Revista Cubana de Ciencias Informáticas
Vol. 10, No. Especial Informática 2016, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 70-81

Tipo de artículo: Artículo original
Temática: Tecnologías de bases de datos
Recibido : 15/04/2016 | Aceptado: 05/05/2016

Características no relacionales de PostgreSQL: incremento del rendimiento en el uso de datos JSON

No relational features of PostgreSQL: increase performance in JSON types use

Yudisney Vazquez Ortíz[1*], Lisleydi Mier Pierre[2], Anthony R. Sotolongo León[3]

[1] DATEC-UCI, Carretera a San Antonio de los Baños, Km 2 ½, La Lisa, La Habana. yvazquezo@uci.cu
[2] FORTES-UCI, Carretera a San Antonio de los Baños, Km 2 ½, La Lisa, La Habana. lmpierre@uci.cu
[3] ARKADIOS K&T, Santiago de Chile, Chile. asotolongo@gmail.com

[*] Autor para correspondencia: yvazquezo@uci.cu

Resumen

El empleo de datos semiestructurados para el almacenamiento de la información ha sido una tendencia en los últimos años. Situación por la que han surgido gestores de bases de datos especializados en manipular estos datos, los que no implementan el modelo relacional, hasta el momento ampliamente utilizado. Estas soluciones se agruparon bajo el nombre de gestores "no relacionales" o NoSQL; los que ofrecen características como escalabilidad y velocidad en sus tiempos de respuesta, superiores a las que los sistemas relacionales podían ofrecer. No obstante, los gestores NoSQL no se solapan con los relacionales ya que cada tipo garantiza las funcionalidades para las que fueron desarrollados, motivo por el que varias empresas los utilizan juntos. PostgreSQL ha ido incorporando gradualmente características NoSQL, lo que puede constituir una ventaja, sobre todo porque permite prescindir de una tercera herramienta. Uno de los pasos en este sentido fue la inclusión, en su versión 9.4, del tipo de dato JSONB, además del ya existente JSON. El propósito de este artículo es realizar un experimento para evaluar el comportamiento de PostgreSQL haciendo uso de ambos tipos de datos, respecto a sus tiempos de respuesta. Para ello se realizaron pruebas de rendimiento sobre la carga de los datos y consultas, determinándose que con la implementación de JSONB, PostgreSQL está avanzando en la implementación de características no relacionales, convirtiéndose en una opción a ser considerada.

Palabras claves: características no relacionales de PostgreSQL; json; jsonb

Revista Cubana de Ciencias Informáticas
Vol. 10, No. Especial Informática 2016, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 70-81

Abstract

The use of semi structured data for information storage has been a trend in the last years. Situation for which are emerging database management systems specialized in manipulate them, that not implement the relational model, widely used until now. These solutions were pooled under the name of "no relational" systems or NoSQL, which offer features like scalability and agility of its response times, higher than relational systems could offer. However, these databases do not overlap with relational types, because each one guarantees the functionalities for which they were developed, reason why many enterprises use them together. PostgreSQL has been included, gradually, NoSQL capabilities, which could be an advantage especially because it could be dispensed of a third tool. One of the steps for the gradual incorporation of these capabilities was the inclusion, in his 9.4 version, of the JSONB type, besides the existing JSON. The purpose of this paper is make an experiment to evaluate the PostgreSQL behavior using both types, respect to their response times. For this, were made benchmarking tests about data loads and selects, determining that with the JSONB implementation, PostgreSQL is moving in the NoSQL capabilities implementation, been an option to be considered.

Keywords: *json; jsonb; PostgreSQL no relational features*

Introducción

El empleo de datos semiestructurados en los sistemas de almacenamiento ha sido una tendencia en los últimos años; surgiendo gestores de bases de datos especializados en manipularlos, que no implementan el modelo relacional hasta el momento ampliamente utilizado (Tauro, y otros, 2012).

Estas soluciones se agruparon bajo el nombre de gestores "no relacionales" o NoSQL, que ofrecen características como la escalabilidad y velocidad en sus tiempos de respuesta y la variabilidad de sus esquemas, superiores a las que los sistemas relacionales podían ofrecer (Leavitt, 2010), (Han, y otros, 2011), (Pokorny, 2013), (Cattell, 2010).

Estas características las logran, principalmente, al no implementar el modelo relacional (no garantizan las propiedades ACID, no utilizan tablas para el almacenamiento de los datos y no usan SQL como lenguaje de consultas), no permitir concatenaciones y tener una arquitectura distribuida, lo que incide favorablemente en sus objetivos de flexibilizar los métodos de manipulación de los datos para ganar en rapidez frente al alto volumen de datos generado en la actualidad (Moniruzzaman, y otros, 2013), (Strauch, 2013), (Linux-Magazine, 2009).

Tal proliferación ha favorecido la diversidad de herramientas (NoSQL, 2016). Algunas de ellas almacenan los datos como documentos (generalmente sobre estructuras JSON o XML), definiendo una clave única para cada registro y

permitiendo la realización de búsquedas por clave-valor, el indexado por sus propiedades y la ejecución de consultas más complejas sobre el contenido del documento, razones por las que son empleadas considerablemente (Tiwary, 2011).

Independientemente de las ventajas que ofrecen, los gestores NoSQL no se solapan con los relacionales, ya que cada tipo garantiza las funcionalidades para las que fueron desarrollados; motivo por el que varias empresas los utilizan juntos para diferentes actividades, tal es el caso de *Facebook* y *Tumblr*, entre otros (Tumblr, 2012).

PostgreSQL, ha ido incorporando gradualmente características NoSQL para brindar mayores opciones a sus usuarios, acercándose desde la versión 9.3 a los tiempos de respuesta ofrecidos por MongoDB, base de datos documental ampliamente utilizada (DB-engines-trend, 2015), según experimento realizado (Sotolongo León, y otros, 2013).

Características que se amplían en la versión 9.4, afirma la *EnterpriseDB*, mejorando su rendimiento considerablemente con la incorporación del tipo de dato JSONB, que se diferencia del ya existente JSON fundamentalmente en la eficiencia (EnterpriseDB, 2014).

Esta situación puede constituir una ventaja, sobre todo porque pudiera prescindirse, en la medida en que el gestor pueda suplir las necesidades de las empresas, de una tercera herramienta para garantizar las funcionalidades requeridas por estas al emplear ambos tipos de gestores.
Producto al uso que está teniendo JSON para el almacenamiento de los datos en las soluciones NoSQL, y que el estudio realizado sobre su versión 9.3 concluyó que PostgreSQL se estaba acercando a los tiempos de respuesta de MongoDB, sería bueno determinar qué tanto ha mejorado PostgreSQL con la implementación de JSONB y, por tanto, si sigue acortando la distancia, en materia de características no relacionales, con la puntera. De ahí que el objetivo de esta investigación sea evaluar su comportamiento haciendo uso de los tipos de datos JSON y JSONB.

Métodos para la evaluación del comportamiento de PostgreSQL

Para determinar la disponibilidad de PostgreSQL para ser empleado en soluciones no relacionales, se emplearon sus características NoSQL implementadas.

Características NoSQL implementadas en PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional que, gracias a su extensibilidad, ha permitido la incorporación de nuevas funcionalidades encaminadas a agilizar y flexibilizar la manipulación de los datos, entre las que destacan el almacenamiento efímero y los tipos de datos HSTORE y JSON.

Los tipos de datos HSTORE y JSON proveen opciones de gestión de datos sin esquema con la ventaja de cumplir con las propiedades ACID, permitiéndole al gestor dar soporte a aplicaciones que requieran flexibilidad en el modelo de datos (EnterpriseDB, 2014).

HSTORE fue añadido en la versión 8.2 como un módulo del *Contrib* que implementa un tipo de datos, con el mismo nombre, para almacenar pares de clave-valor dentro de un único valor de PostgreSQL; útil en situaciones como filas con muchos atributos raramente examinados o semiestructurados (PGDG, 2015).

El almacenamiento efímero fue añadido en la versión 9.1 mediante la implementación de tablas UNLOGGED, opción especificada durante la creación de las tablas que detalla que los datos almacenados en ellas no se escribirán en los WAL, lo que las hace considerablemente más rápidas que las tablas ordinarias, pero no garantiza la permanencia de los datos en caso de fallas (se pierde la propiedad Durabilidad) (PGDG, 2015).

JSON fue añadido en la versión 9.2 para dar soporte al almacenamiento de datos en dicho formato y garantizar su validación. Desde entonces ha sido mejorado en las versiones posteriores, añadiéndose en la 9.3 funciones para el trabajo con este y, en la 9.4 el tipo de dato JSONB con una mejora sobre el ya existente en cuanto a eficiencia (al no tener que analizarse durante cada ejecución las funciones y soportar el indexado) (PGDG, 2015).

Con estas características PostgreSQL permite:

- El almacenamiento de datos de forma ágil.
- El almacenamiento de datos libres de esquema.
- La lectura de datos relacionales de una tabla y su retorno como JSON y viceversa, haciendo uso de sus operadores y funciones (EnterpriseDB, 2014).
- La integración fácil de sentencias convencionales SQL con tipos de datos JSON y HSTORE, ejecutadas en el mismo entorno transaccional ACID y basados en el mismo planificador de consultas, optimizador y tecnologías de indexado (EnterpriseDB, 2014).

Elementos que lo convierten en una opción válida para el desarrollo de aplicaciones NoSQL.

Diseño de las pruebas de rendimiento entre PostgreSQL y MongoDB

Al ser JSON uno de los formatos de intercambio de datos más populares de la web, soportado por varias bases de datos NoSQL (incluida MongoDB), la evaluación del comportamiento del gestor frente a MongoDB, para determinar su avance en las posibilidades brindadas en materia de capacidades NoSQL, se realiza haciendo uso de este.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. Especial Informática 2016, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 70-81

En la siguiente tabla se definen los indicadores y métricas evaluados durante la ejecución de las pruebas de comparación, por ser elementos esenciales en la determinación del rendimiento del servidor de bases de datos.

Tabla 1. Indicadores y métricas definidos para las pruebas entre PostgreSQL y MongoDB

Indicador	**Métricas**	**Unidad de medida**
Tamaño de la base de datos	Tamaño de la base de datos	Mb
Tiempo de respuesta del servidor de bases de datos	Carga de datos	Milisegundos
	Consultas de selección de datos	Milisegundos

Para la realización de las pruebas:

- Se utilizaron instancias de un servidor de bases de datos PostgreSQL 9.4 y de un servidor de bases de datos MongoDB 3.0.2, ambos con su configuración inicial.
- Se utilizó una estación de trabajo con CPU Intel Core i5 a 2.8GHz, 4Gb de RAM, 500Gb de disco duro a 7200rpm y sistema operativo Windows 8 Pro.
- Se creó en PostgreSQL una base de datos db_generar donde se generaron los 1, 2 y 4 millones de registros JSON; para ello:
  - Se diseñó un tipo de dato JSON con la estructura:

```
"persona" {
        "identificador": "valor",
        "cuenta": {
                "usuario": "valor",
                "contrasenna": "valor",
                "correo_electronico": "valor",},
        "perfil": {
                "nombre_apellidos": "valor",
                "edad": "valor",
                "ciudad_vive": "valor",
                "estudios": "valor",
                "ocupacion": "valor",},
}
```

  - Se implementó la función generar_json() donde se utilizó la función *row_to_json()* para la generación de los registros JSON con el fragmento de la consulta siguiente, a ser ejecutada tantas veces como registros se requieran:

Revista Cubana de Ciencias Informáticas
Vol. 10, No. Especial Informática 2016, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 70-81

```
SELECT row_to_json(r)
FROM (
SELECT c AS cuenta, p AS perfil
FROM (
  SELECT "Usuario" || round((random()*100)::numeric,0)::text AS usuario,
  "Clave' || round((random()*100)::numeric,0)::text AS contrasenna,
  "correo" || round((random()*100)::numeric,0)::text || "@electronico.cu" AS correo_electronico) c,
  (SELECT "Nombre" || round((random()*100)::numeric,0)::text || " Apellido" ||
      round((random()*100)::numeric,0)::text AS nombre_apellidos,
  round((random()*100+1)::int,0) AS edad,
  "Ciudad donde vive" || round((random()*100)::numeric,0)::text AS ciudad_vive,
  "Estudios" || round((random()*100)::numeric,0)::text AS estudios,
  "Labor" || round((random()*100)::numeric,0)::text AS ocupacion) p
) r';
```

- Se crearon (en db_generar) las tablas tb_doc_json1, tb_doc_json2 y tb_doc_json4 para guardar los 1, 2 y 4 millones de registros JSON respectivamente; todas con la estructura:

```
CREATE TABLE tb_doc_jsonx (
        persona json);
```

- Se ejecutó la función generar_json() para guardar en cada tabla la cantidad de registros asociados a cada una, de la forma:

```
SELECT * FROM generar_json(1000000, 'tb_doc_json1');
```

- Se creó en PostgreSQL la base de datos db_pruebas_nosql donde se definieron 12 tablas con la siguiente estructura (4 tablas para cargar en cada juego los 1, 2 y 4 millones de registros):

```
CREATE TABLE tb_jsonx (
        persona json);
CREATE UNLOGGED TABLE utb_jsonx (
        persona json);
```

```
CREATE TABLE tb_jsonbx (
        persona jsonb);
CREATE UNLOGGED TABLE utb_jsonbx (
        persona jsonb);
```

- Se hicieron salvas en texto plano de las tablas tb_doc_json1, tb_doc_json2 y tb_doc_json4 de db_generar y se realizaron las siguientes acciones para contar con 12 ficheros .sql:
  - Se crearon y guardaron en un directorio nombrado Ficheros (y ubicado en D en este experimento) 4 copias de los 3 ficheros salvados en texto plano, nombrándolos json1.json, json2.json, json4.json,

json1.sql, json2.sql, json4.sql, ujson1.sql, ujson2.sql, ujson4.sql, jsonb1.sql, jsonb2.sql, jsonb4.sql, ujsonb1.sql, ujsonb2.sql y ujsonb4.sql.

- A los ficheros se les quitaron las líneas generadas por PostgreSQL al inicio y final de cada uno, excepto el comando COPY y, se cambiaron los nombres de las tablas asociándolos a las creadas en db_pruebas_nosql.

- Se definieron las siguientes 5 consultas que devuelven de 30 mil a poco más de 80 mil documentos para la selección de registros que cumplan con:
  - Una condición:

MongoDB	PostgreSQL
db.json4.find({'perfil.estudios': 'Estudios90'}, {identificador: 1, perfil: 1})	**SELECT** persona->'identificador' **AS** id, persona->'perfil' **AS** perfil **FROM** *tb_json4* **WHERE** persona->'perfil'->>'estudios' **LIKE** 'Estudios90';

  - Más de una condición:

MongoDB	PostgreSQL
db.json4.find({$and: [{'perfil.estudios': 'Estudios90'}, {'identificador': {$lte: 3000000}}]}, {identificador: 1, perfil: 1})	**SELECT** persona->'identificador' **AS** id, persona->'perfil' **AS** perfil **FROM** *tb_json4* **WHERE** persona->'perfil'->>'estudios' **LIKE** 'Estudios90' **AND** **cast**(persona->>'identificador' **AS int**) < 3000000;

  - Más de una condición y que estén ordenados por alguna propiedad:

MongoDB	PostgreSQL
db.json4.find({$and: [{'perfil.estudios': 'Estudios90'}, {'identificador': {$lte: 3000000}}]}, {identificador: 1, perfil: 1}) .sort({'identificador': 1})	**SELECT** persona->'identificador' **AS** id, persona->'perfil' **AS** perfil **FROM** *tb_json4* **WHERE** persona->'perfil'->>'estudios' **LIKE** 'Estudios90' **AND** **cast**(persona->>'identificador' **AS int**) < 3000000 **ORDER BY cast**(persona->>'identificador' **AS int**);

  - Al menos una de las condiciones y que estén ordenados por alguna propiedad:

MongoDB	PostgreSQL
db.json4.find({$or: [{'perfil.estudios': 'Estudios24'}, {'perfil.estudios': 'Estudios90'}]},	**SELECT** persona->'identificador' **AS** id, persona->'perfil' **AS** perfil **FROM** *tb_json4* **WHERE** persona->'perfil'->>'estudios' **LIKE** 'Estudios24' **OR** persona->'perfil'->>'estudios' **LIKE** 'Estudios90'

{identificador: 1, perfil: 1}) .sort({'identificador': 1})	**ORDER BY cast**(persona->>'identificador' **AS int**);

- Utilizando una función de agregado:

MongoDB	PostgreSQL
db.json4.group({key: {'perfil.edad': 1}, reduce: function(curr, result) {result.total++;}, initial: {total: 0}})	**SELECT cast**(persona->'perfil'->>'edad' **AS int**) **AS** edad, **count**(*) **FROM** *tb_json4* **GROUP BY cast**(persona->'perfil'->>'edad' **AS int**);

Las pruebas, que se realizaron 3 veces promediándose sus tiempos para obtener un valor aproximado (excepto la determinación del tamaño de las bases de datos), incluyeron la:

1. Carga de 1, 2 y 4 millones de documentos JSON.
2. Determinación del tamaño de las bases de datos una vez cargados los documentos JSON.
3. Ejecución de consultas de selección de registros aleatorios, para las que se utilizaron las tablas y colecciones de 4 millones de registros sin definirse índices y entibiando la caché.

Resultados de la ejecución de las pruebas de rendimiento

Una vez realizadas las pruebas en ambos gestores de bases de datos se obtuvieron los siguientes resultados.

Carga de los registros JSON

La carga de los ficheros JSON se realizó en 12 tablas en PostgreSQL (con y sin la opción UNLOGGED y haciendo uso de los tipos de datos JSON y JSONB). La figura siguiente muestra los tiempos requeridos para ello.

Como muestra la figura 1, los tiempos de respuesta en la carga de registros son inferiores haciendo uso de la opción UNLOGGED en ambos casos, demorándose más en cargar los datos para tablas con tipos JSONB.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. Especial Informática 2016, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 70-81

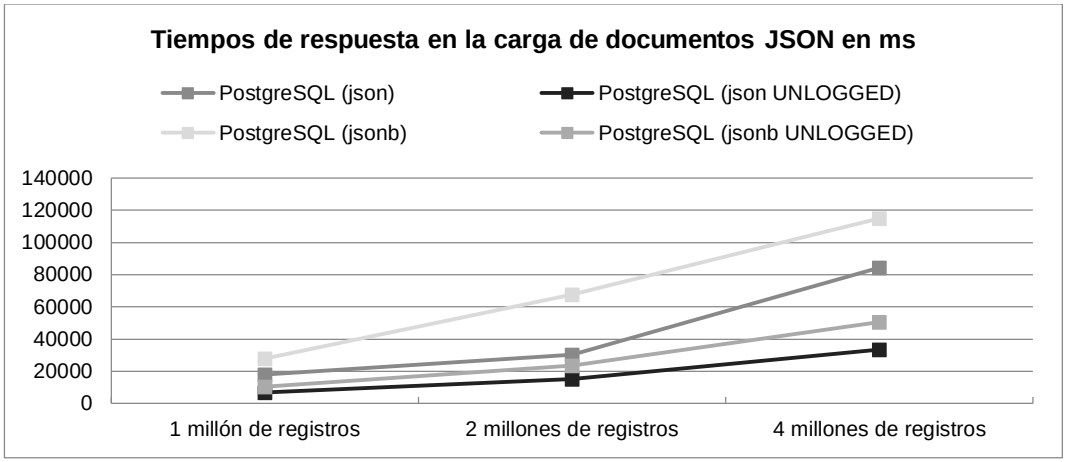

Figura 1. Tiempos de la carga de 1, 2 y 4 millones de registros JSON

Determinación del tamaño de las bases de datos

El tamaño indica el espacio en disco que ocupan las bases de datos. La figura 2 muestra el espacio ocupado una vez cargados los 1, 2 y 4 millones de registros en tablas JSON y JSONB.

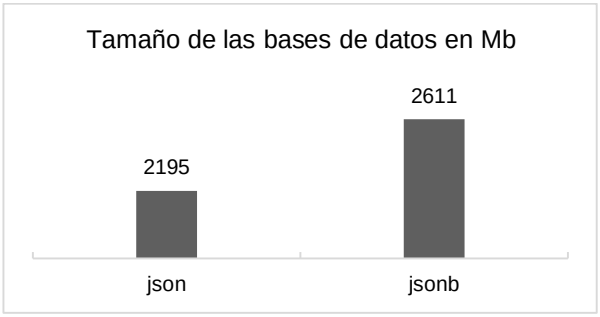

Figura 2: Tamaño de las bases de datos con 7 millones de registros JSON

Como muestra la figura 2, la base de datos que tiene tablas con el tipo de datos JSON ocupa el 84.07% del utilizado por el tipo JSONB.

Revista Cubana de Ciencias Informáticas
Vol. 10, No. Especial Informática 2016, 2016
ISSN: 2227-1899 | RNPS: 2301
http://rcci.uci.cu
Pág. 70-81

Ejecución de consultas de selección de registros aleatorios

Las consultas de selección se ejecutaron en las tablas de 4 millones de registros para contar con la mayor cantidad de datos posibles y, poder evaluar los tiempos de respuesta en la devolución de varios miles de registros JSON. Para ello se utilizó PostgreSQL con y sin la opción UNLOGGED y los tipos de datos JSON y JSONB.

Como muestra la figura 3, los tiempos de respuesta sobre tablas con las mismas opciones para el registro de los datos en los WAL y el mismo tipo de dato son similares, con diferencias no apreciables de menos de 1.5 segundos.

Sin embargo, sí se observa una mejora considerable en los tiempos obtenidos entre JSON y JSONB, comprobándose que este último es, efectivamente, más eficiente al consultar la totalidad de los datos 4.3 veces más rápido que haciendo uso de JSON.

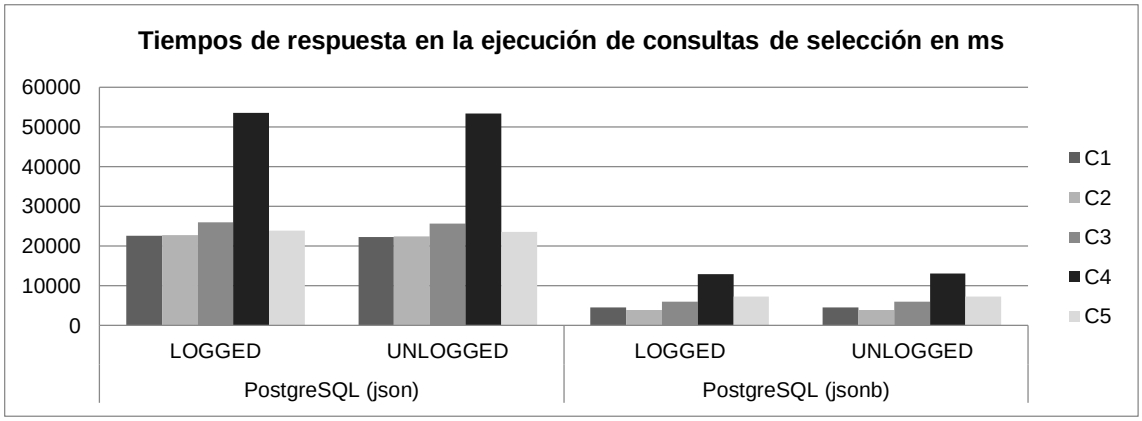

Figura 3. Tiempos de respuesta para consultas de selección de registros aleatorios

Conclusiones

La extensibilidad de PostgreSQL ha permitido la incorporación de características no relacionales a un gestor de bases de datos, inicialmente, objeto-relacional.

De estas nuevas funcionalidades destacan los tipos de datos JSON y el almacenamiento efímero, logrando con la adición del tipo JSONB, en su versión 9.4, una mejora considerable de su rendimiento.

Dichas características fueron evaluadas, mostrándose en los experimentos realizados que la versión 9.4 de PostgreSQL, aun cuando en la carga de los datos y tamaño de la base de datos, el empleo de JSONB queda por debajo de JSON, los tiempos de respuesta para la selección de los registros es 4.3 veces más rápido que usando JSON, lo que evidencia que el gestor ha mejorado considerablemente sus respuestas en una métrica tan utilizada, una vez cargados los datos.

Referencias

CATTELL, RICK. 2010. Scalable SQL and NoSQL Data Stor. New York : ACM SIGMOD Record, 2010. Vol. 39, 4.

DB-engines-trend. 2015. DB-engines. *DB-engines ranking - Trend of MongoDB Popularity.* [En línea] 2015. [Citado el: 06 de abril de 2015.] http://db-engines.com/en/ranking_trend/system/MongoDB.

ENTERPRISEDB. 2014. *Using the NoSQL Capabilities in Postgres.* s.l. : EnterpriseDB Corporation, 2014. White paper.

HAN, JING, y otros. 2011. Survey on NoSQL database. Port Elizabeth : IEEE, 2011. 978-1-4577-0209-9.

LEAVITT, NEAL. 2010. Will NoSQL Databases Live Up to Their Promise? 2010. Vol. Vol. 43, No. 2, págs. 12-14.

LINUX-MAGAZINE. 2009. NoSQL: distributed and Scalable Non-Relational Database Systems. *Linux Magazine.* [En línea] 2009. http://www.linux-mag.com/id/7579/.

MONIRUZZAMAN, A B M Y HOSSAIN, SYED AKHTER. 2013. NoSQL Database: New Era of Databases for Big data Analytics - Classification, Characteristics and Comparison. s.l. : International Journal of Database Theory and Application, 2013. Vol. 6, 4.

NOSQL. 2016. List of NoSQL databases. *NoSQL.* [En línea] 2016. http://nosql-database.org/.

PGDG. 2015. *PostgreSQL 9.4.0 Documentation.* California : s.n., 2015. págs. 151-155, 1479-1494, 2864-2869.

POKORNY, JAROSLAV. 2013. NoSQL databases: a step to database scalability in web environment. s.l. : International Journal of Web Information Systems, 2013. 1744-0084.

SOTOLONGO LEÓN, ANTHONY Y VAZQUEZ ORTÍZ, YUDISNEY. 2013. Evaluación de características NoSQL en PostgreSQL. [En línea] 2013. [Citado el: 02 de mayo de 2015.] http://semanatecnologica.fordes.co.cu/?q=node/856.

Strauch, Christof. 2013. *NoSQl Databases.* 2013.

TAURO, CLARENCE J M, S, ARAVINDH Y A.B, SHREEHARSHA. 2012. Comparative Study of the New Generation, Agile, Scalable, High Performance NOSQL Databases. s.l. : International Journal of Computer Applications, 2012. Vol. 48, 5. 0975 888.

TIWARY, SHASHANK. 2011. *Professional NoSQL.* Indianapolis : John Wiley, 2011. págs. 10-20. 978-0-470-94224-6.

TUMBLR. 2012. Tumblr. *TumblrArchitecture - 15 Billion Page Views A Month And HarderToScaleThanTwitter.* [En línea] 2012. [Citado el: 13 de diciembre de 2012.] http://highscalability.com/blog/2012/2/13/tumblr-architecture-15-billion-pageviews-a-month-and-harder.html.