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: Integer
  + Elev: Integer
  + Lat: Double
  + Lon: Double
* 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.

* Índice 1: *descComposite*
  + Código:
    - mgmt = graph.openManagement()
    - mgmt.buildIndex('descComposite', Vertex.class).addKey(mgmt.getPropertyKey('desc')).buildCompositeIndex()
    - mgmt.commit()
    - ManagementSystem.awaitGraphIndexStatus(graph, 'descComposite').call()
    - mgmt = graph.openManagement()
    - mgmt.updateIndex(mgmt.getGraphIndex("descComposite"), SchemaAction.REINDEX).get()
    - mgmt.commit()
* Índice 2: *runwaysComposite*
  + *Código:*
    - mgmt = graph.openManagement()
    - mgmt.buildIndex('runwaysComposite', Vertex.class).addKey(mgmt.getPropertyKey('runways')).buildCompositeIndex()
    - mgmt.commit()
    - ManagementSystem.awaitGraphIndexStatus(graph, 'runwaysComposite').call()
    - mgmt = graph.openManagement()
    - mgmt.updateIndex(mgmt.getGraphIndex("runwaysComposite"), SchemaAction.REINDEX).get()
    - mgmt.commit()
* Índice 3: *countryDescComposite*
  + Código:
    - mgmt = graph.openManagement()
    - mgmt.buildIndex('countryDescComposite', Vertex.class).addKey(mgmt.getPropertyKey('desc')).indexOnly(mgmt.getVertexLabel('country')).buildCompositeIndex()
    - mgmt.commit()
    - ManagementSystem.awaitGraphIndexStatus(graph, 'countryDescComposite').call()
    - mgmt = graph.openManagement()
    - mgmt.updateIndex(mgmt.getGraphIndex('countryDescComposite'), SchemaAction.REINDEX).get()
    - mgmt.commit()
* Índice 4: *latlonComposite*
  + Código:
    - mgmt = graph.openManagement()
    - mgmt.buildIndex('latlonComposite', Vertex.class).addKey(mgmt.getPropertyKey('lat')).addKey(mgmt.getPropertyKey('lon')).buildCompositeIndex()
    - mgmt.commit()
    - ManagementSystem.awaitGraphIndexStatus(graph, 'latlonComposite').call()
    - mgmt = graph.openManagement()
    - mgmt.updateIndex(mgmt.getGraphIndex("latlonComposite"), SchemaAction.REINDEX).get()
    - mgmt.commit()
* Índice 5: *descCompositeCons*
  + Código:
    - mgmt = graph.openManagement()
    - mgmt.buildIndex('descCompositeCons', Vertex.class).addKey(mgmt.getPropertyKey('desc')).indexOnly(mgmt.getVertexLabel('continent')).buildCompositeIndex()
    - mgmt.commit()
    - ManagementSystem.awaitGraphIndexStatus(graph, 'descCompositeCons').call()
    - mgmt = graph.openManagement()
    - mgmt.updateIndex(mgmt.getGraphIndex("descCompositeCons"), SchemaAction.REINDEX).get()
    - mgmt.commit()
* Índice 6: *descMixedCons*
  + Código:
    - mgmt = graph.openManagement()
    - mgmt.buildIndex('descMixedCons', Vertex.class).addKey(mgmt.getPropertyKey('desc')).indexOnly(mgmt.getVertexLabel('airport')).buildMixedIndex("search")
    - mgmt.commit()
    - ManagementSystem.awaitGraphIndexStatus(graph, 'descMixedCons').call()
    - mgmt = graph.openManagement()
    - mgmt.updateIndex(mgmt.getGraphIndex("descMixedCons"), SchemaAction.REINDEX).get()
    - mgmt.commit()
* Índice 7: *descMixed2*
  + Código:
    - mgmt = graph.openManagement()
    - mgmt.buildIndex('descMixed2', Vertex.class).addKey(mgmt.getPropertyKey('desc')).buildMixedIndex("search")
    - mgmt.commit()
    - ManagementSystem.awaitGraphIndexStatus(graph, 'descMixed2').call()
    - mgmt = graph.openManagement()
    - mgmt.updateIndex(mgmt.getGraphIndex("descMixed2"), SchemaAction.REINDEX).get()
    - mgmt.commit()
* Índice 8: *tripleMixCons2*
  + Código:
    - mgmt = graph.openManagement()
    - mgmt.buildIndex('tripleMixCons2', Vertex.class).addKey(mgmt.getPropertyKey('city')).addKey(mgmt.getPropertyKey('code')).addKey(mgmt.getPropertyKey('desc')).indexOnly(mgmt.getVertexLabel('airport')).buildMixedIndex("search")
    - mgmt.commit()
    - ManagementSystem.awaitGraphIndexStatus(graph, 'tripleMixCons2').call()
    - mgmt = graph.openManagement()
    - mgmt.updateIndex(mgmt.getGraphIndex("tripleMixCons2"), SchemaAction.REINDEX).get()
    - mgmt.commit()
* Índice 9: *runwaysMixed*
  + Código:
    - mgmt = graph.openManagement()
    - mgmt.buildIndex('runwaysMixed', Vertex.class).addKey(mgmt.getPropertyKey('runways')).buildMixedIndex("search")
    - mgmt.commit()
    - ManagementSystem.awaitGraphIndexStatus(graph, 'runwaysMixed').call()
    - mgmt = graph.openManagement()
    - mgmt.updateIndex(mgmt.getGraphIndex("runwaysMixed"), SchemaAction.REINDEX).get()
    - mgmt.commit()
* Índice 10: *cityElevConstrain*
  + Código:
    - mgmt = graph.openManagement()
    - mgmt.buildIndex('cityElevConstrain', Vertex.class).addKey(mgmt.getPropertyKey('city')).addKey(mgmt.getPropertyKey('elev')).indexOnly(mgmt.getVertexLabel('airport')).buildMixedIndex("search")
    - mgmt.commit()
    - ManagementSystem.awaitGraphIndexStatus(graph, 'cityElevConstrain').call()
    - mgmt = graph.openManagement()
    - mgmt.updateIndex(mgmt.getGraphIndex("cityElevConstrain"), SchemaAction.REINDEX).get()
    - mgmt.commit()
* Índice 11: *cityString*
  + Código:
    - mgmt = graph.openManagement()
    - mgmt.buildIndex('cityString', Vertex.class).addKey(mgmt.getPropertyKey('city'), Mapping.STRING.asParameter()).buildMixedIndex("search")
    - mgmt.commit()
    - ManagementSystem.awaitGraphIndexStatus(graph, 'cityString').call()
    - mgmt = graph.openManagement()
    - mgmt.updateIndex(mgmt.getGraphIndex("cityString"), SchemaAction.REINDEX).get()
    - mgmt.commit()
* Índice 12: *runwaysMixedConst*
  + Código:
    - mgmt = graph.openManagement()
    - mgmt.buildIndex('runwaysMixedConst', Vertex.class).addKey(mgmt.getPropertyKey('runways')).indexOnly(mgmt.getVertexLabel('airport')).buildMixedIndex("search")
    - mgmt.commit()
    - ManagementSystem.awaitGraphIndexStatus(graph, 'runwaysMixedConst2').call()
    - mgmt = graph.openManagement()
    - mgmt.updateIndex(mgmt.getGraphIndex("runwaysMixedConst2"), SchemaAction.REINDEX).get()
    - mgmt.commit()
* Índice 13: *distEdge2*
  + Código:
    - mgmt = graph.openManagement()
    - mgmt.buildIndex('distEdge2', Edge.class).addKey(mgmt.getPropertyKey('dist')).buildMixedIndex("search")
    - mgmt.commit()
    - ManagementSystem.awaitGraphIndexStatus(graph, 'distEdge2').call()
    - mgmt = graph.openManagement()
    - mgmt.updateIndex(mgmt.getGraphIndex("distEdge2"), SchemaAction.REINDEX).get()
    - mgmt.commit()
* Índice 14: *cityComposite*
  + Código:
    - mgmt = graph.openManagement()
    - mgmt.buildIndex('cityComposite', Vertex.class).addKey(mgmt.getPropertyKey('city')).buildCompositeIndex()
    - mgmt.commit()
    - ManagementSystem.awaitGraphIndexStatus(graph, 'cityComposite').call()
    - mgmt = graph.openManagement()
    - mgmt.updateIndex(mgmt.getGraphIndex("cityComposite"), SchemaAction.REINDEX).get()
    - mgmt.commit()
* Índice 15: *coordsIndex*
  + Código:
    - mgmt = graph.openManagement()
    - coords = mgmt.makePropertyKey('coords').dataType(Geoshape.class).make()
    - mgmt.buildIndex('coordsIndex', Vertex.class).addKey(coords, Mapping.PREFIX\_TREE.asParameter()).buildMixedIndex("search")
    - mgmt.commit()
    - graph.traversal().V().local( \_\_.as('x').map{ Geoshape.point(it.get().values('lat').next(), it.get().values('lon').next()) }.as('coords').select('x').property('coords', select('coords')) ).iterate()
* Query 1.1: *graph.traversal().V().has('desc', 'Argentina').values('desc')*
  + Índice utilizado:  *descComposite*
* Query 1.2: *graph.traversal().V().has('desc', 'Zambia').values('desc')*
  + Índice utilizado: *descComposite*
* Query 1.3:  *graph.traversal().V().has('desc', 'Xer').values('desc')*
  + Índice utilizado: *descComposite*
* Query 2.1: *graph.traversal().V().has('runways', 5).values('desc')*
  + Índice utilizado: *runwaysComposite*
* Query 2.2: *graph.traversal().V().has('runways', 1).values('desc')*
  + Índice utilizado: *runwaysComposite*
* Query 2.3: *graph.traversal().V().has('runways', 10).values('desc')*
  + Índice utilizado: *runwaysComposite*
* Query 3.1: *graph.traversal().V().hasLabel('country').has('desc', 'Argentina').values('desc')*
  + Índice utilizado: *descComposite* (suponíamos que iba a utilizar *countryDescComposite*)
* Query 3.2: *graph.traversal().V().has('desc', 'Argentina').hasLabel('country').values('desc')*
  + Índice utilizado: *descComposite* (suponíamos que iba a utilizar *countryDescComposite*)
* Query 3.3: *graph.traversal().V().hasLabel('country').has('desc', 'zzz').values('desc')*
  + Índice utilizado: *descComposite* (suponíamos que iba a utilizar *countryDescComposite*)
* Query 3.4: *graph.traversal().V().has('desc', 'zzz').hasLabel('country').values('desc')*
  + Índice utilizado: *descComposite* (suponíamos que iba a utilizar *countryDescComposite*)
* Query 4.1: *graph.traversal().V().has('lat', '-34.8222').has('lon', '-58.5358').values('code', 'desc')*
  + Índice utilizado: *latlonComposite*
* Query 4.2: *graph.traversal().V().has('lat', '0').has('lon', '0').values('code', 'desc')*
  + Índice utilizado: *latlonComposite*
* Query 5.1:  *graph.traversal().V().hasLabel('airport').inE().otherV().hasLabel('continent').groupCount().by('desc').unfold()*
  + Índice utilizado: incierto (no se ve el comienzo del *profile*)
* Query 5.2: *graph.traversal().V().hasLabel('airport').inE().otherV().hasLabel('country').groupCount().by('desc').unfold()*
  + Índice utilizado: incierto (no se ve el comienzo del *profile*)
* Query 6.1: *graph.traversal().V().hasLabel('airport').has('desc', containing('International')).order().by(bothE().hasLabel('route').count()).values('desc')*
  + Índice utilizado: ninguno
* Query 6.2: *graph.traversal().V().hasLabel('airport').has('desc', containing('a')).order().by(bothE().hasLabel('route').count()).values('desc')*
  + Índice utilizado: ninguno
* Query 6.3: *graph.traversal().V().hasLabel('airport').has('desc', containing('zzz')).order().by(bothE().hasLabel('route').count()).values('desc')*
  + Índice utilizado: ninguno
* Query 7.1: *graph.traversal().V().has('desc', textContainsPrefix('Ar')).values('desc')*
  + Índice utilizado: *descMixed2*
* Query 7.2: *graph.traversal().V().has('desc', textContainsPrefix(‘Bel’)).values('desc')*
  + Índice utilizado: *descMixed2*
* Query 8.1:  *graph.traversal().V().hasLabel('airport').has('city', textContainsPrefix('C')).has('code', textContainsPrefix('C')).has('desc', textContainsPrefix('C')).values('desc','city','code')*
  + Índice utilizado: *tripleMixCons2*
* Query 8.2: *graph.traversal().V().hasLabel('airport').has('city', textContainsPrefix('R')).has('code', textContainsPrefix('G')).has('desc', textContainsPrefix('B')).values('desc','city','code')*
  + Índice utilizado: *tripleMixCons2*
* Query 8.3:  *graph.traversal().V().hasLabel('airport').has('city', textContainsPrefix('R')).has('code', textContainsPrefix('B')).has('desc', textContainsPrefix('B')).values('desc','city','code')*
  + Índice utilizado: *tripleMixCons2*
* Query 9.1: *graph.traversal().V().has('runways', gt(5)).values('desc','runways')*
  + Índice utilizado: *runwaysMixed*
* Query 9.2: *graph.traversal().V().has('runways', gt(1)).values('desc','runways')*
  + Índice utilizado: *runwaysMixed*
* Query 9.3: *graph.traversal().V().has('runways', gt(10)).values('desc','runways')*
  + Índice utilizado: *runwaysMixed*
* Query 10.1:  *graph.traversal().V().hasLabel('airport').has('city', textContainsRegex('.\*[z,Z].\*')).has('elev', inside(1000,2000)).values('desc','city','elev')*
  + Índice utilizado: *cityElevConstrain*
* Query 10.2:  *graph.traversal().V().has('city', textContainsRegex('.\*[z,Z].\*')).has('elev', inside(1000,2000)).hasLabel('airport').values('desc','city','elev')*
  + Índice utilizado: *cityElevConstrain*
* Query 10.3:  *graph.traversal().V().hasLabel('airport').has('city', textContainsRegex('.\*[!].\*')).has('elev', inside(1000,2000)).values('desc','city','elev')*
  + Índice utilizado: *cityElevConstrain*
* Query 10.4:  *graph.traversal().V().has('city', textContainsRegex('.\*[!].\*')).has('elev', inside(1000,2000)).hasLabel('airport').values('desc','city','elev')*
  + Índice utilizado: *cityElevConstrain*
* Query 10.5:  *graph.traversal().V().hasLabel('airport').has('city', textContainsRegex('.\*[g,G].\*')).has('elev', inside(1000,2000)).values('desc','city','elev')*
  + Índice utilizado: *cityElevConstrain*
* Query 11.1: *graph.traversal().V().has('city', textFuzzy('Buenoz Airez')).values('desc','city')*
  + Índice utilizado: *cityString*
* Query 11.2: *graph.traversal().V().has('city', textFuzzy('landon')).values('desc','city')*
  + Índice utilizado: *cityString*
* Query 11.3: *graph.traversal().V().has('city', textFuzzy('zzz')).values('desc','city')*
  + Índice utilizado: *cityString*
* Query 12.1: *graph.traversal().V().has('elev', gte(100)).has('runways', inside(5,7)).has('longest', lt(14000)).values('code','elev','runways','longest')*
  + Índice utilizado: *runwaysMixed*
* Query 12.2: *graph.traversal().V().has('longest', lt(14000)).has('elev', gte(100)).has('runways', inside(5,7)).values('code','elev','runways','longest')*
  + Índice utilizado: *runwaysMixed*
* Query 12.3: *graph.traversal().V().has('longest', lt(14000)).has('elev', gte(100)).has('runways', inside(1,7)).values('code','elev','runways','longest')*
  + Índice utilizado: *runwaysMixed*
* Query 13: *graph.traversal().V().order().by('runways').limit(10).values('desc','runways')*
  + Índice utilizado: ninguno
* Query 14.1: *graph.traversal().E().has('dist', lte(400)).order().by('dist', desc).values('dist')*
  + Índice utilizado: *distEdge2*
* Query 14.2: *graph.traversal().E().has('dist', lte(0)).order().by('dist', desc).values('dist')*
  + Índice utilizado: *distEdge2*
* Query 15: *graph.traversal().V().not(bothE()).values('desc')*
  + Índice utilizado: ninguno
* Query 16.1: *graph.traversal().V().has('city', 'New York').out().has('city', within('London', 'Paris', 'Amsterdam', 'Berlin')).path().by('desc')*
  + Índice utilizado: *cityString* (suponíamos que iba a utilizar *cityComposite*)
* Query 16.2: *graph.traversal().V().has('city', 'zzz').out().has('city', within('London', 'Paris', 'Amsterdam', 'Berlin')).path().by('desc')*
  + Índice utilizado: *cityString* (suponíamos que iba a utilizar *cityComposite*)
* Query 16.3: *graph.traversal().V().has('city', 'Buenos Aires').out().has('city', within('London', 'Paris', 'Amsterdam', 'Berlin')).path().by('desc')*
  + Índice utilizado: *cityString* (suponíamos que iba a utilizar *cityComposite*)
* Query 17.1: *graph.traversal().V().has("coords", geoWithin(Geoshape.circle(-34.8222, -58.5358, 1))).valueMap(true)*
  + Índice utilizado: *coordsIndex*
* Query 17.2: *graph.traversal().V().has("coords", geoWithin(Geoshape.circle(-34.8222, -58.5358, 100))).valueMap(true)*
  + Índice utilizado: *coordsIndex*
* Query 17.3: *graph.traversal().V().has("coords", geoWithin(Geoshape.circle(-34.8222, -58.5358, 100000))).valueMap(true)*
  + Índice utilizado: *coordsIndex*
* Query 17.4: *graph.traversal().V().has("coords", geoDisjoint(Geoshape.circle(-34.8222, -58.5358, 1))).valueMap(true)*
  + Índice utilizado: *coordsIndex*
* Query 17.5: *graph.traversal().V().has("coords", geoDisjoint(Geoshape.circle(-34.8222, -58.5358, 100))).valueMap(true)*
  + Índice utilizado: *coordsIndex*
* Query 17.6: *graph.traversal().V().has("coords", geoDisjoint(Geoshape.circle(-34.8222, -58.5358, 100000))).valueMap(true)*
  + Índice utilizado: *coordsIndex*
* Query 17.7: *graph.traversal().V().has("coords", geoIntersect(Geoshape.circle(-34.8222, -58.5358, 1))).valueMap(true)*
  + Índice utilizado: *coordsIndex*
* Query 17.8: *graph.traversal().V().has("coords", geoIntersect(Geoshape.circle(-34.8222, -58.5358, 100))).valueMap(true)*
  + Índice utilizado: *coordsIndex*
* Query 17.9: *graph.traversal().V().has("coords", geoIntersect(Geoshape.circle(-34.8222, -58.5358, 100000))).valueMap(true)*
  + Índice utilizado: *coordsIndex*
* Query 17.10: *graph.traversal().V().has("coords", geoContains(Geoshape.circle(-34.8222, -58.5358, 1))).valueMap(true)*
  + Índice utilizado: *coordsIndex*
* Query 17.11: *graph.traversal().V().has("coords", geoContains(Geoshape.circle(-34.8222, -58.5358, 100))).valueMap(true)*
  + Índice utilizado: *coordsIndex*
* Query 17.12: *graph.traversal().V().has("coords", geoContains(Geoshape.circle(-34.8222, -58.5358, 100000))).valueMap(true)*
  + Índice utilizado: *coordsIndex*
* Query 18.1: *graph.traversal().V().or(has('city', textFuzzy('Landon')), has('desc', textContainsPrefix('London'))).values('city','desc')*
  + Índice utilizado: *descMixed2*
* Query 18.2: *graph.traversal().V().or(has('desc', textContainsPrefix('London')), has('city', textFuzzy('Landon'))).values('city','desc')*
  + Índice utilizado: *cityString*

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 encuentran en el archivo *indexAndQueries.txt*