Comparando Bases de Datos para insertar y consultar datos de tipo JSON

Rebeca Fernandez, Evelyn Kremer
Instituto de Computación
Facultad de Ingeniería, Universidad de la República
Montevideo, Uruguay

***Resumen*—El siguiente documento presenta un análisis comparativo en base a tiempos de respuesta y eficiencia de insertar y consultar datos de tipo JSON[1] en una base de datos documental, específicamente MongoDB; contra una base de datos relacional, más específicamente PostgreSQL. A partir de la importación del conjunto de datos hacia ambos motores, se realizan consultas para evaluar los tiempos de respuesta y pruebas de performance utilizando JMeter como herramienta. De los resultados obtenidos se concluye que MongoDB presenta mejoras en tiempos de respuesta de consultas, especialmente a medida que la carga aumenta. Sin embargo, en operaciones individuales de actualización, PostgreSQL genera mejores resultados.**

I. Introducción

La motivación de este trabajo surge a partir de que PostgreSQL (desde su versión 9.2) ofrece una aceleración significativa cuando se usa la representación binaria de datos JSON, denominada *jsonb*, que puede brindar una ventaja significativa para mejorar el rendimiento. Algunos de los beneficios inmediatos que trae esta nueva representación son: más eficiencia, ya que admite indexación y diseños de esquemas más simples (reemplazando tablas de valor de atributo de entidad (EAV) con columnas *jsonb*, que se pueden consultar, indexar y unir). Lo que se propone es estudiar un conjunto de datos de tipo JSON, almacenar su estructura siguiendo dos diseños distintos en PostgreSQL utilizando esta nueva representación; y también almacenarlos en MongoDB. Posteriormente, se espera realizar ciertas consultas sobre los datos en ambas bases y evaluar su comportamiento según tiempos de respuesta y eficiencia. Finalmente, se desea comprobar, si puede ser más eficiente utilizar este tipo de representación nueva que ofrece PostgreSQL en vez de utilizar una base de datos documental como MongoDB.

El documento se organiza de la siguiente manera. En la primera sección I se plantea el tema principal de estudio y los objetivos de la investigación, luego en la sección II se agrega información sobre los estudios antecedentes sobre la temática. La siguiente sección III presenta el juego de datos elegido para analizar, y cómo se modeló su estructura en las bases elegidas. Por último, en la sección IV se indican las consultas realizadas sobre ambos motores y se presentan los resultados obtenidos para cada una de ellas y en la sección V se desarrollan las conclusiones a partir de los resultados obtenidos.

[1]JSON - acrónimo de JavaScript Object Notation

II. Trabajos relacionados y contexto

Al tratarse de un tema relativamente nuevo en las bases relacionales y que viene mejorando en los últimos años, existen algunos trabajos relacionados sobre comparativa de performance de bases de datos SQL y NoSQL para almacenar este tipo de datos, especialmente en el área de BigData - que se ha convertido en uno de los principales campos de investigación. A partir de algunos trabajos que comparan sistemas de gestión de bases de datos SQL y NoSQL, enofocados en PostgreSQL y MongoDB respectivamente [3], se concluye que en general en dicho contexto, MongoDB muestra una mejora en rendimiento. Sin embargo, algunas operaciones de PostgreSQL se pueden mejorar mediante la indexación. Este estudio plantea profundizar más en esas operaciones para evaluar si en algunos casos corresponde utilizar un enfoque relacional.

JSON se convirtió en el estándar para almacenar datos no estructurados en un formato de texto sin formato. Estos datos no estructurados se pueden utilizar para representar datos más complejos que el estándar, fila unidimensional en bases de datos relacionales tradicionales. Por esa razón, JSON es el formato más utilizado en bases de datos Big Data. Además, por la misma razón, las bases de datos tradicionales también están implementando una forma de almacenar objetos JSON, tratando de estar a la par con las nuevas tecnologías NoSQL. En PostgreSQL, la definición de este nuevo formato de representación es muy simple como definir el tipo de campo como JSON.

Por ejemplo, la declaración de creación `CREATE` para una tabla `orders` con id como primary key e información como json se presenta a continuación:

```
CREATE TABLE orders (ID serial NOT NULL PRIMARY KEY,
   info json NOT NULL);
```

La inserción de datos en formato JSONs es también muy simple, implica solamente interpretarlo como una cadena de caracteres de la misma manera que insertando cualquier otro campo en una declaración `INSERT`, por ejemplo

```
INSERT INTO orders (info)
VALUES('{"customer":"John␣Doe","items":{"product":"Beer
     ","qty":6}}');
```

Las consultas sobre los campos del JSON se pueden realizar de varias formas. La consulta

```
SELECT info FROM orders;
```

retorna todos los JSONs en la tabla. Utilizando la palabra clave "->", por ejemplo

```
SELECT info -> 'customer' AS customer FROM orders;
```

tendremos todos los clientes de los JSONs en la tabla en formato JSON. También se puede usar la cláusula `WHERE` en los campos internos del JSON, por ejemplo la consulta

```
SELECT info ->> 'customer' AS customer FROM orders
WHERE info -> 'items' ->> 'product' = 'Diaper';
```

retorna como resultado que cliente compró "Diaper". También se admiten agregaciones, así como algunos operadores avanzados [5].

III. Escenario a estudiar

Conjunto de datos Para el análisis que se quiere realizar, se eligió el conjunto de datos de prueba Last.fm Million [6]. Para el análisis se eligió un subconjunto más reducido de los datos con un total de 104.212 archivos JSON debido a la capacidad de almacenamiento limitada.

Los archivos JSON que representan una pista tienen seis campos diferentes:

1. `artist`, representa el artista que interpreta la canción.
2. `timestamp` representa la fecha y la hora en que la canción fue creada (insertada en la base)
3. `similars` contiene la lista de pares "key" y "value" que representan el "track_id" y el valor precalculado de la medida de similitud entre la canción y otras canciones
4. `tags` es una lista de pares "key" y "value" de etiquetas de la canción.
5. `track_id` es el identificador único de la canción en la base de datos Last.fm
6. `title` es el título de la canción

```
{
   "artist": "Casual",
   "timestamp": "2011-08-02 20:13:25.674526",
   "similars": [
      {"key": "TRABACN128F425B784", "value": 0.871737},
      {"key": "TRIAINV12903CB4943", "value": 0.751301},
   ],
   "tags": [
      {"key": "Bay Area", "value": "100"},
      {"key": "hieroglyiphics", "value": "100"}
   ],
   "track_id": "TRAAAAW128F429D538",
   "title": "I Didn't Mean To"
}
```

Ejemplo de pista con sus 6 campos asociados

Creación de las bases de datos En MongoDB se creó una colección llamada "track" y en PostgreSQL se optó por modelar los datos de dos maneras diferentes.

En el primer modelo se creó una tabla llamada "track" con una columna "id" de tipo serial y una columna "track_data" de tipo jsonb donde se insertó el JSON, en adelante referido como *PostgreSQL v1*.

En el segundo modelo se creó una tabla con una columna para cada uno de los campos del json, excepto para los campos "similars" y "tags" que se definieron de tipo jsonb y se insertó el arreglo de objetos, en adelante referido como *PostgreSQL v2*.

Para que la comparación fuera justa y aprovechando el rendimiento de los índices en JSONB de PostgreSQL se definieron índices para los campos "track_id", "similars" y "tags", tanto en MongoDB como PostgreSQL en ambas versiones.

Inserción de datos Para realizar las consultas sobre este conjunto de datos elegido, previamente se realizó la carga en las bases de datos MongoDB y PostgreSQL. Al estar cada pista representada en formato JSON en un archivo, se implementaron pequeños programas en Python para la carga de cada uno. Además, los campos "similars" y "tags" en los archivos no están representados como arreglo de objetos, con lo cual se tuvo que hacer una mínima transformación para que quedaran así representados.

Ambiente de prueba Las pruebas se ejecutaron en un 11th Gen Intel(R) Core(TM) i7-1165G7, 2.80GHz, 16.0 GB de memoria, 512GB de disco y el sistema operativo Windows 10. PostgreSQL versión 13 (actual, lanzada en setiembre del 2020). MongoDB Server versión 4.4. Y los programas de carga se ejecutaron con Python 3.8.10.

IV. Experimentación

IV-A. Herramienta y operaciones

Las pruebas se realizaron utilizando la herramienta Apache JMeter [2] al ser apropiada para el contexto de estudio y utilizarse con frecuencia en la actualidad. Se utilizó la versión 5.4.1.

El interés principal está en realizar consultas sobre los datos almacenados de tipo JSON. En base a esto, se realizaron las siguientes operaciones de consulta y actualización:

1. Obtener los datos de una pista por su track_id.
2. Obtener todas las pistas que tienen una etiqueta específica.
3. Obtener todas las pistas que son similares a una pista específica.
4. Obtener la cantidad de pistas que son similares a cada una de las pistas.
5. Actualizar la lista de etiquetas de una pista específica, que no tiene etiquetas, estableciendo solamente una etiqueta.
6. Actualizar el valor de similitud de una pista específica y una pista similar a ella.

Se realizaron las mismas operaciones tanto para medir los tiempos de respuesta como para realizar las pruebas de rendimiento. La diferencia está en que para las últimas se realizaron cien ejecuciones de las operaciones concurrentemente, en vez de una única ejecución.

IV-B. Resultados obtenidos

Tiempos de respuesta en una ejecución En la Tabla I y en la Figura 1 se muestran los tiempos de respuesta promedio para los tres escenarios antes mencionadas y con una única ejecución (medido en milisegundos).

Tiempo de respuesta promedio (ms)			
	PgSQL v1	PgSQL v2	MongoDB
Consulta 1	463	451	76
Consulta 2	424	205	366
Consulta 3	820	708	21
Consulta 4	4627	4210	6382
Update 1	4	2	48
Update 2	1	1	21

Cuadro I: Tiempos de respuesta en una única ejecución

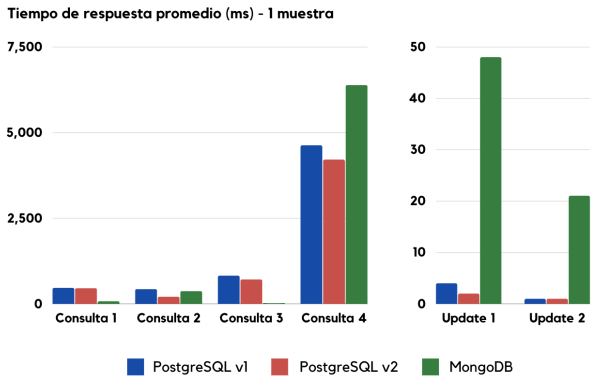

Figura 1: Gráfica de tiempos de respuesta de una única ejecución

De las operaciones de consulta, se observa una notoria variabilidad sobre MongoDB, mientras que ambos modelos de PostgreSQL se mantienen proporcionales a la complejidad de la consulta y similares entre sí, aunque la segunda versión presenta leves mejoras. A excepción de la última consulta, MongoDB presenta mejores tiempos en una proporción considerablemente grande.

Sobre las operaciones de actualización, hay una notoria diferencia entre los tiempo de respuesta de PostgreSQL respecto de las de MongoDB.

Pruebas de performance En la Tabla II y en la Figura 2 se muestran los resultados obtenidos medidos con cien ejecuciones.

De las operaciones de consulta se observa que ambos modelos de PostgreSQL se comportan muy parecidos y se observa que en la consulta cuatro ambos son mejores en tiempos que MongoDB. Sin embargo, para las operaciones de actualización se observa que MongoDB tiene un tiempo considerablemente mejor que PostgreSQL contrariamente a lo que sucede cuando se ejecuta una única actualización. Esto último, puede estar relacionado a como PostgreSQL y MongoDB manejan las conexiones concurrentes. En PostgreSQL el rendimiento depende en gran medida de la cantidad de usuarios simultáneos (conexiones). Cuando se ajusta al número óptimo de conexiones, funciona muy bien pero el rendimiento cae si está abrumado por las conexiones.

Tiempo de respuesta promedio (ms)			
PostgreSQL versión 1			
	mínimo	promedio	máximo
Consulta 1	2153	2879,96	3225
Consulta 2	24583	47010.32	58699
Consulta 3	53694	124690.14	145927
Consulta 4	152065	230594.49	265086
Update 1	2083	2743.60	3087
Update 2	1939	2816.61	3212

PostgreSQL versión 2			
	mínimo	promedio	máximo
Consulta 1	1787	2713,78	3105
Consulta 2	6574	18441,14	23525
Consulta 3	51518	111217,58	128717
Consulta 4	132500	219012,13	253573
Update 1	2041	2855,23	3168
Update 2	1950	2718,69	3076

MongoDB			
	mínimo	promedio	máximo
Consulta 1	119	135.54	152
Consulta 2	8640	10014.24	11203
Consulta 3	134	285.26	410
Consulta 4	511059	584835.53	804008
Update 1	130	143.72	155
Update 2	118	138.10	156

Cuadro II: Tiempos de respuesta en cien ejecuciones

En ambas pruebas, se observa que para la cuarta consulta MongoDB presenta mejores tiempos en una proporción mayor (sobretodo en las pruebas de rendimiento donde es más del doble). Podría estar relacionado al hecho de que para agrupar por claves, en MongoDB primero se realiza una operación `unwind` al estar agrupando sobre un arreglo de objetos descompone cada campo para generar un documento para cada elemento, y el resultado se agrupa aplicando `grouping`. En cambio, la operación `GROUP BY` que ofrece PostgreSQL convierte el arreglo de objetos con `jsonb_to_recordset`

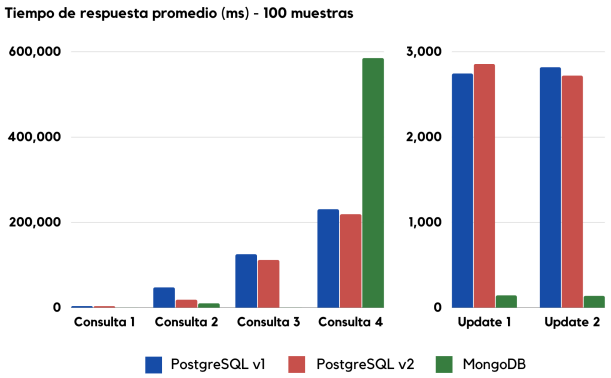

Figura 2: Resultados obtenidos en cien ejecuciones

en un conjunto de registros, lo cual es posible que el modo en que opera según los índices o en orden de ejecución dicha consulta sea más eficiente que el enfoque de MongoDB.

Código desarrollado El código desarrollado está disponible en un repositorio de Gitlab [1]. El repositorio contiene los programas en Python necesarios para realizar la carga de datos y los proyectos de JMeter con las consultas ejecutadas para la experimentación.

V. Conclusiones y trabajo futuro

En base a los recabado de los trabajos relacionados, se esperaba que los tiempos de respuesta de MongoDB fuesen a grandes razgos, más óptimos que los resultados de PostgreSQL y para los experimentos realizados, y los juegos de datos se obtuvo un resultado consistente en este sentido. Las operaciones donde este patrón no se cumplió fueron para las actualizaciones donde se realizó una única ejecución de la operación y en la consulta sobre la obtención de la cantidad de pistas que son similares a cada una de las pistas.

En términos generales, MongoDB resultó ser más eficiente que PostgreSQL, principalmente en consultas más sencillas, y su rendimiento es mejor en consultas concurrentes. Sin embargo, creemos que PostgreSQL puede ser mejor, o no, dependiendo fuertemente de los tipos de consultas que se quieren realizar sobre los datos y sobre el modelo de la realidad con el cual se está trabajando. En particular, para el escenario que se analizó, la mayoría de las consultas y actualizaciones que se realizaron fueron sobre datos que estaban representados como arreglos de objetos, lo cual puede dar lugar a pensar que los resultados obtenidos dependieron en gran medida por esto último.

En casos donde se pueda considerar el uso de PostgreSQL para almacenar JSON el determinante puede estar en si el sistema ya usa una base de datos PostgreSQL y por lo tanto se puede aprovechar esta nueva representación que ofrece y no utilizar una base de datos no relacional como MongoDB.

Por último, el estudio está centrado únicamente en dos herramientas. Como trabajo futuro, sería interesante poder combinar otros motores escalables tales como las recomendadas por el artículo en que se basa el trabajo [4]: *Cassandra* y *Elastic Search*. Al igual que los analizados, ambos se encuentran en crecimiento en la industria actualmente y en particular sobre Elastic Search sería interesante investigar los aspectos técnicos en la creación de índices que permiten mejorar los tiempos de búsqueda para un gran volumen de datos. Por otro lado, las bases de datos de datos de grafos tales como *Neo4j* plantearían un enfoque distinto de estructurar los datos que se podría abordar para ampliar y enriquecer la comparación. También se podría experimentar con un mayor volumen de datos.

Referencias

[1] Código desarrollado. https://gitlab.fing.edu.uy/evelyn.kremer/bdnr2021g4. [accessed: 2021-07-21].
[2] Herramienta apache jmeter. https://jmeter.apache.org/. [accessed: 2021-07-18].
[3] M. Jung, S. Youn, J. Bae, and Y. Choi. A study on data input and output performance comparison of mongodb and postgresql in the big data environment. In: 8th International Conference on Database Theory and Application (DTA). pages 14–17, 2015.
[4] B. I. P. Ribarski and B. Tojtovska. Comparing Databases for Inserting and Querying Jsons for Big Data. pages 1–9, 2019.
[5] Postgresql json functions and operators. https://www.postgresql.org/docs/current/functions-json.html. [accessed: 2019-07-22].
[6] The last.fm dataset. http://millionsongdataset.com/lastfm/index.html. [accessed: 2021-07-17].

APÉNDICE

A continuación se presenta el código de las operaciones de consulta y actualización ejecutadas durante la experimentación de PostgreSQL v1, PostgreSQL v2 y MongoDB respectivamente:

1. Obtener los datos de una pista por su track_id.

```
select * from track where track_data ->> 'track_id
    ' = 'TRAAETD128F92F89A3'
```

```
select * from track where track_id='
    TRAAETD128F92F89A3'
```

```
collection.find(eq("track_id", "TRAAETD128F92F89A3
    ")).first()
```

2. Obtener todas las pistas que tienen una etiqueta específica.

```
select * from track, jsonb_to_record(track.
    track_data) as items(tags jsonb),
    jsonb_to_recordset(tags) as items_tags(key
    text) where items_tags.key = 'post-hardcore'
```

```
select * from track, jsonb_to_recordset(track.tags
    ) as items(key text) where items.key = 'post-
    hardcore'
```

```
collection.find(eq("tags.key", "post-hardcore"))
```

3. Obtener todas las pistas que son similares a una pista específica.

```
select * from track, jsonb_to_record(track.
    track_data) as items(similars jsonb),
    jsonb_to_recordset(similars) as
    items_similars(key text) where items_similars
    .key = 'TRAADXK12903CFAE92'
```

```
select * from track, jsonb_to_recordset(track.
    similars) as items(key text) where items.key
    = 'TRAADXK12903CFAE92'
```

```
collection.find(eq("similars.key", "
    TRAADXK12903CFAE92"))
```

4. Obtener la cantidad de pistas que son similares a cada una de las pistas.

```
select items_similars.key, count(*) from track,
    jsonb_to_record(track.track_data) as items(
    similars jsonb), jsonb_to_recordset(similars)
     as items_similars(key text) group by
    items_similars.key
```

```
select items.key, count(*) from track,
    jsonb_to_recordset(track.similars) as items(
    key text) group by items.key
```

```
Document unwind = new Document("path", "\$similars
    ").append("preserveNullAndEmptyArrays", true)
    ;
 Document grouping = new Document("_id", "\
     $similars.key").append("count", new Document(
     "\$sum", 1));
 MongoCursor<Document> cursor = collection.
     aggregate(asList(new Document("\$unwind",
     unwind), new Document("\$group", grouping)))
     .iterator();
```

5. Actualizar la lista de etiquetas de una pista específica, que no tiene etiquetas, estableciendo solamente una etiqueta.

```
update track set track_data = jsonb_set(track_data
    , '{tags}', '[{"key":␣"dubstep","value":␣
    "100"}]') where track_data ->> 'track_id' = '
    TRDVTAI128F93409CA'
```

```
update track set tags = jsonb_set(tags, '{0}', '{"
    key":␣"dubstep","value":␣"100"}') where
    track_id = 'TRDVTAI128F93409CA'
```

```
BasicDBObject tag = new BasicDBObject("key", "
    dubstep");
 tag.put("value", "100");
 collection.updateOne(eq("track_id", "
     TRDVTAI128F93409CA"), set("tags", Arrays.
     asList(tag)));
```

6. Actualizar el valor de similitud de una pista específica y una pista similar a ella.

```
update track set track_data = jsonb_set(track_data
    , '{similars,␣2,␣value}', '1') where
    track_data ->> 'track_id' = '
    TRDVTAI128F93409CA'
```

```
update track set similars = jsonb_set(similars, '
    {2,␣value}', '1') where track_id = '
    TRDVTAI128F93409CA'
```

```
collection.updateOne(and(eq("track_id", "
    TRDVTAI128F93409CA"), eq("similars.key", "
    TRHHYGW128F930C1BA")), set("similars.\$.value
    ", "1"))
```