GDAL ja ElasticSearch

Jukka Rahkonen, http://latuviitta.org

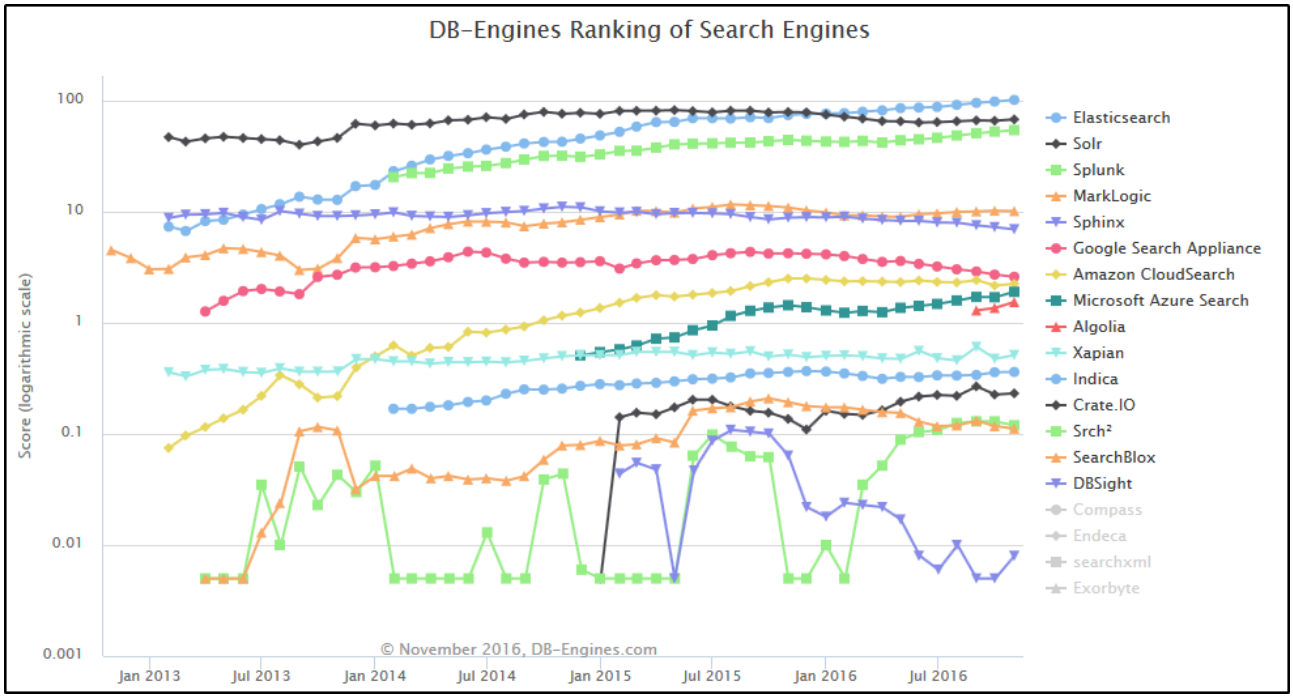
Viimeksi muokattu 23. marraskuuta 2016

ElasticSearch-ajuri esiteltiin GDAL-versiossa 1.10. Ajuri pystyi pelkästään kirjoittamaan dataa ElasticSearch-tietokantaan mutta ei lukemaan tietoja kannasta ulos.
GDAL-versiossa 2.1, joka julkaistiin 2. toukokuuta 2016, tuli merkittäviä parannuksia ElasticSearch-ajuriin http://www.gdal.org/drv_elasticsearch.html.

- GDAL voi myös lukea ElasticSearch-tietokantaa
- Tukee myös GEO_SHAPE-indeksointia: pisteiden lisäksi viivat ja alueet
- ogrinfo:ssa ja ogr2ogr:ssa uusi @file -syntaksi, jonka ansiosta json-muotoiset filter:it ja query:t voidaan lukea tiedostosta
- Rahoittaja: SAFE Software (FME)

Ajuri tukee ElasticSearch-versioita 1.x ja 2.x, mutta loppuvuodesta 2016 ei vielä ElasticSearch-versiota 5.x.

Vuonna 2010 syntynyt ElasticSearch käyttää taustalla Apache Lucenea, kuten jo vuonna 2004 esitelty Solr. ElasticSearch eroaa Solr:sta eniten siinä, kuinka hajautettua hakukoneklusteria hallitaan. Solr ja ElasticSearch ovat ilmeisesti tämän hetken suosituimmat hakukonemoottorit http://db-engines.com/en/ranking/search+engine



ElasticSearch:in asentaminen testaamista varten

1) Lataa ohjelman asennuspaketti zip-tiedostona, esimerkiksi
   https://www.elastic.co/downloads/past-releases/elasticsearch-2-4-1
2) Pura arkisto kovalevylle ja aja elasticsearch.bat tai elasticsearch.sh
3) ElasticSearch on käynnissä osoitteessa http://localhost:9200

Paikkatietojen vienti ElasticSearch-tietokantaan GDAL:lla

Esimerkkiaineistona käytetään GeoNames-paikannimiaineistoa http://download.geonames.org/export/dump/allCountries.zip. Lähtöaineiston tiedot:

```
ogrinfo  -al -so -noextent CSV:allCountries.txt
INFO: Open of `allCountries.txt'
      using driver `CSV' successful.

Layer name: allCountries
Geometry: 3D Point
Feature Count: 10511705
Layer SRS WKT:
(unknown)
GEONAMEID: String (0.0)
NAME: String (0.0)
ASCIINAME: String (0.0)
ALTNAMES: String (0.0)
LATITUDE: Real (0.0)
LONGITUDE: Real (0.0)
FEATCLASS: String (0.0)
FEATCODE: String (0.0)
COUNTRY: String (0.0)
CC2: String (0.0)
ADMIN1: String (0.0)
ADMIN2: String (0.0)
ADMIN3: String (0.0)
ADMIN4: String (0.0)
POPULATION: Real (0.0)
ELEVATION: Integer (0.0)
GTOPO30: Integer (0.0)
TIMEZONE: String (0.0)
MODDATE: String (0.0)
```

Ogr2ogr-komento, joka vie GeoNames-aineiston ElasticSearch:iin. Komennossa on kaksi parametria, joiden käyttö ei ole välttämätöntä, mutta tarpeen tässä ohjeessa myöhemmin esitettävien kyselyiden tekemiseksi

- `-lco GEOM_MAPPING_TYPE=GEO_SHAPE`
  - Geometria voidaan tallentaa joko GEO_POINT- tai GEO_SHAPE-muodossa ja spatiaalikyselyt tehdään niille hieman eri tavoin. Geo_shape on ainoa vaihtoehto muille kuin pisteille, joten jos tallentaa pisteetkin geo_shape-muotoon, niin selviytyy yksien kyselytyyppien opettelulla.
- `-lco NOT_ANALYZED_FIELDS=ASCIINAME`
  - Jokerimerkkien * ja ? käyttö wildcard-kyselyssä toimiin vain kentille, joita ei ole analysoitu.

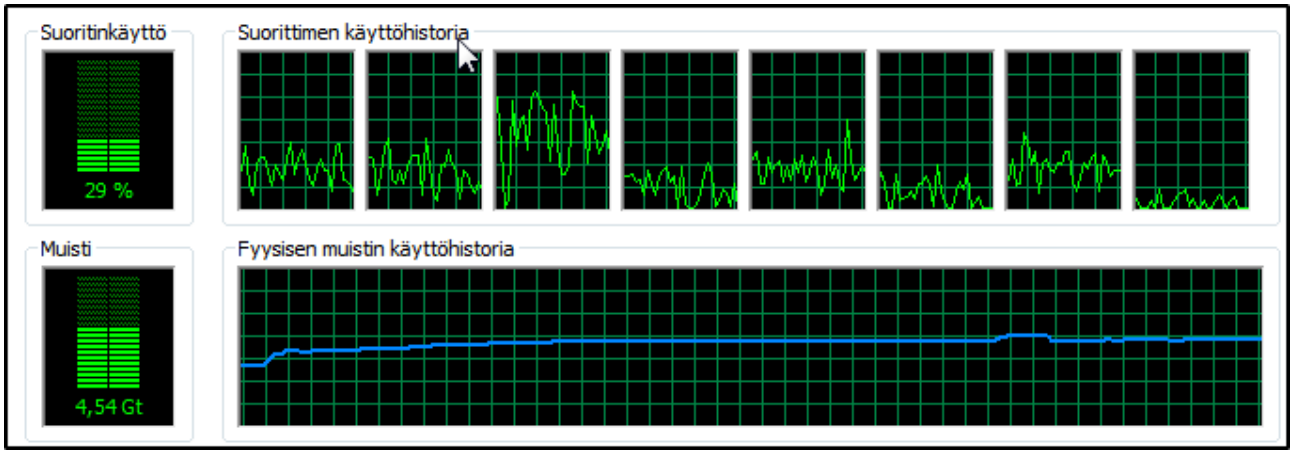
Kolmas parametri muuttaa tiedonsiirroon tapahtumaan 5 MB suuruisina erinä, mikä nopeuttaa operaatiota, jos tietokantaa käyttävässä koneessa riittää tehoja.

- `-lco BULK_SIZE=5000000`

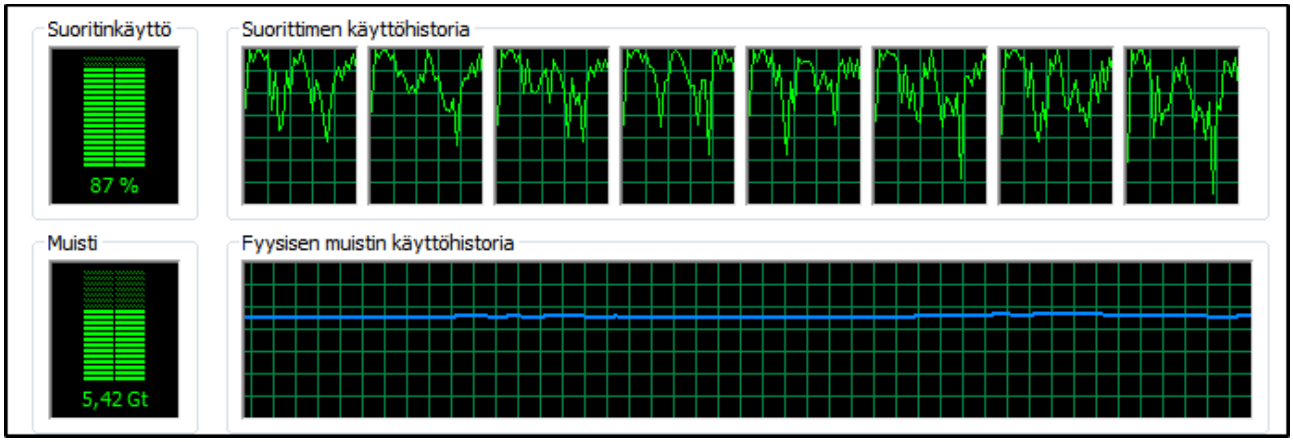
```
ogr2ogr -f "ElasticSearch" http://localhost:9200 CSV:allCountries.txt -lco
GEOM_MAPPING_TYPE=GEO_SHAPE -lco NOT_ANALYZED_FIELDS=ASCIINAME -lco
BULK_SIZE=5000000
Warning 1: No SRS given for geometry column geometry. SRS is assumed to be EPSG:
4326 (WGS84 long/lat)
```

Huom. Jos GDAL-versio on < 2.1, niin yllä oleva komento saattaa toimia väärin.
https://trac.osgeo.org/gdal/ticket/6086

Kaunko kestää ja miten raskasta on



Prosessin nimi	Suoritin	Muisti (yksityinen työskentelysarja)	Kuvaus
java.exe *32	01	1 142 088 kt	Java(TM) Platform SE binary
ogr2ogr.exe	12	149 912 kt	ogr2ogr.exe



Kuorma jakaantuu ogr2ogr-ohjelman ja ElasticSearch-tietokantaohjelman välille, joka näkyy resurssienhallinnassa java-prosessina. Kuormitus ohjelmien välillä vaihtelee ja välillä ogr2ogr

työskentelee kovemmin, välillä taas ElasticSearch. Muunnosohjelma ei näyttäisi pystyvän hyödyntämään kovin hyvin kaikkia käytettävissä olevia prosessoreita. Muistin tarve on varsin pieni.

ElasticSearch:in oletusasetuksilla Javan muistivarauksen ylärajaksi on asetettu 1 Gt mikä on käytännössä myös 32-bittisen Javan maksimi, joten jos muistivarausta halutaan kasvattaa niin on otettava myös 64-bittinen Java käyttöön. ElasticSearch-dokumenttien perusteella muistiasetuksilla pelaaminen näyttää vaativan hyvää ymmärrystä siitä, miten Java toimii, joten tavallisen käyttäjän on parasta olla koskematta niihin.

Muunnokseen meni tässä tapauksessa **169 minuuttia** (Windows-kannettava, Core i7, data USB2-levyllä), **61833 paikannimeä minuutissa**. ElasticSearch-tietokanta on koko ajan käytettävissä myös hakujen tekemistä varten. Vertailutestissä pöytäkoneella vauhti oli **142000 vietyä paikannimeä minuutissa**. Alueiden vieminen tietokantaan on huomattavasti hitaampaa.

Päivitys 22.marraskuuta 2016

Toistin testin uusilla ohjelmaversioilla:

- GDAL 2.2.0-dev,
- ElasticSearch 2.4.1
- Java 1.8.0_102 64-Bit Server VM
- Pöytäkone sama kuin edellä, mutta Geonames-tekstitiedosto koneen omalla kovalevyllä ja ElasticSearch-tietokanta ulkoisella USB3-kovalevyllä.
- BULK_SIZE 5000000 käytössä

Numerot näyttävät melko erilaisilta kuin aikaisemmassa testissä ElasticSearch-versiolla 1.7.3 ja 32-bittisellä Javalla, joka ei ollut Server VM -versio.

- Javan muistinkulutus on pienempi, noin 440 kilotavua
- Javan prosessorikuorma korkeampi, yli 10 %, ja aina suurempi kuin ogr2ogr:llä
- ogr2ogr:n muistin käyttö vain 16 kilotavua, prosessorikuorma 1-3 %.
- muunnosvauhti **301700** paikannimeä minuutissa
- kokonaisaika 11163481 paikannimelle **37 minuuttia**
- BULK_SIZE:n käyttö säästi aikaa peräti 23 minuuttia

ElasticSearch:in ja sen indeksien toiminta

ElasticSearch indeksoi dokumentteja, jotka ovat json-muodossa. GIS-maailmassa tällaista dokumenttia vastaa kohde, yksi geometria ja kaikki sen ominaisuustiedot. Oletusarvoilla ElasticSearch etsii dokumentista kaikki kentät ja tulkitsee parhaan kykynsä mukaan automaattisesti minkä tyyppisiä kentät ovat: tekstiä, kokonaislukuja, päivämääriä ym. Tämä toimii yleensä hyvin, mutta kentät voidaan myös mapata käsin asetustiedoston avulla.

Oletuksena on, että jokaiselle kentälle luodaan indeksi, ja sen lisäksi koko indeksoitava dokumentti tallennetaan tietokantaan.

Seuraava ogrinfo-kysely yhteenvetotietojen lukemiseksi on nopea, jos geometriat on tallennettu geo_point-muodossa ja todella hidas, jos geometriat on tallennettu geo_shape-muodossa. Ero syntyy siitä, että ensimmäisessä tapauksessa tietojen maantieteellinen kattavuus saadaan selville suoraan tietokannasta, mutta **geo_shape-tapauksessa GDAL joutuu selvittämään aineiston maksimi- ja minimikoordinaatit geometrioiden perusteella ja lukemaan tietokannasta joka ikisen kohteen**, tässä tapauksessa siis yli 10 miljoonaa riviä. Käytä siis tätä kyselyä harkiten!

```
ogrinfo ES: allcountries -so
INFO: Open of `ES:'
      using driver `ElasticSearch' successful.

Layer name: allcountries
Geometry: Point
Feature Count: 10511705
Extent: (-179.983600, -90.000000) - (180.000000, 90.000000)
Layer SRS WKT:
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0,
        AUTHORITY["EPSG","8901"]],
    UNIT["degree",0.0174532925199433,
        AUTHORITY["EPSG","9122"]],
    AUTHORITY["EPSG","4326"]]
FID Column = ogc_fid
Geometry Column = geometry
_id: String (0.0)
ADMIN1: String (0.0)
ADMIN2: String (0.0)
ADMIN3: String (0.0)
ADMIN4: String (0.0)
ALTNAMES: String (0.0)
ASCIINAME: String (0.0)
CC2: String (0.0)
COUNTRY: String (0.0)
ELEVATION: Integer (0.0)
FEATCLASS: String (0.0)
FEATCODE: String (0.0)
GEONAMEID: String (0.0)
GTOPO30: Integer (0.0)
LATITUDE: Real (0.0)
LONGITUDE: Real (0.0)
MODDATE: String (0.0)
NAME: String (0.0)
POPULATION: Real (0.0)
TIMEZONE: String (0.0)
```

Ogrinfo-kysely on nopea, jos jättää siitä ulottuvuuksien selvittämisen pois käyttämällä -noextent-parametria.

```
ogrinfo ES: allcountries -so -noextent
INFO: Open of `ES:'
      using driver `ElasticSearch' successful.

Layer name: allcountries
...
```

Tarkistus suoraan ElasticSearch-tietokannasta selaimella

http://localhost:9200/allcountries

Vastaus tulee JSON-muodossa:


```
{"allcountries":{"aliases":{},"mappings":{"FeatureCollection":{"_meta":
{"fid":"ogc_fid","geomfields":{"geometry":"POINT"}},"properties":{"geometry":
{"type":"geo_shape"},"ogc_fid":{"type":"long"},"properties":{"properties":
{"ADMIN1":{"type":"string","store":true},"ADMIN2":
{"type":"string","store":true},"ADMIN3":{"type":"string","store":true},"ADMIN4":
{"type":"string","store":true},"ALTNAMES":
{"type":"string","store":true},"ASCIINAME":{"type":"string","store":true},"CC2":
{"type":"string","store":true},"COUNTRY":
{"type":"string","store":true},"ELEVATION":
{"type":"integer","store":true},"FEATCLASS":
{"type":"string","store":true},"FEATCODE":
{"type":"string","store":true},"GEONAMEID":
{"type":"string","store":true},"GTOPO30":
{"type":"integer","store":true},"LATITUDE":
{"type":"double","store":true},"LONGITUDE":
{"type":"double","store":true},"MODDATE":{"type":"string","store":true},"NAME":
{"type":"string","store":true},"POPULATION":
{"type":"double","store":true},"TIMEZONE":
{"type":"string","store":true}}},"type":
{"type":"string","store":true}}}},"settings":{"index":
{"creation_date":"1440756967429","uuid":"uYAygBwaSDGpQkg-
AlLywQ","number_of_replicas":"1","number_of_shards":"5","version":
{"created":"1070199"}}},"warmers":{}}}
```


Kysely, joka näyttää ElasticSearch-tietokannasta paljon hyödyllisiä tietoja:

```
http://localhost:9200/allcountries/_status?pretty=true
```

Tilantarve

Indeksin käyttämän levytilan määrän saa selville status-kyselyllä http://localhost:9200/allcountries/_status?pretty=true. Koska oletusarvo tietokantaan tallennetaan sekä indeksoitavat dokumentit kokonaisuudessaan että jokaiselle kentälle luodut indeksit, niin tilantarve on varsin suuri.

AllCounties.txt: 1309395816 tavua
Tilantarve indeksissä: 4201190039 tavua

Vertailu marraskuun 2016 testistä (11163481 paikannimeä):

AllCounties.txt: 1402855067 tavua
Tilantarve indeksissä: 5357139726 tavua

Pistekohteiden tallentamiseen geo_point käyttää hieman vähemmän tilaa kuin geo_shape.

Kyselyiden teko ElasticSearch:sta selaimella tai cURL:lla

Yksinkertaisin kyselytapa on ”URI-Search”
https://www.elastic.co/guide/en/elasticsearch/reference/current/search-uri-request.html
Esimerkiksi:

```
http://localhost:9200/allcountries/_search?
q=properties.NAME:Tampere&pretty=true
```

Tavallisempaa on lähettää kysely esimerkiksi wget:llä tai cURL:lla ja liittää kysely mukaan json-muodossa:

```
curl -XGET "http://localhost:9200/allcountries/_search" -d "{\"query\":
{ \"fuzzy\": {\"properties.NAME\":
{\"value\":\"Hellinki\",\"fuzziness\":2} } } }"
```

Kuten esimerkistä näkyy niin kysely on ikävä kirjoittaa erityisesti Windows:ssa koska lainausmerkeillä on erityismerkitys joten niiden eteen on laitettava escape-merkki \.
Kätevämpää on tallentaa kyselyrunko tiedostoon ja osoittaa se curl:lle @tietosto -syntaksilla.
HUOM! Tiedostot on tallennettava UTF-8 -muodossa tai muuten ei-ASCII-merkit johtavat virheeseen! Varminta on vielä tallentaa ilman BOM-merkkiä (Byte order marker).

```
fuzzy.json:

{"query":
  {
      "fuzzy": {"properties.NAME":{"value":"Äänekossi","fuzziness":2}}
  }
}
```

```
curl -XGET "http://localhost:9200/allcountries/_search" -d @fuzzy.json
```

ElasticSearch-kyselyiden tekoa voi opetella esimerkiksi tämän oppaan perusteella:
http://okfnlabs.org/blog/2013/07/01/elasticsearch-query-tutorial.html

Alla muutamia kyselyitä, jotka on testattu toimiviksi harjoittelutietokannalla.

```
wildcard.json:

{"query":
  { "wildcard": {
     "properties.ASCIINAME":"Hels*"
     }
  }
}
```

```
match.json:

{"query":
  { "match": {
     "properties.NAME": "Helsinki"
       }
  }
}
```

envelope.json:

```
{
    "query": {
        "geo_shape": {
            "geometry": {
                "shape": {
                    "type": "envelope",
                    "coordinates": [[14, 52],[14.1, 52.2]]
                }
            }
        }
    }
}
```

Huom! Koordinaatit annetaan järjestyksessä pituusaste-leveysaste (lon-lat). Taustalla on jotain hämärää, josta ElasticSearch-ohjeissa on oma kappalekin
https://www.elastic.co/guide/en/elasticsearch/guide/current/lat-lon-formats.html

Kyselyiden teko ElasticSearch:sta GDAL:lla

GDAL:in ogrinfo ja ogr2ogr-ohjelmien -where- ja -sql-parametrit tukevat ElasticSearch:in omia json-muotoisia query- ja filter-juttuja. Myös aluerajauksen antaminen -spat-parametrilla johtaa siihen, että rajaus tehdään ElasticSearch:ssä ja on siis nopeaa. Tavalliset SQL-kyselyt käsitellään GDAL:in puolella, mikä on hidasta koska kaikki data on ensin luettava ulos ElasticSearch:stä. GeoNames-tietokannalla tämä merkitsee sitä, että mikä tahansa SQL-kysely kestää vähintään puoli tuntia. Kannattaa siis opetella tekemään ElasticSearch:in omia kyselyitä.

```
ogrinfo ES: allcountries -where "{\"query\": { \"match\":
{ \"properties.NAME\": \"Helsinki\" } } }"
```

```
ogrinfo ES: allcountries -where @e:\elastic\match.json
```

```
ogrinfo ES: allcountries -spat 14 52 14.1 52.2
```

ElasticSearch-kyselyn upottaminen GDAL VRT-tiedostoon

Tallenna alla oleva VRT-tiedosto esimerkiksi nimellä ”elastic.vrt”

```
<OGRVRTDataSource>
    <OGRVRTLayer name="ES_with_filter">
        <SrcDataSource>ES:http://localhost:9200 allcountries</SrcDataSource>
        <SrcSQL dialect="ES">
        {
    "query": {
        "geo_shape": {
            "geometry": {
                "shape": {
                    "type": "envelope",
                    "coordinates": [[14, 52],[14.1, 52.2]]
                }
            }
        }
    }
}
</SrcSQL>
    </OGRVRTLayer>
</OGRVRTDataSource>
```

Tämän jälkeen voit lukea tietoja määritetyn virtuaalitason ”ES_with_filter” kautta. Tästä pitäisi olla hyötyä esimerkiksi sitten, kun käyttöön tulee QGIS, joka on käännetty GDAL 2.1 -version kanssa. Ennen kuin QGIS:iin on tehty ”ES Query Plugin” niin VRT-konsti on todennäköisesti ainoa tapa saada suodatettua ElasticSearch-dataa QGIS:n kartalle.

```
ogrinfo elastic.vrt
INFO: Open of `elastic.vrt'
      using driver `OGR_VRT' successful.
1: ES_with_filter (Point)
```

```
ogrinfo elastic.vrt ES_with_filter -so
INFO: Open of `elastic.vrt'
      using driver `OGR_VRT' successful.

Layer name: ES_with_filter
Geometry: Point
Feature Count: 56
Extent: (13.999590, 52.000000) - (14.100870, 52.200000)
Layer SRS WKT:
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
….
```

ElasticSearch:in tukemia hakuvaihtoehtoja

- Täsmällinen osuma **”Helsinki”**
- Jokerimerkit **”Helsi*”**. Jokerimerkit ovat * ja ?. Haku ei ole tehokas jos jokerimerkkin on ensimmäisenä **”*elsin*”.**
- Sumea haku vähän sinne päin **”Hellingi”**. Sumeuden aste annetaan fuzziness-parametrillä, esim. ”fuzziness”:2 merkitsee ”Etsi Hellingi ja kaikki merkkijonot, jotka eroavat Hellingi:stä korkeintaan kahden merkin verran.
- Hakutermien läheisyys: Jukka + Rahkonen, sitä parempi mitä lähempänä toisiaan; haluttaessa myös Rahkonen + Jukka otetaan mukaan.
- Spatiaalihaut: Alueen sisällä (Envelope tai vapaamuotoinen polygoni), etäisyys
- Kaikki yhdistelmät
- Hakutekijöiden painottaminen: oikea kaupunki =3x painotus, oikea nimi 1x painotus; etäisyysjärjestys paikan x suhteen
- Haut voidaan tehdä tietokannan kaikkien indeksien kaikista kentistä tai rajatusti

Onko ElasticSearch oikeasti nopea?

Kymmenen miljoonan pisteen GeoNames-aineistolla tehtyjen kokeilujen perusteella, on se. Luotettavien nopeustestien tekeminen sovelluksesta, joka on rakennettu käyttämään välimuistia niin paljon kuin mahdollista, on vaikeaa, mutta suuntaa antavasti Envelope-kysely on erittäin nopea, 10-200 millisekuntia. Match-kysely on myös nopea, noin 200-300 ms. Sumea fuzzy-kysely jonka luulisi olevan raskas, on selvästi hitaampi, mutta ei kuitenkaan kestä yli 2 sekuntia. Kaikki testit on tehty kannettavalla tietokoneella ja niin, että ElasticSearch-tietokanta on asennettu ulkoiselle USB2-kovalevylle, mikä ei varmaan ole nopein mahdollinen kokoonpano.

Miksi ElasticSearch eikä PostgreSQL Full Text Search?

DevALPO-tilaisuudessa esitettiin tämä kysymys, johon en osaa vastata kunnolla. Vaikuttaa siltä, että kummallekin tavalle löytyy käyttökohteensa. ElasticSearch on ilmeisesti helpompi laajentaa kun on kyse todella suurista ja kuormitetuista hakupalveluista. Se vaikuttaisi myös olevan luotettavasti nopea oletusarvoillaan, kun taas netistä löytyvät vastaukset Full text search:in hitauteen yleensä kehottavat analysoimaan PostgreSQL-suorituksia ja luomaan indeksit jollain toisella tavalla. Suurten hakupalvelujen käyttäjät myös arvostavat sitä, että hakupalvelun kuorma on erotetty tuotannossa olevasta relaatiokannasta. Mutta jos hakutarpeet pystyy tyydyttämään suoraan PostgreSQL-kannasta, niin ElasticSearch:in ylläpito ja tietojen päivittäminen relaatiokannasta hakukokeen kantaan tuottaa vain ylimääräistä työtä.

Viitteitä:

http://www.slideshare.net/billkarwin/full-text-search-in-postgresql

- Epäilen nopeustestien tuloksia. Testiaineisto on liian pieni (1,2 miljoonaa riviä) ja esitetyt numerot epäilyttävän tarkkoja. En väitä että tulokset laittaisivat tietokannat väärään järjestykseen, mutta annetut tiedot koejärjestelystä ovat liian ylimalkaiset jotta kokeen voisi toistaa. Omissa kokeissani nopeus vaihteli varsin paljon peräkkäisillä suorituksilla vaikka hakuehtoja muutti joka kerta, puhumattakaan saman kyselyn toistamisesta, jolloin myöhemmät haut selvästikin tulevat lähes suoraan välimuistista.

http://stackoverflow.com/questions/10875674/any-reason-not-use-postgresqls-built-in-full-text-search-on-heroku