# JURUSAN TEKNI DEPARTEMEN INFORMATIKA

**FAKULTAS TEKNOLOGI INFORMASI DAN KOMUNIKASI**

**INSTITUT TEKNOLOGI SEPULUH NOPEMBER**

**USULAN TUGAS AKHIR**

# IDENTITAS PENGUSUL

**NAMA : Nafiar Rahamnsyah**

**NRP : 05111440000109**

**DOSEN WALI : Prof.Ir. Supeno Djanali, M.Sc.,Ph.D**

**DOSEN PEMBIMBING : 1. Nurul Fajrin Ariyani, S.Kom.,M.Sc.  
 2. Abdul Munif, S.Kom.,M.Sc.**

# JUDUL TUGAS AKHIR

“Implementasi algoritma Vertex Signature Tree dalam optimasi SPARQL query ”

# LATAR BELAKANG

*Resource Description Framework* (RDF) telah diusulkan sebagai data model utama dalam pengembangan aplikasi Semantic Web. RDF sendiri sudah digunakan pada berbagai macam aplikasi. Contohnya YAGO2 dan DBpedia yang mengekstrak data – data pada Wikipedia dan menyimpannya dalam format RDF agar data yang telah disimpan dapat didapatkan kembali dengan menggunakan *query* yang terstruktur. Selain itu masih banyak contoh lain seperti Bio2RDF [1] dan Uniprot RDF [2] yang menggunakan RDF data model untuk menyimpan data – data hasil eksperimen. Sehingga saat ini data RDF yang sudah dibuat pada jaringan internet sudah sangat besar dan akan semakin membesar.

Secara umumnya, data RDF dapat direpresentasikan sebagai kumpulan *triple* yang dilambangkan sebagai SPO (*Subject Predicate Object*) dimana setiap satu *triple* dapat digambarkan sebagai *node* atau *relationship* terhadap *node* yang lain sehingga secara keseluruhan data yang ada pada sebuah RDF dapat digambarkan sebagai graph. Dalam mendapatkan data yang ada pada sebuah RDF kita dapat menggunakan bahasa query SPARQL yang telah direkomendasikan oleh W3C. Sebagai contoh kita bisa mendapatkan individual yang telah lahir pada 12 Februari 1809, dan meniggal pada 15 April 1865 dari dataset RDF dengan query SPARQL sebagai berikut :

Q1 : Select ?name Where { ?m <hasName> ?name . ?m <BornOnDate> “1809-02-12” . ?m <DiedOnDate> “1865-04-15” .}

Walaupun cara memanajemen data RDF sudah dipelajari dalam satu dekade kebelakang, kebanyakan solusi yang sudah ada tidak dapat menghadapi data RDF dalam skala yang besar dan tidak dapat menjawab query SPARQL yang kommples secara efisien. Untuk menghadapi masalah tersebut terdapat beberapa Teknik yang sudah diusulkan sebelumnya, contohnya *vertical partitioning* [3], *multiple indexing* [4], *sextuple indexing* [5]*, graph partitioning* [6]. Walaupun begitu *query engine* yang sudah ada tersebut seperti SW-Store [7], Hexastore [5], RDF -3x [4] memiliki batasan seperti berikut : (1) sistem – sistem tersebut tidak dapat menghadapi SPARQL dengan *wildcard query* (2) pada beberapa sistem yang ada sangat sulit untuk menangani update pada sebuah repositori RDF, sehingga memaksa sistem tersebut untuk memproses dataset *from scratch* ketika ada update data.

Hasil Tugas Akhir ini diharapkan dapat memberikan metode yang lebih baik dalam indexing pada graph database sehingga mampu menghadapi permasalahan di atas dengan optimal dan dapat memberikan masukkan dalam perkembangan semantic web.

# RUMUSAN MASALAH

Rumusan masalah pada tugas akhir ini adalah sebagai berikut :

1. Bagaimana cara membuat basis data untuk menyimpan data graph yang didapatkan dari file RDF ?
2. Bagaimana cara membuat basis data yang dapat menjawab SPARQL dengan *wildcard query* ?
3. Bagaimana cara membuat basis data yang menerapkan indexing sehingga dapat menangani update baru pada sebuah dataset *?*

# BATASAN MASALAH

Batasan masalah pada tugas akhir ini adalah sebagai berikut :

1. Metode indexing menggunakan VS-Tree*.*
2. Percobaan yang dilakukan berupa membandingkan *running time* dalam waktu satuan detik.
3. Dataset yang digunakan adalah YAGO2, dan DBLP.
4. Membandingkan dengan sistem yang sudah ada yaitu SW-Store, RDF-3x, dan apache jena fuseki.
5. Sistem yang dibuat dapat melakukan insert data, update data, dan query data.

# TUJUAN PEMBUATAN TUGAS AKHIR

Tujuan dari pembuatan tugas akhir ini yaitu untuk mengetahui performa *runtime* dalam satuan waktu dari sistem basis data yang dibuat jika dibandingkan dengan sistem basis data yang sudah ada. Selain itu sistem ini juga dibuat untuk menjawab permasalahan *wildcard query.*

# MANFAAT TUGAS AKHIR

Tugas akhir ini diharapkan dapat membantu sebagai masukkan dari metode yang dapat digunakan untuk *preprocessing* wildcard query. Selain itu tugas akhir ini juga diharapkan dapat membantu sebagai masukkan dari metode yang dapat digunakan untuk optimasi *traverse* data graph. Dan tugas akhir ini juga diharapkan dapat memberikan manfaat dalam mengetahui performa basis data yang sudah ada dalam menghadapi SPARQL query terhadap dataset yang besar.

# TINJAUAN PUSTAKA

1. **Semantic Web**

Semantic web adalah sebuah visi, ide atau pemikiran dari bagaimana memiliki data pada web yang didefinisikan dan dihubungkan dengan suatu cara dimana dapat digunakan oleh mesin tidak hanya untuk tujuan display, tetapi untuk otomatisasi, integrasi dan penggunaan kembali data diantara berbagai aplikasi [8].

Semantic web adalah sebuah web dari data, seperti layaknya sebuah basis data global. Pendekatan Semantic web mengembangkan bahasa untuk mengekspresikan informasi dalam bentuk yang dapat diproses oleh mesin (*machine processable*). Ide dasarnya adalah untuk membuat web agar memiliki definisi dan link data sehingga dapat digunakan lebih efektif untuk mencari, otomasi, integrasi dan re-use informasi pada berbagai aplikasi.

1. **YAGO2**

Yet Another Great Ontology 2 (YAGO2) merupakan ekstensi dari YAGO knowledge base, dimana entitas, fakta, pristiwa dilabuhkan baik dalam ruang dan waktu. YAGO2 didapatkan secara otomatis dengan cara mengambil data dari Wikipedia, Geonames, dan WordNet. Dimana data ini berisikan 447 juta fakta tentang 9,8 juta entitas. Telah dilakukan evaluasi oleh manusia dan didapatkan 95% akurasi dari data yang ada [9].

1. **DBpedia**

DBpedia adalah usaha komunitas yang berkepentingan untuk mengambil informasi terstruktur dari Wikipedia dan membuat informasi ini tersedia di Web. DBpedia memungkinkan untuk mengajukan pertanyaan yang canggih terhadap Wikipedia, dan untuk menghubungkan kumpulan data yang berbeda di Web ke data Wikipedia. DBpedia diharapkan dapat mempermudah informasi dalam jumlah besar di Wikipedia untuk digunakan dengan beberapa cara menarik yang baru. Kedepannya, DBpedia bisa menjadi mekanisme baru untuk navigasi, hubungan, dan peningkatan ensiklopedia itu sendiri [10].

1. **SW-Store**

SW-Store adalah proyek yang baru diluncurkan yang tujuannya adalah untuk mengelola dan query data Semantic Web. Dimulai dari daftar bersih dan merancang DBMS khusus untuk jenis data dan model data Semantic Web yang lazim, yaitu Resouce Descriptive Framework, atau RDF. SW-Store mengeksplorasi bagaimana query dan aplikasi Semantic Web yang umum seperti penalaran dan integrasi data biologis dapat dibangun ke dalam database [7].

1. **RDF-3x**

RDF-3X (untuk RDF Triple eXpress), dirancang dan diimplementasikan mulai dari yang khusus untuk pengelolaan dan query data RDF. RDF-3X mengikuti alasan yang dianjurkan bahwa sistem manajemen data yang dirancang dan disesuaikan dengan domain aplikasi secara spesifik dapat mengungguli sistem mainstream generik dengan dua urutan besarnya [4].

1. **Apache Jena Fuseki**

Apache Jena Fuseki adalah server SPARQL. Jena Fuseki bisa berjalan sebagai layanan sistem operasi, seperti aplikasi web Java (file WAR), dan sebagai server mandiri. Jena Fuseki juga menyediakan keamanan (menggunakan Apache Shiro) dan memiliki antarmuka pengguna untuk pemantauan dan administrasi server. Jena Fuseki menyediakan protokol SPARQL 1.1 untuk query dan update serta protokol SPARQL Graph Store. Fuseki terintegrasi dengan TDB untuk menyediakan lapisan penyimpanan persisten yang kuat dan transaksional, dan menggabungkan kueri teks Jena dan kueri spasial Jena. Jena Fuseki dapat digunakan untuk menyediakan mesin protokol untuk query dan sistem penyimpanan RDF lainnya [11].

1. **SPARQL Query Dengan Wildcard**

Dalam aplikasi nyata, memiliki pengetahuan penuh tentang *query object* akan sangat tidak pratis. Dengan demikian, hal ini tidak memungkinkan untuk menentukan kriteria query secara spesifik. Sebagai contohnya kita mungkin tahu bahwa seorang politisi penting lahir pada 12 Februari dan meninggal pada 15 April, namun kita tidak mengetahui kelahiran dan kematiaannya secara detil. Dengan kasus ini kita harus melakukan query dengan wildcard. [12]

Q2:Select ?name Where { ?m <hasName> ?name. ?m <BornOnDate> ?bd. ?m <DiedOnDate> ?dd. FILTER regex(str(?bd), “02-12”), regex(str(?dd), “04-15”) }

Meskipun ada teknik untuk mendukung query SPARQL dengan wildcard dan cara untuk mengelola kumpulan data RDF yang besar, dari data – data yang saya teliti, tidak ada teknik untuk mendukung keduanya, yaitu kemampuan untuk melakukan query SPARQL dengan *wildcard* dengan cara yang terukur. Sistem penyimpanan RDF yang ada, seperti Jena, Yars2, dan Sesame 2.0, tidak dapat bekerja dengan baik dalam dataset RDF yang besar (seperti dataset Yago). SW-store, RDF-3x, dan Hexastore dirancang untuk mengatasi skalabilitas, namun mereka hanya dapat mendukung query SPARQL yang tepat, karena semuanya menggantikan semua literal (dalam triples RDF) dengan id menggunakan kamus pemetaan.

1. **S-Tree**

Pendekatan *signature* adalah metode akses untuk pengambilan cocok sebagian yang memenuhi banyak persyaratan lingkungan kantor. *Signature* adalah kata-kata atau string yang dihash menjadi binary code yang berasal dari objek yang tersimpan dalam basis data. Mereka berfungsi sebagai filter untuk pengambilan kembali untuk membuang sejumlah besar objek yang tidak memenuhi syarat. Dalam metode *indexed signature,* *signature* dari object - object yang tersimpan pada satu halaman digunakan untuk membentuk sebuah *signature* untuk halaman tersebut [13].

1. **Vertex Signature Tree**

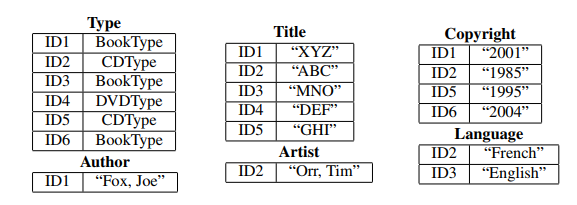
VS-Tree merupakan S-Tree yang ditambahkan aturan untuk mencatat ringkasan graph, yang digunakan untuk mengurangi ruang pencarian dari proses query Terdapat sebuah data graph *signature* G\*. Pertama – tama buat sebuah S-Tree terhadap semua vertex *signature* pada G\* (yaitu, V(G\*)). Terdapat sebuah query signature dinyatakan dengan q dan kumpulan data *signature* adalah {si}, query inklusi digunakan untuk mencari semua data *signature* si, dimana q&si = q. dalam permasalahan ini setiap *leaf entry* pada S-Tree adalah sebuah *vertex* *signature* pada G\* [12].

1. **Multiple Index**

Multiple Index dimotivasi oleh sifat data RDF dan kebutuhan untuk mendapatkan semua bentuk pola triple secara efisien pada waktu pemrosesan query. Secara intuitif, bergantung pada posisi triple mana yang diisi dengan IRI atau literal, akan jauh lebih efisien untuk mengakses triples menggunakan indeks yang lugas. Misalnya, pengikatan untuk variable ?x dalam pola tripple '?x p s', yaitu variabel pada posisi subjek, properti dan objek konstan, akan jauh lebih baik dengan 'pos' atau 'ops' daripada 'spo' atau ‘sop’. Tentu saja, seseorang hanya bisa menggunakan indeks yang didukung oleh sistem. Dengan demikian beberapa sistem mengadopsi beberapa solusi indeks agresif, yaitu hampir semua jalur akses dipertimbangkan, sementara yang lain menganggap bahwa beberapa pola grafik jarang ditemukan dalam situasi praktis dan hanya mendukung yang paling umum. Ini datang dengan biaya pemeliharaan yang tinggi saat data diperbarui pada bagian terdahulu (bagian penting dari indeks harus dijaga) dan ketidakmampuan untuk menangani beberapa pola tiga secara efisien untuk yang terakhir..

1. **Vertical Partitioning**

Pada vertical partitioning table untuk data triples dibuat ulang menjadi tabel dua kolom sebanyak *n,* dimana n adalah jumlah property unik pada sebuah data. Pada setiap table yang dibuat, kolom pertama berisi subject yang mendefinisikan property nya dan kolom kedua berisi nilai yang mendefinisikan subjectnya. Sebagai contohnya dapat dilihat pada gambar dibawah.



**Gambar a**. Tabel dua kolom

Setiap tabel disorting berdasarkan subjectnya, jadi subject tertentu dapat dicari secara cepat, dan merge join secara cepat dapat dilakukan untuk mendapatkan informasi dari beberapa subset property dari banyak subject. Niali kolom dari setiap table dapat diindex secara opsional. [3]

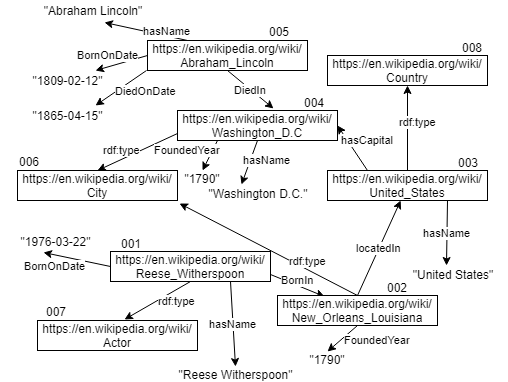
# RINGKASAN ISI TUGAS AKHIR

Tugas akhir yang akan dikerjakan berkaitan dengan penyelesaian permasalahan optimasi SPARQL query terkait *wildcard query* yang dapat dijawab oleh sebuah *query engine* dan indexing yang mendukung sedikit proses jika ada perubahan data pada dataset yang sudah ada. Sistem yang akan dibuat berupa query engine atau database berbasis graph sehingga sistem yang dibuat berdasarkan perspektif *storage scheme*. Dalam menyelesaikan permasalahan ini akan dilakukan pembuatan struktur indexing dan algoritma query.

**9.1. Struktur Indexing**



**Gambar 1.** Triples pada RDF

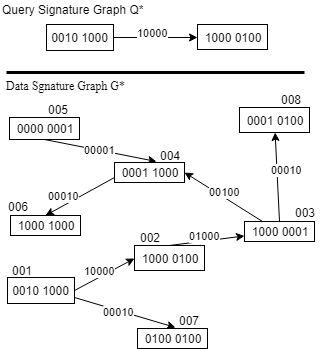
****

**Gambar 2.** Model Graph

Dalam implementasi tugas akhir ini pertama-tama kita akan memperlakukan data *triple* pada sebuah RDF sebagai data model graph. Pada Gambar 1 merupakan contoh data yang ada pada sebuah RDF dimana datanya berbentuk *triple* yang terdiri dari *Subject Predicate* dan *Object*. Data *triple* tersebut kita perlakukan dalam perspektif model data graph seperti pada Gambar 2., pada Gambar 2 merupakan bentuk model graph dari data *triple* pada Gambar 1. Graph model tersebut menjadi acuan skema penyimpanan untuk data pada sebuah RDF. Kemudian graph tersebut disimpan pada disk atau storage dalam berntuk *adjacency list*. Pada Gambar 3 merupakan *adjacency list* yang akan dibentuk berdasarkan model graph yang ada pada Gambar 2.



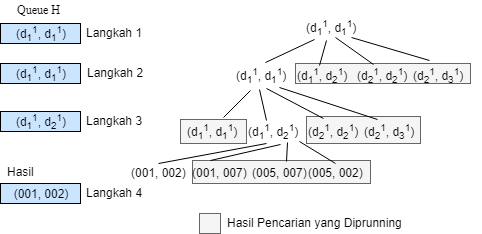
**Gambar 3.** *Adjacency List* yang Dibentuk

****

**Gambar 4.** Data Signature Graph

Kemudian graph pada RDF akan ditransformasi menjadi *data signature graph* dengan cara mengencode setiap entitas, class, dan vertex. Pada Gambar 4. adalah *data signature graph* yang merupakan hasil encode dari data graph pada Gambar 2. Setelah itu indexing VS-Tree akan dibuat terhadap semua data pada *data signature graph*.

**9.2. Algoritma Query**



Gambar 5. Proses Algoritma

Pada dasarnya permasalahan yang ada adalah bagaimana caranya mencari Q\* (*query signature graph*) pada G\* (*data signature graph*) secara efisien. Algoritma query yang digunakan menggunakan queue dalam proses pencarian untuk mendapatkan setiap bagian yang sesuai pada Q\* terhadap G\*. Pembuatan VS-Tree merupakan aturan *filtering* untuk query subgraph terhadap *data signature graph.* Aturantersebut dapat disingkronkan kedalam algoritma query sehingga dapat menjawab query SPARQL secara tepat dan *query wildcard* dengan cara yang seragam. Pada Gambar 5 merupakan contoh gambaran proses algoritma yang digunakan untuk query dengan menggunakan queue.

**9.3. Pengujian**

Pengujian pada tugas akhir ini dilakukan dengan cara membandingkan performa dari metode VS-Tree dengan metode *vertical partitioning* yang diimplementasikan pada SW-Store dan *multiple indexing* yang diimplementasikan pada RDF-3x. pada tahap ini akan didemonstrasikan performa dari segi runtime dari sistem yang telah dibuat dalam menghadapi query SPARQL yang sesuai aturan dan wildcard query, kemudian dibandingkan dengan sistem lain yang sudah ada yaitu SW-Store dan RDF-3x. Dan mendemonstrasikan performa sistem yang membutuhkan sedikit waktu terhadap update pada sebuah dataset.

# METODOLOGI

## Penyusunan proposal tugas akhir

Penyusunan proposal Tugas Akhir ini adalah tahap awal untuk memulai pengerjaan Tugas Akhir. Pada proposal ini, penulis mengajukan gagasan untuk mengimplementasikan algoritma Vertex Signature Tree dalam optimasi SPARQL query.

## Studi literatur

Pada tahap ini akan dilakukan pendalaman studi literatur yang bersumber utama dari *paper* yang berasal dari jurnal internasional dengan judul “*gStore: Answering SPARQL Queries via Subgraph Matching”*. Selain itu akan digunakan sejumlah referensi yang diperlukan dalam pembuatan aplikasi, yaitu mengenai penggunaan *S-Tree*.

## Analisis dan perancangan

Tahap ini meliputi perancangan sistem berdasarkan studi literatur dan pembelajaran konsep teknologi dari perangkat lunak yang ada. Tahap ini mendefinisikan alur dari implementasi. Langkah-langkah yang dikerjakan juga didefinisikan pada tahap ini. Pada tahapan ini dibuat prototype sistem, yang merupakan rancangan dasar dari sistem yang akan dibuat. Serta membuat pseudocode yang menjadi dasar – dasar algoritma yang akan digunakan.

## Implementasi perangkat lunak

Implementasi untuk melakukan percobaan ini dibangun dengan menggunakan algoritma *Vertex Signature Tree* dan dengan menggunakan Bahasa C, dan Java dengan fungsi – fungsi bantuan yang sudah ada.

## Pengujian dan evaluasi

Pada tahap ini dilakukan pengujian terhadap perangkat lunak yang telah dibuat. Pengujian dan evaluasi akan dilakukan dengan melihat performa *runtime* sistem perangkat lunak yang sudah dibuat dalam satuan waktu. Tahap ini dimaksudkan juga untuk mengevaluasi jalannya sistem, mencari masalah yang mungkin timbul dan mengadakan perbaikan jika terdapat kesalahan. Skenario pengujian pertama-tama dengan memasukkan dataset ke dalam sistem yang sudah dibuat, kemudian dilakukan query terhadapt dataset tersebut, dataset yang digunakan adalah YAGO2 dan DBLP kemudian dihitung waktu runtime untuk mendapatkan hasil query. Setelah itu dilakukan skenario yang sama terhadap sistem lainnya yaitu RDF-3x, SW-Store, dan Apache Jena Fuseki. Hasil akhir dibandingkan waktu runtime sistem yang sudah dibuat dengan sistem lainnya.

## Penyusunan buku tugas akhir

Pada tahap ini dilakukan penyusunan laporan yang menjelaskan dasar teori dan metode yang digunakan dalam tugas akhir ini serta hasil dari implementasi aplikasi perangkat lunak yang telah dibuat. Sistematika penulisan buku tugas akhir secara garis besar antara lain:

1. Pendahuluan
   1. Latar Belakang
   2. Rumusan Masalah
   3. Batasan Tugas Akhir
   4. Tujuan
   5. Metodologi
   6. Sistematika Penulisan
2. Tinjauan Pustaka
3. Desain dan Implementasi
4. Pengujian dan Evaluasi
5. Kesimpulan dan Saran
6. Daftar Pustaka

# JADWAL KEGIATAN

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Tahapan | 2017 | | | | | 2018 | | | | | | | | | | | | | | | | | | | |
| Desember | | | | | Januari | | | | | Februari | | | | | Maret | | | | | April | | | | |
| Penyusunan Proposal |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Studi Literatur |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Analisis dan perancangan |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Implementasi perangkat lunak |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Pengujian dan evaluasi |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| Penyusunan buku TA |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

1. **DAFTAR PUSTAKA**

|  |  |
| --- | --- |
| [1] | “Bio2RDF,” [Online]. Available: http://bio2rdf.org/. [Diakses 3 Januari 2018]. |
| [2] | “UniProt,” [Online]. Available: http://www.uniprot.org/format/uniprot\_rdf. [Diakses 3 januari 2018]. |
| [3] | D. J. Abadi, A. Marcus, S. R. Madden dan K. Hollenbach, “vertical partitioning,” dalam *Scalable semantic web data management using vertical partitioning*, Vienna, VLDB, 2007, pp. 411-422. |
| [4] | T. Neumann dan G. Weikum, RDF-3X: a risc-style engine for RDF, VLDB, 2008, pp. 647-659. |
| [5] | C. Weiss, P. Karras dan A. Bernstein, “sextuple indexing,” dalam *Hexastore: sextuple indexing for semantic web data management*, PVLDB, 2008, pp. 1008-1009. |
| [6] | Y. Yan, C. Wang, A. Zhou, W. Qian, L. Ma dan Y. Pan, Efficient Indices Using Graph Partitioning in RDF Triple Stores, IEEE, 2009. |
| [7] | D. J. Abadi, A. Marcus, S. R. Madden dan K. Hollenbach, “SW-Store,” dalam *SW-Store: a vertically partitioned DBMS for Semantic Web data management*, VLDB J., 2009, pp. 385-406. |
| [8] | T. Berners-Lee, J. Hendler dan O. Lassila, The Semantic Web, Scientific American, 2001. |
| [9] | J. Hoffart, F. M. Suchanek, K. Berberich dan G. Weikum, YAGO2: A spatially and temporally enhanced knowledge base from Wikipedia, Elsevier, 2013. |
| [10] | “DBpedia,” [Online]. Available: http://wiki.dbpedia.org/. [Diakses 3 Januari 2018]. |
| [11] | “Apache Jena Fuseki,” [Online]. Available: https://jena.apache.org/documentation/fuseki2/. [Diakses 3 Januari 2018]. |
| [12] | L. Zou, J. Mo, L. Chen, M. Tamer Özsu dan D. Zhao, “VS-Tree,” dalam *gStore: Answering SPARQL Queries via Subgraph Matching*, VLDB, 2011. |
| [13] | U. Deppisch, S-tree: a dynamic balanced signature index for office retrieval, SIGIR, 1986. |