnicas decidimos cambiar Oracle por PostgreSQL.

A continuación se explica el proceso de instalación de cada gestor y puesta en marcha de la base de datos: En primer lugar, descargamos la base de datos Ensembl (<http://www.ensembl.org/info/data/ftp/index.html>) y el organismo con el que decidimos trabajar: homo_sapiens_core_92_38. Para ello hemos utilizado FileZilla:



En segundo lugar creamos cada uno un usuario y contraseña para usar el gestor que le correspondiese. Por último se describe paso a paso el procedimiento de la puesta en marcha de la base de datos en cada gestor individualmente:

1. MariaDB (Paloma Dominguez)

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.

- 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)]>
```

- Ver base de datos existentes

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

```
+-----+
| Database      |
+-----+
| information_schema |
| mysql          |
| performance_schema |
| test           |
+-----+
4 rows in set (0.00 sec)
```

- Crear nueva base de datos

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

```
Query OK, 1 row affected (0.00 sec)
```

- Crear nuevo usuario y dar permisos

```
MariaDB [(none)]> CREATE USER 'paloma'@'localhost' IDENTIFIED BY 'palomag1';  
Query OK, 0 rows affected (0.01 sec)
```

```
MariaDB [(none)]> GRANT SELECT,INSERT,DROP,CREATE ON refseq.* TO 'paloma'@'localhost' IDENTIFIED BY 'palomag1';  
Query OK, 0 rows affected (0.01 sec)
```

- Insertar tablas
 - Dentro del directorio donde se hayan guardado las tablas (acceder con el cd y la ruta) con ls -la podemos ver los archivos existentes y vemos que están todos en formato gz.
 - Para descomprimirlos
gunzip *
- Volcamos el archivo con todas las tablas en la base de datos 'bdb', para la creación de las tablas hemos escogido el archivo .sql del homo sapiens.

```
mysql -u root -p bdb < homo_sapiens_core_91_38.sql
```

Comprobamos que se ha creado:

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

```
+-----+  
| Database |  
+-----+  
| bdb      |  
| homo_sapiens_core_91_38 |  
| information_schema |  
| mysql    |  
| performance_schema |  
| test     |  
+-----+
```

- Entramos dentro de la base de datos 'bdb' y comprobamos que se han creado las tablas correctamente

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

```
MariaDB [homo_sapiens_core_91_38]> show tables;
```

```

+-----+
| Tables_in_homo_sapiens_core_91_38 |
+-----+
| alt_allele |
| alt_allele_attrib |
| .....
+-----+

```

- Ver datos de alguna de las tablas

MariaDB [homo_sapiens_core_91_38]> describe alt_allele;

- Cargar los archivos .txt en la estructura creada, desde el terminal y dentro del directorio donde se encuentran los archivos

```

$ mysqlimport -u root -p --fields-terminated-by='\t' --fields_escaped_by=\\
homo_sapiens_core_91_38 -L *.txt

```

2. MySQL (Naira Chiclana)

Los comandos usados y resultados son los mismos que en el apartado anterior.

```

$mysql -u root -p
$create database bdb;
$mysql> show databases;
$mysql -u root -p bdb< homo_sapiens_core_92_38.sql
$mysql[homo_sapiens_core_91_38]> show tables;
$mysqlimport -u root -p --fields-terminated-by='\t' --fields_escaped_by=\\ -L bdb *.txt

```

3. PSQL (José Rodríguez)

Para poder introducir las bases de datos, primero he creado las bases de datos y poblarlas en otro gestor, en este caso he utilizado MariaDB. Y de ahí lo he exportado e importado dentro de Postgres.

Para la crear y poblar las bases de datos he seguido la información proporcionada por mis compañeras.

Partimos de la base de que tenemos las bases de datos ya creadas y pobladas en MariaDB. De ahí lo que vamos a hacer ahora será exportar dichas bases de datos 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 se debería poder importar bien dentro de postgres, pero daba errores. Todos, errores de sintaxis puesto que no había traducido bien las sentencias.

Para arreglar este fallo, he recurrido al uso de un script de python que lo que hacía era cambiar la sintaxis de MySQL/MariaDB a PSQL.

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

Con todo esto todavía seguíamos encontrando algún fallo a la hora de importar las BD dentro de PSQL. Al importar dentro de PSQL ahora 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 se ha podido solucionar este error ha sido cambiando los tipos de las variables de la base de datos y 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, volvemos a realizar todo el proceso realizado anteriormente y finalmente podemos crear las bases de datos:

```
create database <nom_database>;
```

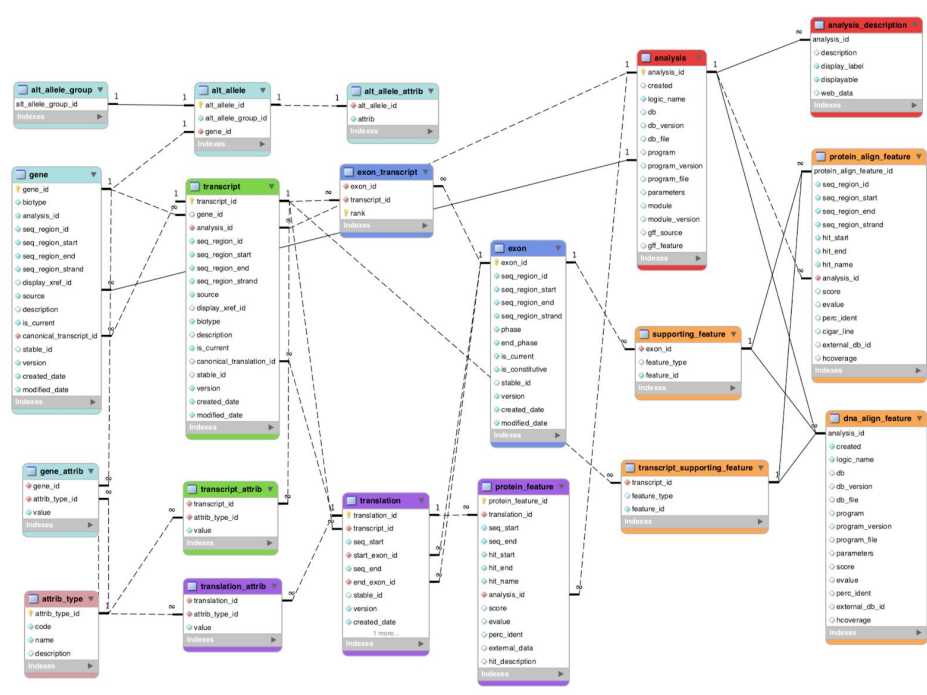
Y importar los datos dentro de ellas:

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

3. DISEÑO DE CONSULTAS

En esta sección se trabaja con el diseño 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, donde a partir de ella se han diseñado las consultas.



Lista de consultas:

→ 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 WHERE  
  }
```

Otros biotypes:

```
biotype=snRNA  
biotype=proteinCoding  
biotype=misc_RNA
```

❖ 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%';
```

→ 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 terminacion y biotipo.

```
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

```
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
```

4. VERSIONES OPTIMIZADAS

En esta sección del documento 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 → Sin índices ni almacenamiento (Sin índices + Motor de búsqueda MyISAM)

V2 → Con índices pero sin almacenamiento (Índices + motor búsqueda MyISAM)

V3 → Sin índices pero con almacenamiento (Sin índices + Motor de búsqueda InnoDB)

V4 → Con índices y con almacenamiento (Índices + motor búsqueda InnoDB)

1. MOTORES DE ALMACENAMIENTO

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. 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:

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*.

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, cambiaremos el motor a InnoDB, para ello usaremos la clase en java que cambia todas las tablas de la base de datos seleccionada y aplicaremos la consulta:

```
selectSql = "ALTER TABLE " + table + " ENGINE=InnoDB";  
statement = connection.createStatement();  
result = statement.executeQuery(selectSql);
```

```
try {  
    Class.forName("com.mysql.jdbc.Driver"); //Class.forName("org.postgresql.Driver");  
    connection = DriverManager.getConnection(connectionString);  
    statement = connection.createStatement();  
  
    //Guarda todos los nombres de las tablas para aplicarle algun cambio  
    selectSql = "select * from information_schema.tables where table_schema = 'cldb_cica'";  
    List<String> nombreTablas = new ArrayList<>();  
    result = statement.executeQuery(selectSql);  
    while (result.next()) { nombreTablas.add(result.getString(1));}  
    //Cambio a todas las tablas  
    for (int i = 0; i < nombreTablas.size(); i++) {  
        String table = nombreTablas.get(i);  
        selectSql = "ALTER TABLE " + table + " ENGINE=InnoDB";  
        statement = connection.createStatement();  
        result = statement.executeQuery(selectSql);  
    }  
  
    long start = System.currentTimeMillis();  
    Thread.sleep(2000);  
    //REALIZAR CONSULTA  
    selectSql = "SELECT ex.exon_id, end_phase, biotype\n" +  
        "FROM transcript tr, exon_transcript extr, exon ex\n" +  
        "WHERE tr.transcript_id = extr.transcript_id AND extr.exon_id = ex.exon_id AND biotype='miRNA'\n" +  
        "ORDER BY exon_id;";  
  
    result = statement.executeQuery(selectSql);  
    long elapsedTimeMillis = System.currentTimeMillis() - start;  
    float elapsedTimeSec = elapsedTimeMillis/1000F;  
  
    while (result.next()) {  
        System.out.println(result.getInt(1)+ " " + result.getString(2));  
        //columnas, una por cada atributo pedido  
    }  
  
    System.out.println("execution time: " + elapsedTimeSec + " seconds");  
}  
  
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) {}  
}
```

2. ÍNDICES

En el esquema de tablas y relaciones mostrado en el apartado 2, 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.

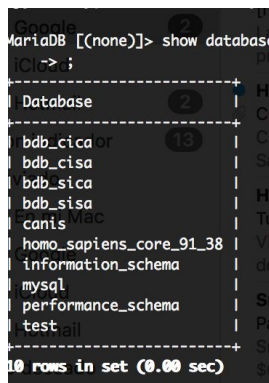
Por ejemplo, para la consulta

```
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'
ORDER BY exon_id;
```

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

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

Tras definir los tipos de motores de almacenamiento e índices, procedemos a la creación de las 4 versiones:



```
MariaDB [(none)]> show databases
+-----+
Database
+-----+
bdb_cicador
bdb_cisa
bdb_sisa
bdb_sisa
canis
homo_sapiens_core_91_38
information_schema
mysql
performance_schema
test
+-----+
10 rows in set (0.00 sec)
```

Comprobamos que los cambios se han aplicado correctamente:

Se definen una serie de comandos para listar los motores de almacenamiento utilizados por todas las tablas en cada una de las base de datos y así comprobar que todo se ha desarrollado correctamente:

```
MariaDB [(none)]> select table_name,engine from information_schema.tables where
table_schema='bdb_sisa';
```

```
+-----+-----+
```

| table_name | engine |
|-------------------|--------|
| alt_allele | MyISAM |
| alt_allele_attrib | MyISAM |
| alt_allele_group | MyISAM |

MariaDB [(none)]> **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 |

Finalmente, tras haber creado las cuatro versiones de nuestra base de datos en cada gestor, comparamos los tiempos de consulta en cada versión basándonos en la siguiente consulta:

SELECT algo FROM dond

```
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;
```

V1 → Sin índices ni almacenamiento (bdb_sisa)

V2 → Con índices pero sin almacenamiento (bdb_cisa)

V3 → Sin índices pero con almacenamiento (bdb_sica)

V4 → Con índices y con almacenamiento (bdb_cica)

| Gestor | versión | Tiempo |
|---------|---------|----------|
| MariaDB | V1 | 15.546 s |
| MariaDB | V2 | 0.081 |
| MariaDB | V3 | 0.094 |
| MariaDB | V4 | 0.025 |
| MySQL | V1 | 4.509 |
| MySQL | V2 | 0.042 |
| MySQL | V3 | 2.134 |
| MySQL | V4 | 0.091 |

Por último añadimos el enlace de nuestro repositorio GitHub donde hemos estamos trabajando el proyecto:

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

5 .DESPLIEGUE EN SGBD XML

En este apartado vamos a trabajar con un SGBD XML denominado ExistDB, para el cual se diseñar 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.

Para este apartado hacemos uso Docker, un proveedor de contenedores que aborda cada aplicación a través de contenedores virtuales.

<https://www.docker.com/what-docker>

Instalación de MySQL y ExistDB en Docker

En primer lugar procedemos a la **instalación de la imagen de MySQL:**

En este punto, suponemos que ya tenemos instalado Docker en nuestro equipo.

A continuación descargamos la imagen Docker de MySQL con el siguiente comando

\$docker pull mysql:5.7.17

Una vez descargada la imagen podemos comprobar que la tenemos instalada con el comando

\$docker images

| REPOSITORY | TAG | IMAGE ID | CREATED | SIZE |
|------------------------|-------------|--------------|---------------|-------|
| evolvedbinary/exist-db | eXist-4.1.0 | 8aaf9e12f4b3 | 4 weeks ago | 956MB |
| mysql | 5.7.17 | 9546ca122d3a | 13 months ago | 407MB |

El siguiente paso es ejecutar la imagen mysql y para ello ejecutamos el comando

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

3b1f6c4967b51cd703b98376785de49cd180bb5fb62984d0150d22f76336aefb

Con este comando estamos ejecutando la imagen mysql con el tag 5.7.1 y se crea un contenedor con el nombre *container-name* y expone el puerto 3306 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.

En segundo lugar se explica la **instalación de la imagen ExistDB** en Docker:

Descargamos la imagen del gestor:

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

Corremos la imagen del gestor:

```
$docker run -d -p 8080:8080 -e EXIST_ADMIN_PASSWORD=name -v  
$localPath:/opt/exist_data/export/ davidgaya/existdb:latest
```

Comando para comprobar las imágenes que tenemos:

```
docker ps -a
```

Correr imagen:

```
curl http://localhost:8080/exist/
```

Obtención del archivo XML

A continuación y ya que tenemos los contenedores correspondientes creados, procedemos a la creación de una base de datos con menos información que las trabajadas hasta ahora, ya que un tamaño demasiado grande nos complicarán las consultas posteriormente (bdb_xml):

```
$docker start mysql-container
```

```
$docker exec -it mysql-container /bin/bash
```

```
root@6a5de33c448e:/# mysql -u root -p
```

```
Enter password:
```

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

```
Your MySQL connection id is 9
```

```
Server version: 5.7.17 MySQL Community Server (GPL)
```

```
Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.
```

```
Oracle is a registered trademark of Oracle Corporation and/or its  
affiliates. Other names may be trademarks of their respective  
owners.
```


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

```
mysql> show databases;
```

```
+-----+  
| Database      |  
+-----+  
| information_schema |  
| bdb_xml        |  
| mysql          |  
| performance_schema |  
| sys            |  
+-----+
```

En la base de datos bdb_xml vamos a introducir un archivo .sql con la información de una tabla del organismo HomoSapiens, posteriormente y mediante un comando, convertiremos dicha información a un archivo XML.

Para la inserción del archivo .sql, éste debe estar dentro del contenedor:

```
root@6a5de33c448e:/home# exit
```

```
exit
```

```
$ cd /Users/Paloma/MEGA/3SegundoCuatrimestre/BDB/Proyecto
```

```
$ docker cp /Users/.../bdb_sica_for_xml.sql mysql-container:/home/bdb_sica_for_xml.sql
```

```
$ docker exec -it mysql-container /bin/bash
```

```
root@6a5de33c448e:/# cd home
```

```
root@6a5de33c448e:/home# ls
```

```
bdb_sica_for_xml.sql
```

Y a continuación podemos volcar dicho archivo en la base de datos creada:

```
oot@6a5de33c448e:/home# mysql -u root -p bdb_xml < bdb_sica_for_xml.sql
```

```
Enter password:
```

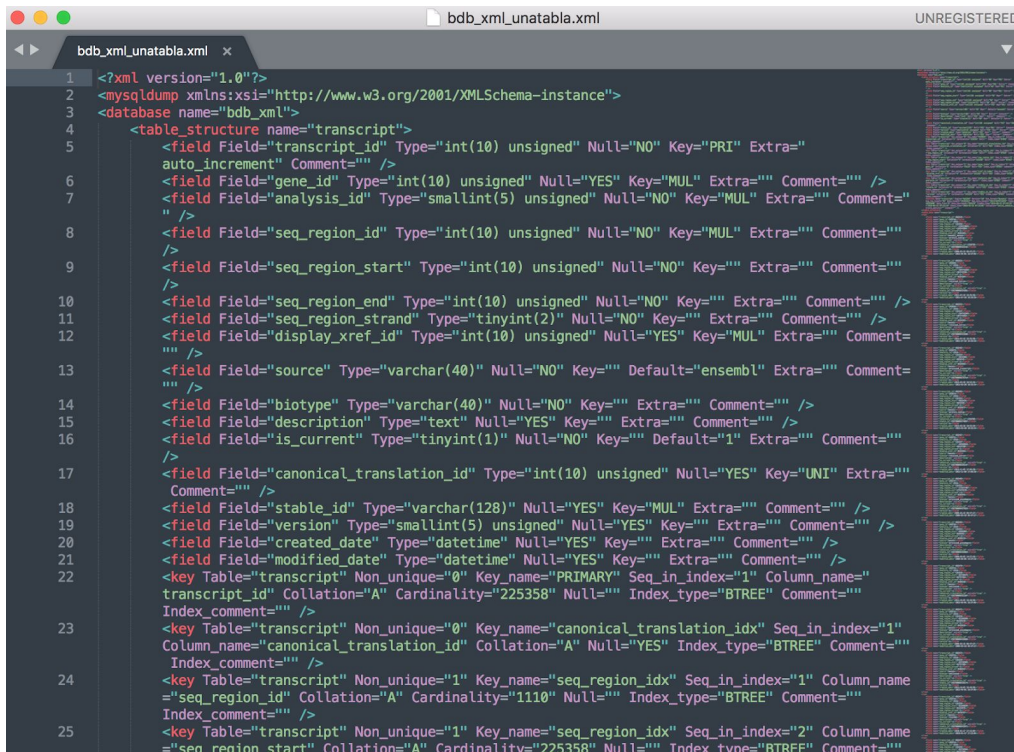
Por último convertimos la base de datos a un archivo XML y lo exportamos al ordenador:

```
mlot@6a5de33c448e:/home# mysqldump --xml -u root -p bdb_xml > bdb_xml_unatabla.x
```

```
Enter password:
```

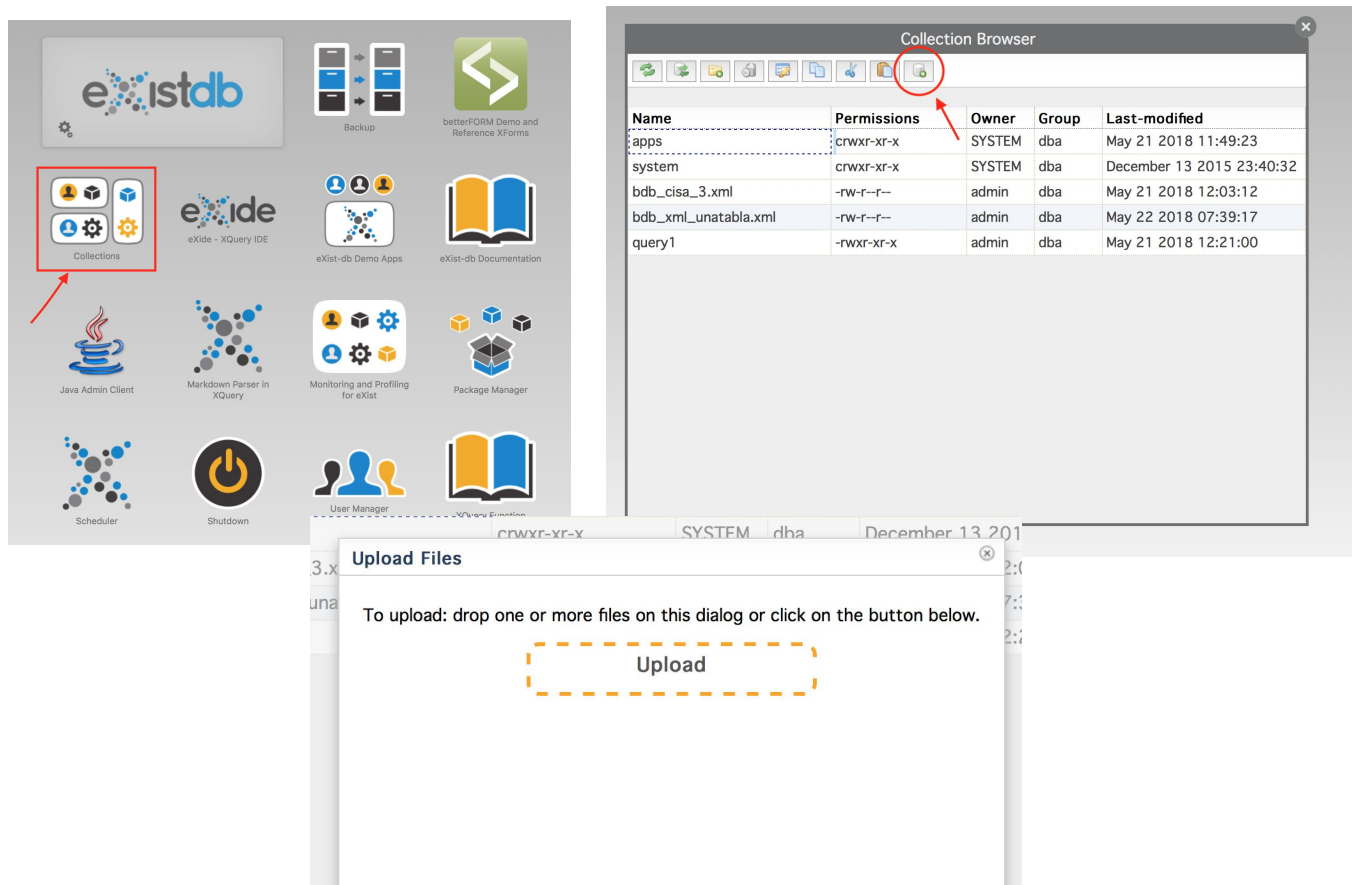
```
#exit
```

```
$docker cp mysql-container:/home/bdb_xml_unatabla.xml /Users/...
```



```
<?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="analysis_id" Type="smallint(5) unsigned" Null="NO" Key="MUL" Extra="" Comment=""
        />
      <field Field="seq_region_id" Type="int(10) unsigned" Null="NO" Key="MUL" Extra="" Comment=""
        />
      <field Field="seq_region_start" Type="int(10) unsigned" Null="NO" Key="" Extra="" Comment=""
        />
      <field Field="seq_region_end" Type="int(10) unsigned" Null="NO" Key="" Extra="" Comment="" />
      <field Field="seq_region_strand" Type="tinyint(2)" Null="NO" Key="" Extra="" Comment="" />
      <field Field="display_xref_id" Type="int(10) unsigned" Null="YES" Key="MUL" Extra="" Comment=""
        />
      <field Field="source" Type="varchar(40)" Null="NO" Key="" Default="ensembl" Extra="" Comment=""
        />
      <field Field="biotype" Type="varchar(40)" Null="NO" Key="" Extra="" Comment="" />
      <field Field="description" Type="text" Null="YES" Key="" Extra="" Comment="" />
      <field Field="is_current" Type="tinyint(1)" Null="NO" Key="" Default="1" Extra="" Comment=""
        />
      <field Field="canonical_translation_id" Type="int(10) unsigned" Null="YES" Key="UNI" Extra=""
        Comment="" />
      <field Field="stable_id" Type="varchar(128)" Null="YES" Key="MUL" Extra="" Comment="" />
      <field Field="version" Type="smallint(5) unsigned" Null="YES" Key="" Extra="" Comment="" />
      <field Field="created_date" Type="datetime" Null="YES" Key="" Extra="" Comment="" />
      <field Field="modified_date" Type="datetime" Null="YES" Key="" Extra="" Comment="" />
      <key Table="transcript" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1" Column_name="
        transcript_id" Collation="A" Cardinality="225358" Null="" Index_type="BTREE" Comment=""
        Index_comment="" />
      <key Table="transcript" Non_unique="0" Key_name="canonical_translation_idx" Seq_in_index="1"
        Column_name="canonical_translation_id" Collation="A" Null="YES" Index_type="BTREE" Comment=""
        Index_comment="" />
      <key Table="transcript" Non_unique="1" Key_name="seq_region_idx" Seq_in_index="1" Column_name
        ="seq_region_id" Collation="A" Cardinality="1110" Null="" Index_type="BTREE" Comment=""
        Index_comment="" />
      <key Table="transcript" Non_unique="1" Key_name="seq_region_idx" Seq_in_index="2" Column_name
        ="seq_region_start" Collation="A" Cardinality="225358" Null="" Index_type="BTREE" Comment=""
        Index_comment="" />
    </table_structure>
  </database>
</mysqldump>
```

Para la segunda parte de este apartado, se abre existDB en <http://localhost:8080/exist/>. En collections podemos seleccionar el script .xml creado y subirlo a eXistDB, desde donde realizaremos posteriormente las consultas correspondientes.



Con respecto a las consultas en ExistDB, en un principio nos enfrentamos al siguiente error:

GC Overhead limit exceeded

Dicho error se debía al tamaño de la caché de la página. El objetivo principal de esta memoria caché es asegurarse de que las páginas de estos archivos que se usan con más frecuencia se guarden en la memoria. Si el caché de un archivo es demasiado pequeño, eXist comienza a descargar páginas, solo para volver a cargarlas un momento después. Este "efecto de eliminación de basura" da como resultado una caída inmediata del rendimiento, en particular al indexar documentos.

Para solucionar dicho error averiguamos que todas las memorias caché comparten un único conjunto de memoria, cuyo tamaño está determinado por el atributo `cacheSize` en la sección `<db-connection>` de `conf.xml`.

Por tanto accedimos al contenedor donde teníamos instalado Exit DB:

```
alu-242-176:~ Paloma$ docker exec -it compassionate_hypatia /bin/bash
root@28cdead9c871:/# ls -l /opt/
```

Posteriormente buscamos el archivo `conf.xml` y modificamos el tamaño de la caché a 500 MB:

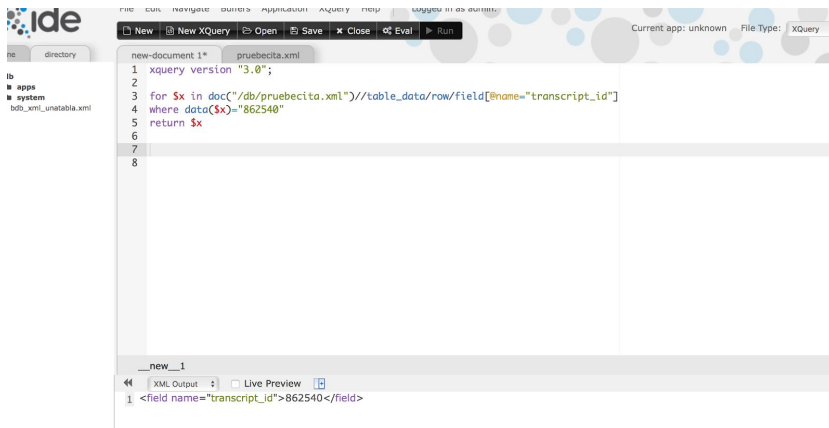
```
root@28cdead9c871:/# nano /opt/exist/conf.xml
```

```
<db-connection cacheSize="48M" collectionCache="24M" database="native"
    files="webapp/WEB-INF/data" pageSize="4096" nodesBuffer="-1">
```

En este momento, procedimos a realizar las consultas con normalidad:

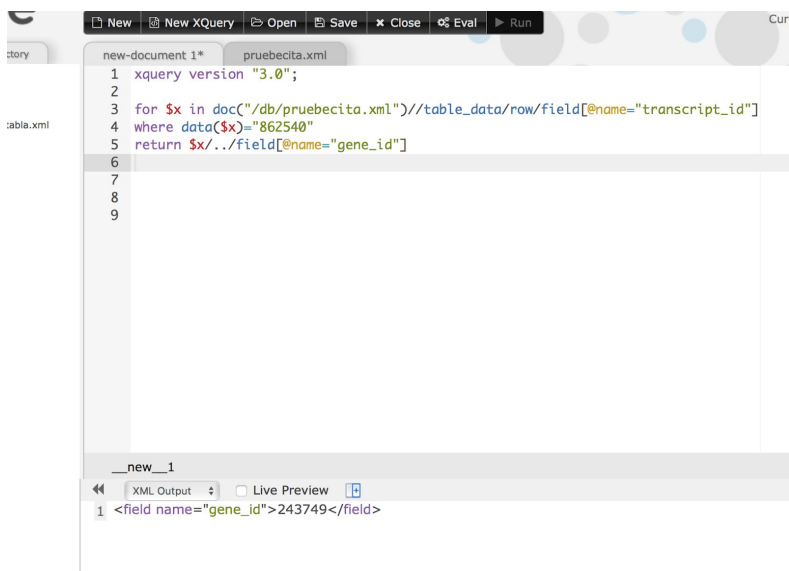
#Consulta que devuelve el transcript_id dado:

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



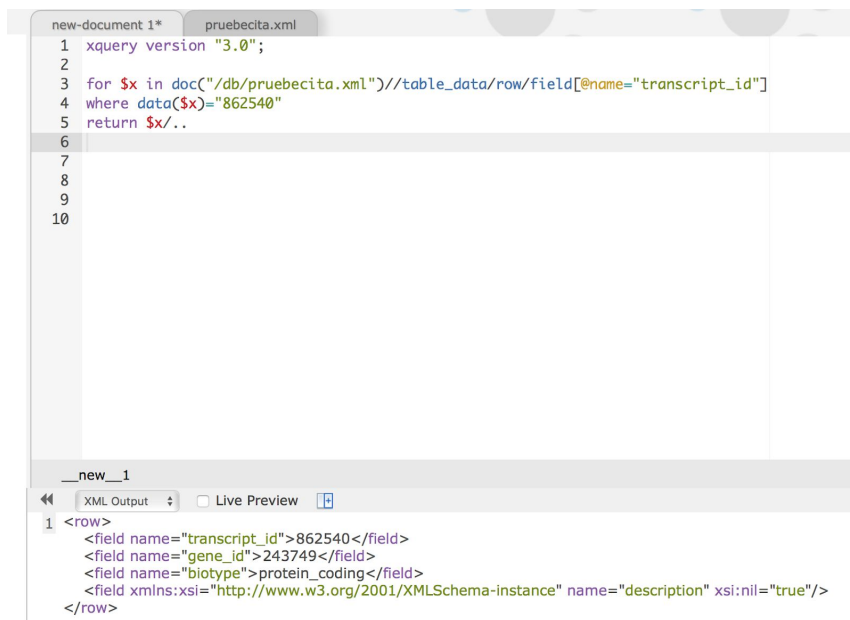
#Consulta que devuelve el gene_id para un transcript_id dado

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



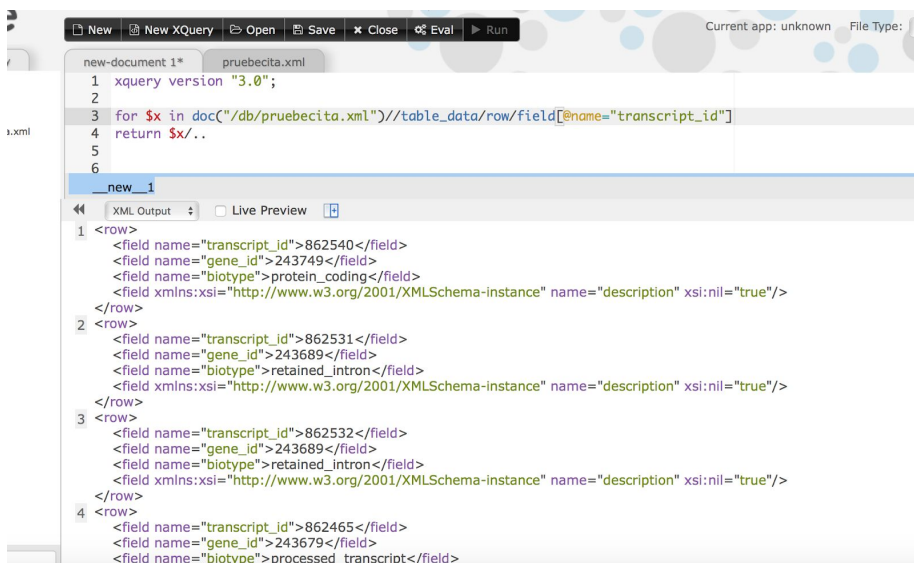
#Consulta que devuelve todos los datos de la tabla para un transcript_id dado

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



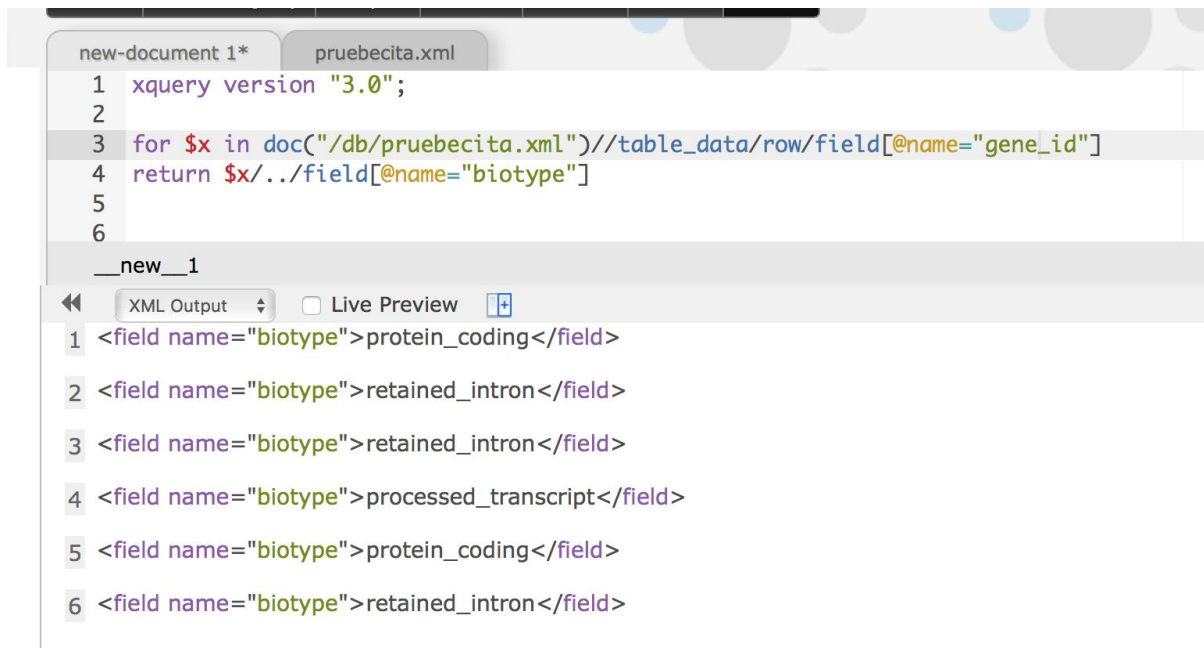
#Consulta que devuelve la información de todas las tablas presentes filtrando por transcript_id

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



#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"]
```



6. INTEGRACIÓN DE FRONT-END Y BACK-END

Para explicar adecuadamente este punto, es necesario aclarar como se ha realizado cada una de las partes. Por un lado hemos generado una interfaz con Angular 4. Hemos tenido diferentes versiones hasta llegar a la que tenemos actualmente:

Pestaña Home:

The screenshot shows a web browser window at localhost:4200. The page has a dark header with a logo and navigation links for 'Home' and 'Buscador'. The main content area has a title 'Encuentra como hacer la mejor petición' and a paragraph explaining the service's goal: to create a list of requests over different databases and allow users to check the time taken for each request to find the fastest way to make a request. Below the text is a blue button labeled 'Buscador'.

Pestaña Buscador:

The screenshot shows the 'Buscador' page of the application. It features a form with three main sections: 'Columnas', 'Tablas', and 'Condiciones'. Each section has input fields for the number of items and a list of specific items. Below the form are buttons for 'Group by' and 'Crear consulta'. At the bottom, there is a section titled 'Datos introducidos' with a table showing the input data and a 'Send petitions' button.

| Numero de Columnas | Columnas | Numero de tablas | Tablas | Numero de condiciones | Condiciones | Group by |
|--------------------|--|------------------|-------------------------------|-----------------------|--|----------|
| 4 | Nombre de columna 1 Nombre de columna 2 Nombre de columna 3 Nombre de columna 4 | 3 | Tabla 1 Tabla 2 Tabla 3 | 4 | Condición 1 AND Condición 2 AND Condición 3 AND Condición 4 | |

Esta interfaz 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 son enviadas a través de un método post hacia el backEnd.

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.

Una vez aclarada la implementación de la interfaz pasamos a buscar cómo implementar el BackEnd.

Para ello hemos decidido utilizar Spring. Esta tecnología nos permite integrar esta interfaz dentro de un proyecto de Spring en el que podremos implementar el Back-End.

¿Cómo se ha integrado el proyecto?

Tecnologías necesarias:

1. Java 1.8
2. Maven 3.3.9
3. Spring Tool Suite – Version 3.8.1.RELEASE
4. Spring Boot: RELEASE
5. Angular 4
6. Node.js

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.

```
org.springframework.boot spring-boot-starter-web
```

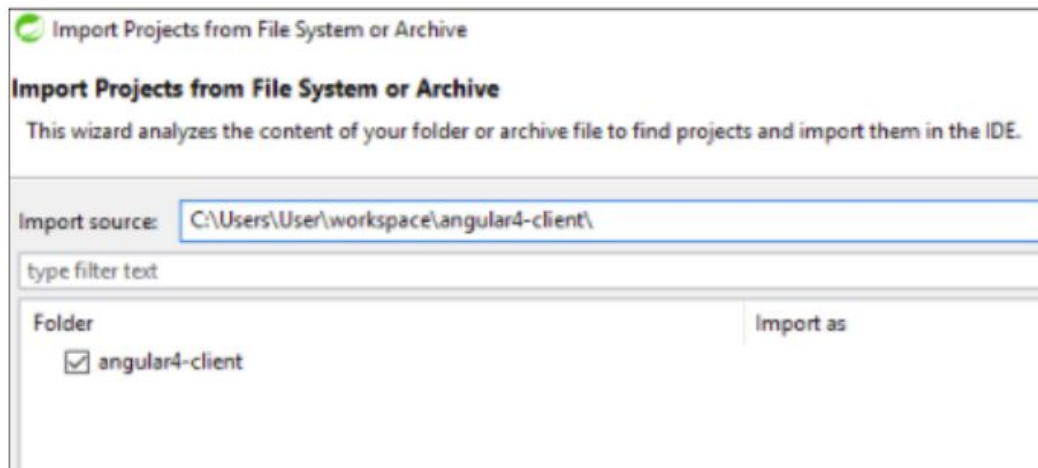
En el proyecto vamos a crear un controlador muy simple, en

```
/source/main/java/com/example/demo/packagecom.javasampleapproach.restful.controller;
```

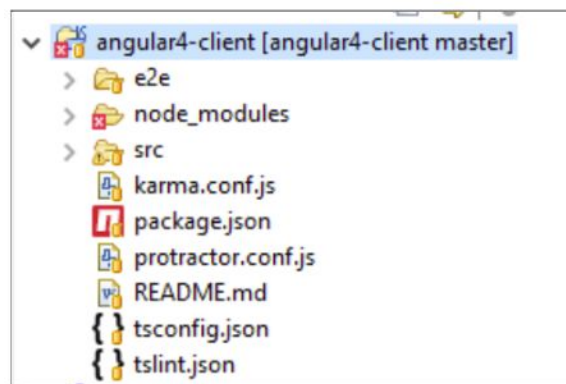
```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController public class WebRestController {
    @RequestMapping("/api/hi") public String hi() {
        return "Hello World from Restful API";
    }
}
```

Teniendo ya nuestra primera versión para el controlador, pasaremos a integrar ambas partes. Para ello solo tendremos que abrir SpringToolSuite y ir a:

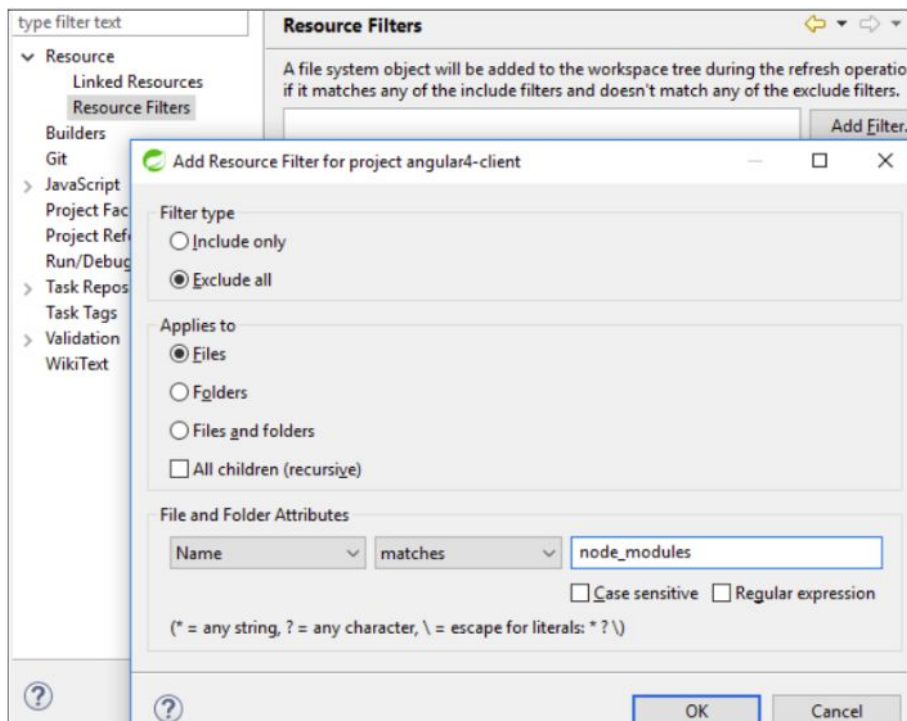
Import -> General -> Projects from Folder or Archive -> aceptar.



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



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:



Cuando acabe tendremos un proyecto de Spring en el que tenemos el BackEnd, 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.

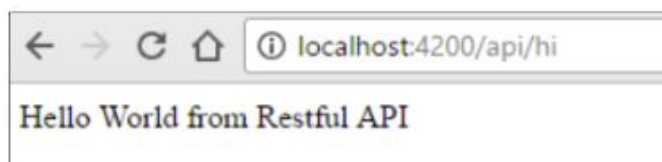
```
{ "/api": { "target": "http://localhost:8080", "secure": false } }
```

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"
},
...
```

5

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> obtenemos el siguiente resultado:



Ya 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 construir y ejecutar nuestro SpringBoot server.

- Build:

```
mvn clean install
```

- Run:

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

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.

7. DESARROLLO DEL BACK-END

El esqueleto del back-end consiste en un servicio REST en el que está esperando a que llegue una petición a la dirección /api/hi.

```
package com.example.demo;

import ...

@SpringBootApplication
public class DemoApplication {

    public static void main(String[] args) { SpringApplication.run(DemoApplication.class, args); }

}
```

Esta clase está mapeando la llegada de peticiones a la dirección /api/hi y en el momento que llega se usa la función getFoosBySimplePath1. Recoge el “Request Body” y lo procesa, devolviendo los datos deseados a la interfaz.

```
@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 String getFoosBySimplePath1(@RequestBody String body) {
        System.out.println(body);
        return body;
    }

}
```

8. FLUJO DE FUNCIONAMIENTO

En primer lugar creamos una petición rellenando los datos correspondientes.

The screenshot shows a web browser at localhost:4200. The application has a dark header with 'Home' and 'Buscador' links. The main content area is a form with three columns:

- Columnas:** 'Numero de Columnas' is 4. Below, 'SELECT' section has four input fields for column names.
- Tablas:** 'Número de tablas' is 3. Below, 'FROM' section has three input fields for table names.
- Condiciones:** 'Número de condiciones' is 4. Below, 'WHERE' section has four input fields for conditions.

At the bottom of the form are two buttons: 'Group by' and 'Crear consulta'. Below the form is a section titled 'Datos introducidos' with a horizontal list of tabs: 'Numero de Columnas', 'Columnas', 'Numero de tablas', 'Tablas', 'Numero de condiciones', 'Condiciones', and 'Group by'. At the very bottom is a blue button labeled 'Send petitions'.

Esta petición se guardará en la tabla y podremos almacenar tantas peticiones como queramos, esta funcionalidad la hemos implementado en JavaScript.

```
savePetition(petitionForm: NgForm):void{
  this.petition.push(petitionForm.value);
  this.msg='Petición añadida exitosamente!';
  this.ocultaGroupBy=true;
}
```

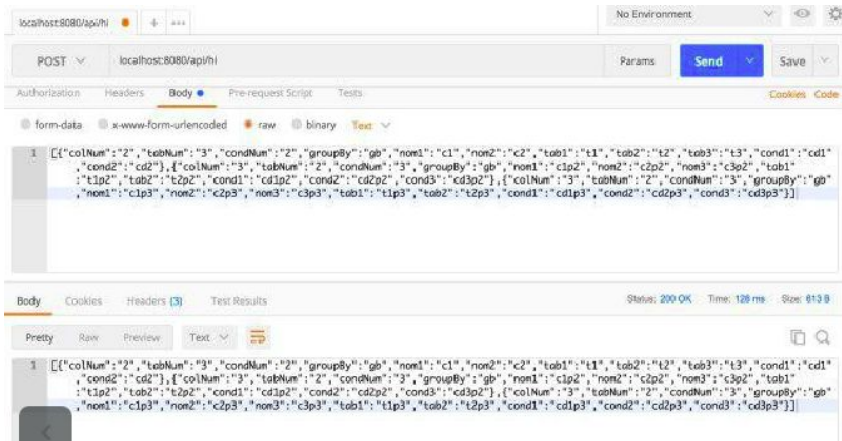
Una vez tenemos todas las peticiones, con el botón de “Send petitions” enviamos las peticiones al backEnd. Esto se hará a través de un método POST que hemos implementado tal que así:

```

enviarPost():void{
  const req= this._http.post("http://localhost:8080/api/hi/", this.petition,{responseType: 'text'} )
  .subscribe(
    res=> {
      console.log("Petición POST realizada al servidor");
      console.log(res);
      console.log("Esperando respuesta...");
    },
    err=>{
      console.log("Petición fallida");
      console.log(err);
    }
  )
}
}

```

Esta petición envía un objeto JSON que tiene la siguiente estructura al back-end.



En el back-end vamos a desgranar este JSON en sus diferentes componentes.

```

// ...
herServlet: initialization completed in 12 ms
JSONOBJECT 0: {"nom2":"c1", "nom1":"c1", "tabNum":"3", "cond2":"cd2", "colNum":"2", "tab3":"t3", "cond1":"cd1", "tab1":"t1", "condNum":
"2", "groupBy":"gb", "tab2":"t2"}

jsonobj 0 NUM TABLAS :3 NUM COL: 2 NUM COND: 2

columnas: [c1, c2]

tablas: [t1, t2, t3]

condiciones: [cd1, cd2]

JSONOBJECT 1: {"nom3":"c3p2", "nom2":"c2p2", "nom1":"c1p2", "cond3":"cd3p2", "tabNum":"2", "cond2":"cd2p2", "colNum":"3", "cond1":"cd
p2", "tab1":"t1p2", "condNum":"3", "groupBy":"gb", "tab2":"t2p2"}

jsonobj 1 NUM TABLAS :2 NUM COL: 3 NUM COND: 3

columnas: [c1p2, c2p2, c3p2]

tablas: [t1p2, t2p2]

condiciones: [cd1p2, cd2p2, cd3p2]

JSONOBJECT 2: {"nom3":"c3p3", "nom2":"c2p3", "nom1":"c1p3", "cond3":"cd3p3", "tabNum":"2", "cond2":"cd2p3", "colNum":"3", "cond1":"cd
p3", "tab1":"t1p3", "condNum":"3", "groupBy":"gb", "tab2":"t2p3"}

jsonobj 2 NUM TABLAS :2 NUM COL: 3 NUM COND: 3

columnas: [c1p3, c2p3, c3p3]

tablas: [t1p3, t2p3]

condiciones: [cd1p3, cd2p3, cd3p3]

SELECT t1, t2t3, FROM t1, t2t3, WHERE cd1 AND cd2 GROUP BY gb:

```

Y en base a estos elementos generamos las consultas que dependiendo del tipo de base de datos que elijamos, generará una query diferente que se pasará a la correspondiente clase para hacer la petición (Clases de java subidas anteriormente). Y una vez se calculan los tiempos se vuelven a pasar en el return a la interfaz.

9.No RELACIONALES

Porqué usar una BD no relacional:

-El servicio de MySQL está 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*.

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>

Primero, preparamos las herramientas para la migración, para ello nos apoyaremos sobre un conjunto de scripts del usuario lovette en GitHub.

Dispondremos de las siguientes herramientas.

my2mo-fields

my2mo-export

my2mo-import

Pasos seguidos:

1.**Crear un entorno** donde trabajar cómodamente, en nuestro caso lo que realizo es lo siguiente:

```
$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...

| | |
|---------------------------|------------------|
| <i>...country</i> | <i>7 fields</i> |
| <i>...ip2nation</i> | <i>2 fields</i> |
| <i>...language</i> | <i>5 fields</i> |
| <i>...offer</i> | <i>15 fields</i> |
| <i>...source</i> | <i>2 fields</i> |
| <i>...updater</i> | <i>18 fields</i> |
| <i>...updater_country</i> | <i>3 fields</i> |
| <i>...updater_offer</i> | <i>2 fields</i> |
| <i>...updater_user</i> | <i>13 fields</i> |
| <i>...version</i> | <i>4 fields</i> |

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**

El siguiente comando no modifica nada de la base de datos existente, lo único que hace es crear el fichero *export.sql* que ejecuta los comandos necesarios para poder generar los *csv*.

```
$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
```

[export es el nombre por defecto que genera]

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 este importado

\$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

Este proceso lo repetiremos para las 4 bases de datos diferentes.

Algunos ejemplos de consultas que hemos probado son:

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

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

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

51

REFERENCIAS

Enlace base de datos ensembl:

<http://www.ensembl.org/index.html>

<ftp://ftp.ensembl.org/pub/>">FTP site

Comandos MAC:

<https://www.macnux.tk/comprimir-descomprimir-consola-mac-osx/>

Descarga Ensembl:

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

MariaDB para mac:

<https://mariadb.com/resources/blog/installing-mariadb-10010-mac-os-x-homebrew>

Instalacion mySQL:

<http://migueleonardortiz.com.ar/mysql/instalando-mysql-en-windows-o-linux/1015>

Definicion Base de Dato Relacional:

https://es.wikipedia.org/wiki/Base_de_datos_relacional

Creación de una BD para MariaDB:

https://elbinario.net/wp-content/uploads/2015/02/Introducción_a_MariaDB1.pdf

<http://migueleonardortiz.com.ar/mysql/creacion-base-de-datos-en-mariadb-mysql/1070>

<http://codigoxules.org/tutorial-mariadb-creando-tablas-en-sql/>

Resetear contraseña root en MariaDB:

<https://www.rosehosting.com/blog/how-to-reset-your-mariadb-root-password/>

Creación de las TABLE desde EMSEMBL: (explicado por pasos)

<https://www.ensembl.org/info/docs/webcode/mirror/install/ensembl-data.html>

Para insertar datos en una tabla a través de archivo .txt:

<http://www.webestilo.com/mysql/cargar-datos-tabla.phtml>

<http://www.forosdelweb.com/f86/importar-csv-txt-mysql-739305/>

Mejor ayuda de ensembl para meter todos los archivos a la vez:

<https://m.ensembl.org/info/docs/webcode/mirror/install/ensembl-data.html>

Ensembl core schema documentation:

https://www.ensembl.org/info/docs/api/core/core_schema.html

Enlace GitHub con los comandos de la base de datos:

<https://gist.github.com/hofmannsven/9164408>

Consultas multitabla:

<https://www.campusmvp.es/recursos/post/Fundamentos-de-SQL-Consultas-SELECT-multi>

[-tabla-JOIN.aspx](#)

Problema GC Time Exceed:

<https://exist-db.org/exist/apps/doc/tuning.xml>

Consultas XQUERY:

<https://www.adictosaltrabajo.com/tutoriales/introduccion-x-query/>