

MEMORIA PRÁCTICA – 4: Creación de la BBDD, carga de datos y consultas en MongoDB



Escuela Politécnica Superior - Universidad Autónoma de Madrid
GRADO EN INGENIERÍA BIOMÉDICA
GESTIÓN DE BASES DE DATOS BIOMÉDICOS

Versión del documento número 1

Práctica realizada por: Felipe Ruiz Bernal y Laura Sánchez Garzón

Nº grupo: 6212

Nº pareja: 11

Fecha: 17/04/2023

Profesorado de la práctica: Ruth Cobos Pérez y Julio Esparza Ibáñez

1. ÍNDICE

Contenido

1. ÍNDICE.....	2
2. ANEXO	3
3. INTRODUCCIÓN	4
4. DIFERENCIAS ENTRE BASES DE DATOS SQL Y NoSQL.....	5
4.1. Mongo DB.....	6
5. CÓDIGO	7
6. CONSULTAS.....	8
6.1. Consulta 0.....	8
6.1.1. Consulta en Mongo Shell.....	8
6.1.2. Explicación.....	8
6.1.3. Resultado.....	8
6.1.4. Comprobación en PostgreSQL.....	9
6.2. Consulta 1.....	10
6.2.1. Consulta en Mongo Shell.....	10
6.2.2. Explicación.....	10
6.2.3. Resultado.....	10
6.2.4. Comprobación en PostgreSQL.....	11
6.3. Consulta 2.....	12
6.3.1. Consulta en Mongo Shell.....	12
6.3.2. Explicación.....	12
6.3.3. Resultado.....	12
6.3.4. Comprobación en PostgreSQL.....	13
6.4. Consulta 3.....	14
6.4.1. Consulta en Mongo Shell.....	14
6.4.2. Explicación.....	14
6.4.3. Resultado.....	14
6.4.4. Comprobación en PostgreSQL.....	15
6.5. Consulta 4.....	16
6.5.1. Consulta en Mongo Shell.....	16
6.5.2. Explicación.....	16
6.5.3. Resultado.....	16
6.5.4. Comprobación en PostgreSQL.....	17
7. CONCLUSIONES	19

2. ANEXO

Para esta última práctica, se ha reutilizado la estructura del modelo relacional aplicado a todas las prácticas anteriores. Sin embargo, para facilitar el traspaso de información, se han realizado cambios en el nombre de las entidades, en concreto, se han sustituido aquellas palabras que incluían la letra “ñ”, y las vocales con tilde han sido sustituidas por vocales simples. Estos detalles, que en SQL no afectaban, en MongoDB se comprobó que deceleraba la automatización, por lo que el modelo relacional utilizado hasta entonces (ver Figura 1), se ha modificado por el actualizado (ver Figura 2). Este nuevo modelo relacional servirá para realiza parte del código *Python* utilizado para traspasar los datos en forma *.csv*, a forma *.js*.

Cabe destacar que la entidad Pertenece que venía de la relación muchos a muchos de Pais con Agrupacion_Paises no está presente en la base de datos en MongoDB, en su lugar, se ha introducido un campo nuevo en Pais que contiene las agrupaciones a las que pertenece, a su vez, se ha añadido de manera voluntaria otro parámetro en Agrupacion_Paises que contiene los países asociados a cada agrupación, lo que hizo de manera involuntaria que la consulta 3 se simplificase.

Región	CódigoR*						
Nivel_Profesional	CódigoP*	DescripciónP					
Sexo	CódigoS*	DescripciónS					
Unidad_de_Medida	Unidad*	DescripciónU					
Tipo_de_Representación	Tipo*	DescripciónR					
Categoría	Categ*	DescripciónC					
Pais	CódigoPa	CódigoOMS	Nombre_cortoP	Nombre_completoP			
Agrupación_Paises	CódigoG*	Nombre_cortoG	Nombre_completoG				
Indicador	CódigoI*	DescripciónI	Unidad	Tipo	Categ		
Valor_anual	Clave_Serial*	Valor	Año	Código	CódigoR	CódigoS	CódigoP
Pertenece	Clave_Ser*	CódigoPa	CódigoG				

Figura 1: Modelo relacional que se utilizó para realizar las prácticas anteriores, sobre el que se han realizado cambios para poder trabajar sin complicaciones en MongoDB.

Region	CodigoR*						
Nivel_Profesional	CodigoP*	DescripcionP					
Sexo	CodigoS*	DescripcionS					
Unidad_de_Medida	Unidad*	DescripcionU					
Tipo_de_Representacion	Tipo*	DescripcionR					
Categoria	Categ*	DescripcionC					
Pais	CodigoPa	CodigoOMS	Nombre_cortoP	Nombre_completoP			
Agrupacion_Paises	CodigoG*	Nombre_cortoG	Nombre_completoG				
Indicador	CodigoI*	DescripcionI	Unidad	Tipo	Categ		
Valor_anual	Clave_Serial*	Valor	Year	Codigol	CodigoR	CodigoS	CodigoP

Figura 2: Modelo relacional actualizado.

3. INTRODUCCIÓN

Esta tercera práctica va enfocada a aplicar los conocimientos teóricos sobre cómo realizar consultas en lenguaje MongoDB, en prácticas reales. Para ello se han utilizado las tablas cargadas con las bases de datos creadas en las anteriores prácticas, y se le ha ofrecido al alumno una serie de enunciados a seguir para realizar dichas consultas. También se le da la libertad de realizar consultas extras por pura curiosidad.

En la práctica anterior se utilizó SQL para realizar las consultas. En este caso, se ha utilizado MongoDB, un programa que trabaja con bases de datos NoSQL, lo que implica cambiar de mentalidad al pensar en Bases de Datos como modelos *ACID* (*Atomicity, consistency, isolation* y *durability*), y empezar a considerar las bases de datos como modelos *BASE* (*Basically Available, soft state* y *eventually consistent*). El siguiente apartado va dirigido a entender las principales diferencias entre las bases de datos SQL y NoSQL, para poder entender más fácilmente las consultas de MongoDB.

Se recuerda que la base de datos que se está utilizando corresponde a una base de datos real, (llamada HlthRes-DB), compartida en Internet por la OMS (accesible mediante el enlace <https://gateway.euro.who.int/en/datasets/european-database-on-human-and-technical-resources-for-health/>), que recoge estadísticas sobre recursos humanos y técnicos relacionados con el ámbito de la salud a lo largo de los años, en aquellos países europeos a los que la OMS tiene acceso.

A lo largo de la memoria se va a explicar los pasos dados durante la práctica.

4. DIFERENCIAS ENTRE BASES DE DATOS SQL Y NoSQL

Para explicar las principales diferencias, se va a recurrir a las nemotecnias *ACID* y *BASE*. Las **bases SQL** son modelos **ACID**:

Atomicidad significa que, para realizar una transacción, es necesario completar todos los pasos o ninguno. Esto nos lleva al segundo punto, **consistencia**, dado que una transacción no deja una base de datos en un estado que viola la integridad de los datos. Además, estas transacciones no son visibles (**aislamiento**) a otros usuarios hasta que se completa su ejecución. Por último, **durabilidad** implica que, una vez completada la transacción, el efecto permanecerá incluso si se cae el servidor (los datos son almacenados en medios persistentes).

Las bases de datos **NoSQL** funcionan de manera mucho más flexible (nemotecnia **BASE**): Dado que las bases NoSQL se han creado con el objetivo de desarrollar la interoperabilidad entre sistemas distribuidos, una de las características es que es **basically available**, es decir, puede haber fallos parciales en algunas partes del sistema distribuido, y por ello se guardan varias copias en servidores diferentes. Además, la base de datos puede estar en un estado inconsistente momentáneamente (**eventually consistent**), pero el propio mecanismo de actualización hará desaparecer las inconsistencias, es decir, el estado puede cambiar con el tiempo (**soft state**).

Gracias a la flexibilidad con la que se trabaja en NoSQL (basado en una arquitectura distribuida, no se utilizan estructuras fijas como tablas para el almacenamiento de los datos y no se realizan operaciones JOIN), se pueden hacer cambios de los esquemas (no hay esquemas predefinidos), de manera que la escalabilidad es posible, y existe una considerable optimización de consultas para grandes cantidades de datos, es decir, existen los sistemas distribuidos.

En el caso de esta práctica, MongoDB es un programa que trabaja con bases de datos NoSQL, orientadas a documentos.

Estos documentos pueden almacenar información (de la forma clave-valor, *arrays* o documentos embebidos (de esta forma se evitan los JOINS, razón por la cual se ha eliminado la entidad “pertenece” del modelo relacional), y programas como MongoDB pueden realizar consultas sobre los mismos. El próximo apartado va dirigido a explicar brevemente cómo funcionan los comandos principales de MongoDB.

4.1. Mongo DB

Este programa, que tiene por características el alto rendimiento, la auto replicación, la escalabilidad horizontal, el balanceo de carga y la alta disponibilidad, es uno de los más utilizados para crear, modificar y realizar búsquedas sobre bases de datos tipo NoSQL.

Los pasos por seguir antes de realizar cualquier consulta son, como se hizo ya al trabajar con bases de datos SQL: plasmar la base de datos en un modelo relacional, automatizar el traspaso de información de las tablas en .csv a estructuras tipo documentos NoSQL, y finalmente, crear bases y colecciones en Mongo sobre las que poder programar cualquier consulta.

Como se ha mencionado, MongoDB es un lenguaje de programación orientado a documentos. Éstos, escritos entre llaves ({}), forman parte de colecciones, que forman parte a su vez de bases de datos independientes entre sí. En comparación a los JOINS del lenguaje SQL, en este caso, si se quieren relacionar entidades entre sí, en una misma colección se insertan los documentos que contienen sus correspondientes datos.

Las consultas tienen una estructura sencilla y existen dos principales tipos: *db.nombre_colección.find()*, que en SQL se entiende como la estructura básica SELECT, FROM, WHERE; y *db.nombre_colección.aggregate()*, función con la que poder agrupar, como GROUP BY y similares. Además, existen una serie de métodos y operadores que permiten filtrar las consultas y especificarlas al máximo:

OPERADORES	MÉTODOS
	.insert()
\$gt, \$gte, \$lt, \$lte, \$or, \$ne, \$in	.find()
\$group, \$match, \$avg, \$addFields	.aggregate()
\$count	.count()
	.sort()
	.skip()
\$size	.size()
\$set, \$inc, \$min, \$max, \$mul, \$rename, \$unset, \$setOnInsert	.update()
	.replaceOne()
	.save()
\$text	

5. CÓDIGO DE PYTHON

Para cargar la base de datos en mongo, se han utilizado distintos ficheros de JavaScript que contenían los comandos y datos necesarios tanto para crear las diferentes colecciones como para rellenarlas con los datos pertinentes.

Se ha decidido reutilizar el código de Python ya implementado en la práctica 2 para generar los .js necesarios para generar la base de datos.

Las funciones que accedían a la información han permanecido inalteradas, sólo se han creado 2 funciones más y rehecho el insert completamente para que se ajuste a los formatos tanto de .js como de mongo.

Las funciones nuevas son específicamente para los parámetros nuevos en Agrupacion_Paises y Pais, Paises y Agrup respectivamente. En Países, parámetro de Agrupacion_Países, contiene un array de todos los países que pertenecen a la agrupación en cuestión, mientras que Agrup, parámetro de Pais, contiene las agrupaciones a las que está afiliado un país, también en un array.

Todas las funciones están comentadas dentro del fichero insertmongo.py.

El formato del superscript es similar al proporcionado para la práctica, pero todos los inserts han sido generados por código, ninguno de los ficheros proporcionados ha sido utilizado, aunque sí que se han tenido en cuenta para tener una idea general de cómo debían ser los .js .

Para la comodidad de uso, los inserts se generan en una carpeta en la misma dirección donde se encuentre el .py y te devuelve directamente el comando load('dirección de la carpeta') para que se copie y pegue en el shell de mongoDB.

6. CONSULTAS

En esta sección va dirigida a realizar las consultas de los enunciados propuestos. En cada consulta se mostrará el código utilizado para resolverla, una breve explicación del razonamiento seguido y los comandos utilizados, la salida que se muestra por pantalla, y la comprobación de las búsquedas realizada en PostgreSQL.

6.1. Consulta 0

Obtén un listado con los 20 primeros indicadores registrados en la base de datos. Queremos visualizar el listado con los siguientes campos de cada indicador: código y descripción.

6.1.1. Consulta en Mongo Shell

```
HLTHres> db.Indicador.find({}, {CodigoI:1, DescripcionI:1, _id:0 }).limit(20)
```

Figura 3: Código utilizado para resolver la consulta guiada 0.

6.1.2. Explicación

Se trata de una típica búsqueda SELECT FROM LIMIT. En bases de datos NoSQL, se utilizará el método `.find`, en el que no se especificará ninguna condición (ningún WHERE), pero sí qué columnas devolver, en este caso, CódigoI y DescripciónI. El método `.limit(20)` limita la búsqueda a los primeros 20 resultados (tal y como pide el enunciado).

6.1.3. Resultado

```
[
  {
    CodigoI: 'HLTHRES_1',
    DescripcionI: 'Associate nurses employed by hospital in FTE, per 100 000 population'
  },
  {
    CodigoI: 'HLTHRES_2',
    DescripcionI: 'Associate nurses employed by hospital in FTE, total number'
  },
  {
    CodigoI: 'HLTHRES_3',
    DescripcionI: 'Associate professional midwives, licensed to practice, per 100 000 population'
  },
  ...
  {
    CodigoI: 'HLTHRES_18',
    DescripcionI: 'Associate professional nursing graduates, total number'
  },
  {
    CodigoI: 'HLTHRES_19',
    DescripcionI: 'Beds in for-profit privately owned hospitals, per 100 000 population'
  },
  {
    CodigoI: 'HLTHRES_20',
    DescripcionI: 'Beds in for-profit privately owned hospitals, total number'
  }
]
```

Figura 4: Salida de la consulta guiada 0.

6.1.4. Comprobación en PostgreSQL

Query	Query History
1	SELECT CódigoI, DescripciónI
2	FROM indicador
3	LIMIT 20

Figura 5: Código utilizado para resolver la consulta guiada 0 en SQL.



	códigoi [PK] character varying (100) 	descripcióni character varying (100) 
1	HLTHRES_1	Associate nurses employed by hospital in FTE, per 100 000 population
2	HLTHRES_2	Associate nurses employed by hospital in FTE, total number
3	HLTHRES_3	Associate professional midwives, licensed to practice, per 100 000 population
...		
18	HLTHRES_18	Associate professional nursing graduates, total number
19	HLTHRES_19	Beds in for-profit privately owned hospitals, per 100 000 population
20	HLTHRES_20	Beds in for-profit privately owned hospitals, total number

Figura 6: Salida de la consulta guiada 0.

Se ha podido comprobar que se obtiene el mismo resultado programando en lenguaje SQL y NoSQL.

6.2. Consulta 1

Obtén un listado que muestre cuántas mediciones se han hecho por región. Mostrando de mayor a menor valor por el número mediciones. Limita el resultado a las 10 regiones con mayor número de mediciones.

6.2.1. Consulta en Mongo Shell

```
HLTHres> db.Valor_anual.aggregate([
... {$group: {_id: "$CodigoR", count_mediciones: {$count: {}}}},
... {$sort: {count_mediciones: -1}},
... {$limit: 10}
... ])
```

Figura 7: Código utilizado para resolver la consulta 1.

6.2.2. Explicación

Dado que se buscan mediciones hechas sobre regiones, y la colección Valor_anual está formada por una serie de documentos embebidos, entre ellos, los códigos de región (ver Figura 2), se puede realizar la búsqueda directamente sobre Valor_anual.

El propio enunciado especifica que se agrupe por regiones, y se cuente el número de mediciones hechas en cada una de ellas, por lo que se utilizará el operador *\$group*, seguido por la columna que se quiere agrupar. Además, dentro de este comando, se va a crear una nueva columna, llamada *count_mediciones*, a la que se le aplicará la operación *\$count*. Para ajustarse a lo que indica el enunciado, se han utilizado otros dos operadores: *\$sort*, que ordena lo que se especifique como valor (en este caso, la columna *count_mediciones*, en orden decreciente (*{count_mediciones: -1}*)); y el operador *\$limit*, que limita el número de resultados al número especificado, en este caso, 10 regiones.

6.2.3. Resultado

```
[
{ _id: 'ISR', count_mediciones: 5419 },
{ _id: 'DNK', count_mediciones: 5247 },
{ _id: 'LTU', count_mediciones: 4920 },
{ _id: 'AUT', count_mediciones: 4738 },
{ _id: 'ESP', count_mediciones: 4514 },
{ _id: 'ISL', count_mediciones: 4466 },
{ _id: 'FRA', count_mediciones: 4232 },
{ _id: 'EST', count_mediciones: 3925 },
{ _id: 'BLR', count_mediciones: 3852 },
{ _id: 'NLD', count_mediciones: 3833 }
]
```

Figura 8: Salida de la consulta propuesta 1. Observar que se obtienen dos columnas: *_id* (los códigos de las regiones), y *count_mediciones*, ordenados de mayor a menor.

6.2.4. Comprobación en PostgreSQL

Query	Query History
1	<code>SELECT CódigoR, COUNT(CódigoR) AS count_mediciones</code>
2	<code>FROM Valor_anual</code>
3	<code>GROUP BY CódigoR</code>
4	<code>ORDER BY count_mediciones DESC</code>
5	<code>LIMIT 10</code>

Figura 9: Código utilizado para resolver la consulta guiada 1 en SQL.

	<code>códigoR</code> character varying (100) 🔒	<code>count_mediciones</code> bigint 🔒
1	ISR	5419
2	DNK	5247
3	LTU	4920
4	AUT	4738
5	ESP	4514
6	ISL	4466
7	FRA	4232
8	EST	3925
9	BLR	3852
10	NLD	3833

Figura 10: Salida de la consulta guiada 1.

Se ha podido comprobar que se obtiene el mismo resultado programando en lenguaje SQL y NoSQL.

6.3. Consulta 2

Obtén un listado que muestre cuántas mediciones se han hecho en España para cada nivel profesional.

6.3.1. Consulta en Mongo Shell

```
HLTHres> db.Valor_anual.aggregate([
...  {$match: {"CodigoR": "ESP"}},
...  {$group: {_id: "$CodigoP", count_mediciones_esp: {$count: {}}}},
...  ])
```

Figura 11: Código utilizado para resolver la consulta 2.

6.3.2. Explicación

Se trata de una consulta con estructura similar a la anterior. Al igual que antes, trabajamos sobre la colección Valor_anual, y se ha de agrupar, en este caso por niveles profesionales (*_id*: "\$CodigoP"), y a partir de ahí realizar un \$count sobre cada grupo (la columna que lleva la cuenta de mediciones se le ha llamado count_mediciones_esp). La mayor diferencia es que en esta consulta hay un limitante, y es que las mediciones se cuenten sólo sobre España (el código de región debe ser ESP). Lo que en PostgreSQL hubiéramos resuelto con un WHERE y un GROUP BY, en este caso se resuelve utilizando el operador \$match, que limita lo que se le ponga por valor: (\$match:{"CodigoR": "ESP"}).

6.3.3. Resultado

```
[
{ _id: 'ASSOCIATE_PROF', count_mediciones_esp: 306 },
{ _id: null, count_mediciones_esp: 3596 },
{ _id: 'ALL', count_mediciones_esp: 268 },
{ _id: 'PROFESSIONAL', count_mediciones_esp: 344 }
]
```

Figura 12: Salida de la consulta propuesta 2.

Observar que se obtienen dos columnas: *_id*, cuyas filas son cada tipo de nivel profesional, incluidos los valores null, y *count_mediciones_esp*.

Dado que el enunciado no especifica ningún tipo de orden, no se han ordenado las salidas.

6.3.4. Comprobación en PostgreSQL

Query	Query History
1	SELECT CódigoP, COUNT(Valor) AS count_mediciones_esp
2	FROM Valor_anual
3	WHERE códigoR = 'ESP'
4	GROUP BY CódigoP

Figura 13: Código utilizado para resolver la consulta guiada 2 en SQL.

	códigoP character varying (100)	count_mediciones_esp bigint
1	ALL	268
2	[null]	3596
3	ASSOCIATE_PROF	306
4	PROFESSIONAL	344

Figura 14: Salida de la consulta guiada 2.

Se ha podido comprobar que se obtiene el mismo resultado programando en lenguaje SQL y NoSQL.

6.4. Consulta 3

Obtén un listado del número de asociaciones de países a los que pertenece cada país. Ordena la lista en orden descendente (de mayor a menor número de asociaciones de países) y limita el resultado a los 10 países con el mayor número

6.4.1. Consulta en Mongo Shell

```
HLTHres> db.Pais.aggregate([
...  {$project: {_id:"$Nombre_completoP", Num_agrup:{$size:"$Agrup"}}},
...  {$sort:{Num_agrup:-1}},
...  {$limit:10}
... ])
```

Figura 15: Código utilizado para resolver la consulta 3.

6.4.2. Explicación

Como bien dice el enunciado, debemos distinguir por países, y a partir de ahí, contar a cuántas agrupaciones de países pertenece cada uno. Para ello, accedemos a la colección País, y aplicamos la operación *\$project*, que realiza una proyección sobre las columnas que deben salir por pantalla: primero el nombre completo de cada país, y después, la columna *Num_agrup*, que calcula la longitud de los arrays de *agrup* (asociaciones de países a las que pertenece cada país). Tras aplicar a la consulta dos filtrados más: *\$sort:{Num_agrup: -1}* y *\$limit:10*, obtendremos los 10 primeros países, ordenados de mayor a menor según *Num_agrup*.

6.4.3. Resultado

```
[
{ _id: 'Bulgaria', Num_agrup: 4 },
{ _id: 'Malta', Num_agrup: 4 },
{ _id: 'Sweden', Num_agrup: 4 },
{ _id: 'Denmark', Num_agrup: 4 },
{ _id: 'Finland', Num_agrup: 4 },
{ _id: 'Romania', Num_agrup: 4 },
{ _id: 'Luxembourg', Num_agrup: 4 },
{ _id: 'Cyprus', Num_agrup: 4 },
{ _id: 'Austria', Num_agrup: 3 },
{ _id: 'Estonia', Num_agrup: 3 }
]
```

Figura 16: Salida de la consulta propuesta 3.

Es de destacar que esta consulta en concreto ha resultado ser más fácil de lo esperado debido a la inserción previa de un nuevo campo llamado *Agrup*, que contiene a todos los países afiliados a una agrupación en concreto.

6.4.4. Comprobación en PostgreSQL

	Data Output	Notifications	Messages	Query	Query History
1	SELECT CódigoPa, COUNT(CódigoG) AS Num_agrup				
2	FROM Pertenece				
3	GROUP BY CódigoPa				
4	ORDER BY Num_agrup DESC				
5	LIMIT 10				

Figura 17: Código utilizado para resolver la consulta guiada 3 en SQL.

	códigopa character varying (100)	num_agrup bigint
1	SWE	4
2	DNK	4
3	FIN	4
4	MLT	4
5	BGR	4
6	CYP	4
7	ROU	4
8	LUX	4
9	HRV	3
10	FRA	3

Figura 18: Salida de la consulta guiada 3.

Se ha podido comprobar que se obtiene el mismo resultado programando en lenguaje SQL y NoSQL.

uh

Consulta 3

6.5.

Obtén un listado con todas las unidades de medida existentes (código) así como el número de indicadores que pertenecen a cada una de ellas.

6.5.1-6.4.5. Consulta en Mongo Shell

```
HLTHres> db.Indicador.aggregate([
...  {$group: {_id: "$Unidad", Indicadores: {$count: {}}}},
...  {$sort: {Indicadores: -1}}
... ])
```

Figura 19: Código utilizado para resolver la consulta 4.

6.5.2-6.4.6. Explicación

Tal y como exige el enunciado, el primer paso es agrupar por cada tipo de unidad de medida (cuya clave primaria es Unidad). Por ello, se utiliza el código `$group: {_id: "$Unidad", Indicadores: {$count: {}}}`.

Después, aunque no lo especifica el enunciado, hemos querido ordenar la lista de mayor a menor, utilizando el operador `$sort()` sobre la columna de `Indicadores`, creada para contar los tipos de Unidad.

6.5.3-6.4.7. Resultado

```
[
  { _id: 'NUM_PROFESS', Indicadores: 77 },
  { _id: 'PROFESS_100K', Indicadores: 70 },
  { _id: 'PERCENT', Indicadores: 24 },
  { _id: 'EQUIP_100K', Indicadores: 24 },
  { _id: 'NUM_EQUIP', Indicadores: 24 },
  { _id: 'BEDS_100K', Indicadores: 10 },
  { _id: 'NUM_BEDS', Indicadores: 10 },
  { _id: 'FACIL_100K', Indicadores: 5 },
  { _id: 'NUM_FACIL', Indicadores: 5 },
  { _id: 'NUM_PERS', Indicadores: 1 }
]
```

Figura 20: Salida de la consulta propuesta 4.

Con formato: Fuente de párrafo predeter., Fuente: (Predeterminada) +Cuerpo (Calibri), 11 pto

Con formato: Normal, Sin viñetas ni numeración

6.5.4-6.4.8. Comprobación en PostgreSQL

Query	Query History
1	SELECT Unidad, COUNT (Unidad) AS Indicadores
2	FROM Indicador
3	GROUP BY Unidad
4	ORDER BY Indicadores DESC

Figura 21: Código utilizado para resolver la consulta guiada 4 en SQL.



	unidad character varying (100) 	indicadores bigint 
1	NUM_PROFESS	77
2	PROFESS_100K	70
3	NUM_EQUIP	24
4	PERCENT	24
5	EQUIP_100K	24
6	NUM_BEDS	10
7	BEDS_100K	10
8	NUM_FACIL	5
9	FACIL_100K	5
10	NUM_PERS	1

Figura 22: Salida de la consulta guiada 4.

Se ha podido comprobar que se obtiene el mismo resultado programando en lenguaje SQL y NoSQL.

7. CONSULTA EXTRA

Se ha querido hacer una consulta extra, por la curiosidad de probar cómo funciona el método \$text. Por ello, se ha realizado una consulta sencilla, que sea contar cuántos indicadores referidos a Hospitales hay por cada tipo de Unidad.

7.1. Consulta en Mongo Shell

```
HLTHres> db.Indicador.dropIndexes()
{
  nIndexesWas: 3,
  msg: 'non-_id indexes dropped for collection',
  ok: 1
}
HLTHres> db.Indicador.createIndex({DescripcionI:"text"})
DescripcionI_text

HLTHres> db.Indicador.aggregate([
... {$match : { $text: { $search: "Hospitals" }}} ,
... {$group: {_id: "$Unidad" , count_tipos : {$count: {}}}}
... ])
```

Figura 23: Código utilizado para resolver la consulta extra.

7.2. Explicación

Antes de poder utilizar el método \$text, se debe crear un índice, y previamente se han eliminado los creados para evitar errores. Entonces, se especifica que el índice consta de {DescripcionI: "text"}, para poder utilizar el método \$text en dicha columna. Después, se ha utilizado un aggregate, que filtra primero aquellos indicadores en cuya descripción aparece la palabra "Hospitals" (debido a que MongoDB recurre a stemming, buscará todas aquellas descripciones con palabras derivadas de "hospital". Después, se agrupa por Unidad, y se aplica el método \$count para contar cuántos indicadores corresponden a cada tipo de Unidad.

Además, es interesante darse cuenta de que todas las consultas propuestas se han podido realizar también en SQL, salvo esta última, ya que SQL no cuenta con un método que busque palabras sueltas.

7.3. Resultado

```
[
  { _id: 'NUM_PROFESS', count_tipos: 16 },
  { _id: 'FACIL_100K', count_tipos: 5 },
  { _id: 'BEDS_100K', count_tipos: 6 },
  { _id: 'PERCENT', count_tipos: 10 },
  { _id: 'NUM_BEDS', count_tipos: 6 },
  { _id: 'NUM_EQUIP', count_tipos: 8 },
  { _id: 'PROFESS_100K', count_tipos: 15 },
  { _id: 'EQUIP_100K', count_tipos: 8 },
  { _id: 'NUM_FACIL', count_tipos: 5 }
]
```

Figura 24 Salida de la consulta propuesta extra.

8. CONCLUSIONES

Esta última práctica ha permitido al alumno poner en práctica lo aprendido durante todo un curso de Gestión de Datos Biomédicos, ya que, al poder aplicar la teoría dada, el alumno comprende de mejor manera cómo optimizar las búsquedas y consultas de los datos, y de qué manera deben entenderse las relaciones de datos en NoSQL en comparación con SQL. Las consultas son una de las razones principales por las que se utilizan las bases de datos, por lo que esta práctica ha sido crucial para el alumno. Además, se ha podido comprobar que MongoDB es un lenguaje de programación bastante intuitivo, y que con las herramientas adecuadas funciona perfectamente para aplicar los conocimientos vistos recientemente en clase, en casos reales.

Valoración personal de MongoDB

La sintaxis de mongo es mucho menos intuitiva que la de SQL, teniendo que estar usando “{}” para parámetros y valores. Por otra parte, los documentos para insertar son mucho más legibles que los de SQL, tienen una sintaxis de json con sus claves, valores y la posibilidad de tener documentos embebidos. Sin embargo, aunque uno de los principales motivos para usar bases de datos no SQL es para evitar los JOINS, esto viene pagando un precio, usar documentos embebidos, los cuales complican la inserción de los datos según la tabla de datos con la que se parte. Aunque MongoDB ofrece algunas mejoras en cuanto a cómo elaborar consultas, sobre todo con el parámetro \$text, una cosa que no tiene SQL y es muy versátil, la sintaxis, la inserción y el uso de comandos hacen que personalmente prefiramos SQL.

Si bien es cierto que con SQL utilizamos un entorno de desarrollo especializado en ese lenguaje y no usamos, al igual que en MongoDB, la Shell, los comandos en SQL son más “limpios” a plena vista que los de Mongo. La cuestión es acostumbrarse a poner las {} y los parámetros para hacer diferentes consultas, pero habiendo aprendido primero a usar SQL, el paso a mongo se puede comparar como, una vez ya programando a alto nivel, te pasas a programar a bajo nivel. Todo funciona también por comandos, pero la presentación y la ejecución varían ligeramente.

En conclusión, MongoDB, pese a nuestras dificultades y preferencias personales, es un lenguaje útil de saber manejar y entender al igual que SQL.