TP Especial

El Paradigma NOSQL – Bases de Datos de Grafos

*Diego Agustín Orlando – Joaquín Ormachea – Lucía Tay*

**Introducción**

En el siguiente informe estudiaremos el modelado de datos de aeropuertos y rutas aéreas. Además, analizaremos posibles optimizaciones a través de la introducción de índices.

**Modelado de datos**

Implementamos el siguiente grafo:

Route

Contains

Contains

Donde los datos fueron modelados de la siguiente manera:

* Country:
  + Myid: Integer
  + Code: String
  + Desc: String
* Continent:
  + Myid: Integer
  + Code: String
  + Desc: String
* Version:
  + Myid: Integer
  + Code: String
  + Desc: String
* Airport:
  + Myid: Integer
  + Code: String
  + Icao: String
  + City: String
  + Desc: String
  + Runways: Integer
  + Longest: String
  + Elev: Integer
  + Lat: String
  + Lon: String
* Contains (Multiplicidad SIMPLE):
  + Myid: Integer
* Route (Multiplicidad MULTI):
  + Myid: Integer
  + Dist: Integer

Además, fue necesario utilizar un índice de tipo *composite* para la propiedad *myid,* ya que cada vez que se quería insertar una arista era necesario recorrer todo el grafo para encontrar los dos nodos que conectaba y dicha operación consumía demasiado tiempo.

El código que contiene tanto la creación del *schema* como la inserción de los nodos y aristas se encuentra en el archivo llamado *CSVLoad.groovy*.

**Índices**

A continuación, detallaremos distintas consultas y los tipos de índices creados para tratar de optimizarlas. También analizaremos el tiempo de ejecución para cada una de ellas con y sin índice.

* Query 1: *graph.traversal().V().has('desc', 'Argentina').values('desc')*
  + Tipo de índice: *composite simple* en la propiedad *desc*
* Query 2: *graph.traversal().V().has('runways', 5).values('desc')*
  + Tipo de índice: *composite simple* en la propiedad *runways*
* Query 3: *graph.traversal().V().hasLabel('country').has('desc', 'Argentina').values('desc')*
  + Tipo de índice: *composite* restringido en la propiedad *desc* para los nodos *country*
* Query 4: *graph.traversal().V().has('lat', '-348.222').has('lon', '-585.358').values('code', 'desc')*
  + Tipo de índice: *composite* doble en las propiedades *lat* y *lon*
* Query 5:  *graph.traversal().V().has('desc', textContainsPrefix('Ar')).values('desc')*
  + Tipo de índice: *mixed full text* en la propiedad *text*
* Query 6: *graph.traversal().V().has('runways', gt(5)).values('desc','runways')*
  + Tipo de índice: *mixed range* en la propiedad *runways*
* Query 7: *graph.traversal().V().has('city', textFuzzy('Buenoz Airez')).values('desc','city')*
  + Tipo de índice: *mixed string* en la propiedad *city*
* Query 8: *graph.traversal().V().hasLabel('airport').inE().otherV().hasLabel('continent').groupCount().by('desc').unfold()*
  + Tipo de índice: *composite* en la propiedad *desc*
* Query 9: *graph.traversal().V().hasLabel('airport').has('city', textContainsPrefix('C')).has('code', textContainsPrefix('C')).has('desc', textContainsPrefix('C')).values('desc','city','code')*
  + Tipo de índice: *mixed full text* en las propiedades *city, code* y *desc*
* Query 10:  *graph.traversal().V().hasLabel('airport').has('city', textContainsRegex('.\*[z,Z].\*')).has('elev', inside(1000,2000)).values('desc','city','elev')*
  + Tipo de índice: *mixed range* en las propiedades *city* y *elev*
* Query 11: *graph.traversal().E().has('dist', lte(400)).order().by('dist', desc).values('dist')*
  + Tipo de índice: *mixed Edge* en la propiedad *dist*
* Query 12: *graph.traversal().V().hasLabel('airport').has('desc', containing('International')).order().by(bothE().hasLabel('route').count()).values('desc')*
  + Tipo de índice: *composite* en la propiedad *desc*
* Query 13: *graph.traversal().V().order().by('runways').limit(10).values('desc','runways')*
  + Para esta *query* no logramos optimizar de ninguna manera con índices, intentamos usar *mixed, composite*, y restringirlo para los aeropuertos, pero siempre necesita iterar por todos los nodos incluso sabiendo que tiene que mostrar únicamente los 10 primeros.
* Query 14: *graph.traversal().V().has('elev', gte(100)).has('runways', inside(5,7)).has('longest', lt(14000)).values('code','elev','runways','longest')*
  + A pesar de no tener todos los índices que creeríamos necesarios para que la *query* funcione de forma óptima, creemos que por el orden en que se hacen los filtros la solución no itera por muchos nodos y eso hace que el *warning* de utilizar índices no aparezca. Además, *longest* en nuestro grafo es un *String* y se está tratando como un *integer* sin hacer ningún tipo de conversión manual.

Como se puede observar, en casi todos los casos la creación de índices trajo mejoras significativas. Creemos que aquellas *queries* que no presentan mejoras se deben a optimizaciones internas del motor. Además, es interesante notar que en la *query* 4 casi no se observan diferencias entre utilizar dos índices individuales en vez de un solo índice doble.

Los códigos de creación de las *queries* y de los índices se encuentra en el archivo *indexAndQueries.txt*