BAB 2
LANDASAN TEORI

Pada Bab ini dijelaskan mengenai konsep *Semantic Web*, kerangka kerja yang digunakan untuk mendeskripsikan *Resource* yang dinamakan RDF, dan studi literatur yang membahas mengenai beberapa penelitian RDF *Storage program* sebelumnya.

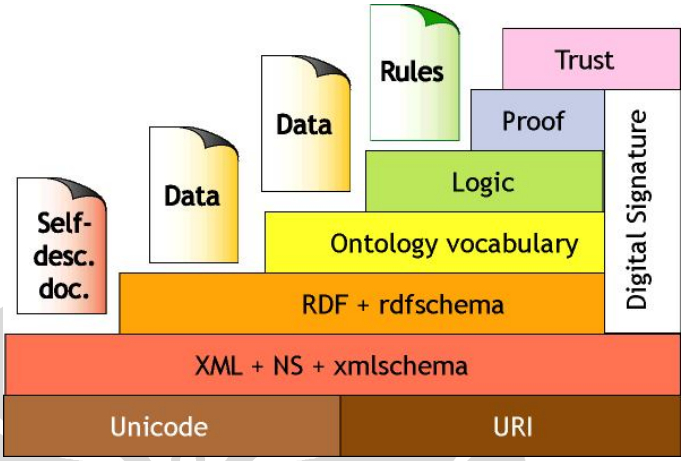
2.1 *Semantic Web*

Menurut Tim Berners-Lee dalam [28], konsep *Semantic Web* bertujuan mengembangkan mekanisme untuk mengekspresikan informasi yang dapat diproses, dimengerti, dan dimanfaatkan oleh mesin. Secara sederhana, *Semantic Web* dapat dipandang sebagai web tentang data, atau lebih seperti sebuah basis data yang memiliki cakupan global. Pada saat ini, konsep *Semantic Web* ini masih terus dikembangkan oleh tim khusus dalam *World Wide Web Consortium* (W3C)[1]. Tim ini bertugas untuk mengembangkan standar bahasa dan *tools* yang digunakan dalam *Semantic Web*.

Sean B. Palmer dalam [26] menyebutkan bahwa *Semantic Web* menyediakan metode untuk merepresentasikan data yang kita miliki dalam bentuk yang dapat dimanfaatkan untuk berbagai kepentingan. Dengan demikian, aplikasi *Semantic Web* akan mencakup beberapa bidang dan menyediakan layanan lebih banyak dibandingkan dengan aplikasi web biasa.

[1] http://www.w3c.org

Semantic Web adalah sebuah komposisi dari beberapa komponen-komponen yang tersusun seperti pada Gambar 2.1:



Gambar 2.1: Komponen Penyusun *Semantic Web* [30]

Tim Berners-Lee dalam [29] menjelaskan mengenai komponen penyusun *Semantic Web*:

1. *Uniform Resource Identifier* (URI): Sebuah URI adalah penanda untuk setiap *Resource* yang terdapat dalam *Semantic Web*. URI dapat berbentuk sebuah *Uniform Resource Locator* (URL) atau penanda unik dari sebuah *Resource*. URI dapat berbentuk nomor telepon, nomor ISBN, atau letak koordinat dari sebuah lokasi. Sebagai contoh adalah URI dari Google, yaitu `http://www.google.com`.

2. *Resource Description Framework* (RDF): RDF adalah model metadata untuk merepresentasikan *Resource* yang terdapat di web. RDF dijelaskan lebih detil pada sub bab 2.3.

3. RDF *Schema*: Komponen ini digunakan untuk mendefinisikan tipe data yang digunakan dalam berkas RDF. Dalam komponen ini, kita dapat membuat kelas yang dapat digunakan untuk mendeskripsikan *Resource* dan propertinya.
4. *Ontology*: Komponen *ontology* menyediakan mekanisme untuk mendefinisikan semantik tentang relasi antara *Resource* yang terdapat dalam berkas RDF.
5. *Rules/Query*: Komponen ini meyediakan kemampuan untuk mendapatkan *knowledge* yang tidak dinyatakan secara eksplisit dalam berkas RDF. Kemampuan ini dikenal sebagai *inference.*
6. *Logic*: Komponen ini mendasari kemampuan melakukan *inference* pada komponen *Rules/Query*.

2.2 *eXtensible Markup Language* (XML)

eXtensible Markup Language adalah *general-purpose markup language*. XML memberikan kesempatan kepada pengguna untuk mendefinisikan *tag* yang sesuai dengan kebutuhan. Tujuan utama dari XML adalah memfasilitasi distribusi data antar sistem yang berbeda-beda, misal melalui jaringan Internet.

XML merupakan salah satu bagian dari *Standard Generalized Markup Language* (SGML). SGML adalah standar internasional (ISO 8879) tentang definisi metode untuk merepresentasikan informasi dalam bentuk yang dapat dimengerti oleh manusia maupun mesin, yang tidak tergantung pada jenis perangkat, baik perangkat keras maupun lunak. XML banyak digunakan sebagai format berkas untuk standar-

standar lain, misal XHTML, RSS, MathML, GraphML dan *Scalable Vector Graphics* (SVG) [15].

Sebuah berkas XML dapat dikategorikan menjadi dua jenis. Kategori pertama adalah berkas XML yang *Well-Formed*. Pada kategori ini, sebuah berkas XML memenuhi semua aturan sintaks XML. Kategori kedua adalah berkas XML yang valid. Berkas XML yang valid tidak hanya benar secara sintaks XML, namun memenuhi aturan semantik yang tersimpan pada XML *Schema*.

2.3 *Resource Description Framework* (RDF)

RDF adalah spesifikasi kerangka kerja yang diterbitkan oleh *World Wide Web Consortium* (W3C) sebagai model metadata untuk merepresentasikan *Resource* yang terdapat di web. Ide dasar RDF adalah mendeskripsikan *Resource* dalam bentuk ekspresi Subyek-Predikat-Obyek (SPO), atau dikenal dengan **triplet** dalam terminologi RDF. [31, 32].

RDF memiliki 3 (tiga) konsep dasar, yaitu *Resources*, *Properties* dan *Statement*. *Resources* adalah obyek yang ingin direpresentasikan dalam *Semantic Web*. Setiap *Resources* memiliki URI sebagai penanda yang unik. *Properties* adalah sebuah *Resources* yang spesial karena digunakan untuk mendeskripsikan hubungan antar *Resources*. *Statement* adalah representasi properti dari sebuah *Resources* yang dinyatakan dalam bentuk triplet *object-attribute-value*. Triplet ini terdiri dari sebuah *Resource*, sebuah properti, dan sebuah nilai. Nilai yang dimiliki bisa berbentuk literal ataupun sebuah *Resources* lain [15].

Selain menggunakan XML, RDF juga menyediakan beberapa format *serialization* yang dapat digunakan untuk merepresentasikan *knowledge*:

Notation 3 (N3). Format ini diajukan oleh Tim Berners-Lee yang bertujuan untuk membuat berkas RDF lebih mudah dibaca dan dimengerti oleh manusia [27].

Turtle. Format ini merupakan sebuah subset dari **N3**, yang dikembangkan oleh Dave Beckett. Format ini lebih populer di kalangan pengembang *Semantic Web* sebagai alternatif yang lebih ramah kepada pengguna dibandingkan dengan format XML. Hampir semua perangkat lunak untuk RDF menyediakan kemampuan melakukan *parsing* ke format ini [6].

N-Triples. Format ini merupakan penyederhanaan dari format **Turtle**.

Graf. Format ini merepresentasikan RDF dalam bentuk graf berarah. Format ini merupakan yang paling mudah dimengerti oleh manusia, karena dapat menggambarkan hubungan antar *Resource* dengan jelas.

Berikut ini akan diberikan contoh representasi berkas RDF dalam format *serialization* yang telah dijelaskan:

▷ Representasi RDF/XML

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
            xmlns:dc="http://purl.org/dc/elements/1.1/"
            xmlns:ex="http://example.org/stuff/1.0/">
      <rdf:Description rdf:about="http://www.w3.org/TR/rdf-syntax-grammar"
           dc:title="RDF/XML Syntax Specification (Revised)">
            <ex:editor>
                 <rdf:Description ex:fullName="Dave Beckett">
                      <ex:homePage rdf:Resource="http://purl.org/net/dajobe/" />
                 </rdf:Description>
            </ex:editor>
      </rdf:Description>
</rdf:RDF>
```

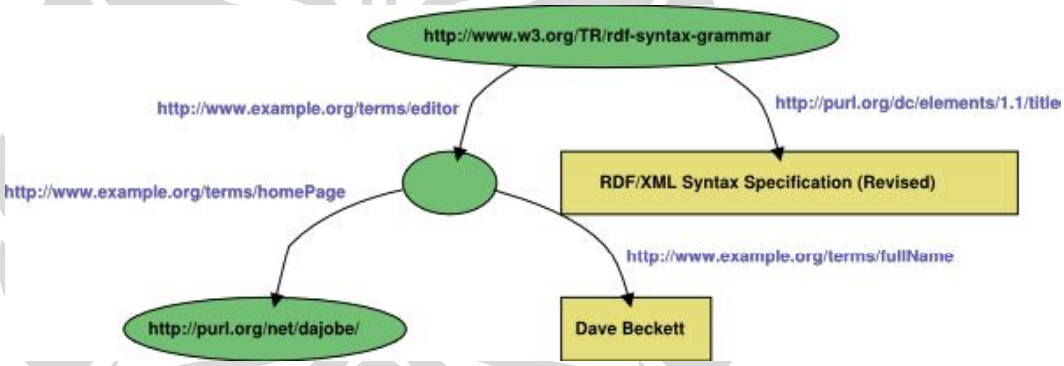
▷ Representasi Turtle

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ex: <http://example.org/stuff/1.0/> .
<http://www.w3.org/TR/rdf-syntax-grammar>
  dc:title "RDF/XML Syntax Specification (Revised)" ;
  ex:editor [
    ex:fullname "Dave Beckett";
    ex:homePage <http://purl.org/net/dajobe/>
  ] .
```

▷ **Representasi N-Triples**

```
<http://www.w3.org/TR/rdf-syntax-grammar>
      <http://purl.org/dc/elements/1.1/title> "RDF/XML Syntax Specification (Revi-
sed)" .
_:genid1 <http://example.org/stuff/1.0/fullName> "Dave Beckett" .
_:genid1 <http://example.org/stuff/1.0/homePage> <http://purl.org/net/dajobe/> .
<http://www.w3.org/TR/rdf-syntax-
grammar> <http://example.org/stuff/1.0/editor> _:genid1 .
```

▷ **Representasi Graf**



Gambar 2.2: Representasi Graf [33]

Untuk memperoleh informasi dalam data yang tersimpan dalam berkas RDF, diperlukan sebuah bahasa kueri yang dibuat khusus untuk data RDF. Agar mudah digunakan, bahasa kueri ini memiliki sintaks yang mirip dengan SQL[2]. Bahasa kueri menyediakan fitur untuk menghasilkan himpunan *statement* yang lebih kecil, ataupun untuk membandingkan data. Kemampuan lebih lanjut dari bahasa kueri ini menjadi

[2] *Structured Query Language*, bahasa kueri yang digunakan untuk memperoleh informasi dari dalam basis data.

dasar dari kemampuan *inference*. Bahasa kueri yang saat ini banyak digunakan dan telah menjadi standar *de facto* dinamakan **SPARQL**[3].

Penggunaan RDF dalam dunia nyata terlihat dari berbagai macam standar industri yang menggunakannya sebagai mekanisme representasi metadata. Frank Minola dan Eric Miller dalam [32] menjelaskan beberapa standar industri tersebut:

1. **Dublin Core Metadata.**

   Dublin Core Metadata adalah kumpulan elemen/properti yang dapat digunakan untuk mendeksripsikan sebuah berkas. Kumpulan set ini pertama kali dibuat pada Maret 1995 saat berlangsungnya *Metadata Workshop* di Dublin, Ohio. Saat ini, kumpulan properti ini masih terus dikembangkan oleh *Dublin Core Metadata Initiative* (DCMI)[4]. Dublin Core Metadata digunakan secara luas dalam mendokumentasikan berkas yang terdapat di Internet. Metadata ini akan dimanfaatkan oleh *crawler*[5] yang digunakan dalam mesin pencari informasi untuk mencari *Resource* dari Internet. Kumpulan elemen yang termasuk dalam Dublin Core Metadata didefinisikan dalam Dublin Core Metadata Element Set [9].

2. **PRISM.**

   Publishing Requirement for Industry Standard Metadata (PRISM)[6] adalah spesifikasi metadata yang dikembangkan oleh praktisi industri penerbitan. Berbagai penerbitan dan vendor membentuk PRISM *working group* untuk mengidentifikasi kebutuhan metadata yang digunakan dalam industri dan mengembangk-

[3] http://www.w3.org/TR/rdf-sparql-query/
[4] http://dublincore.org
[5] Kode program yang digunakan untuk mengunduh halaman web yang terdapat di Internet.
[6] http://www.prismstandard.org/

an spesifikasi yang cocok untuk kebutuhan tersebut. Kebutuhan akan metadata ini didasarkan pada keadaan dimana sebuah penerbit menggunakan isi yang sama dalam beberapa format penerbitan. Sebagai contoh, sebuah koran memiliki dua jenis format, yaitu format *online* dan format cetak.

Penggunaan beberapa format penerbitan memberikan kemungkinan terjadinya perbedaan interpretasi terhadap isi, yang menyebabkan perbedaan luaran dari masing-masing format. Untuk menghindari hal tersebut, spesifikasi PRISM [23] memanfaatkan standar yang telah banyak digunakan seperti XML, RDF, Dublin Core Metadata dan beberapa standar ISO lainnya. Penggunaan metadata dalam isi ini memberikan keuntungan bagi penerbit karena isi tersebut dapat digunakan kembali dengan mudah dan orisinalitasnya dapat dijaga dari gangguan pihak lain.

3. **XPackage.**

   XML *Package* (XPackage) adalah sebuah kerangka kerja yang dikembangkan untuk mendefinisikan informasi mengenai pengelompokan beberapa *Resources* agar dapat digunakan sebagai sebuah unit. Pengelompokan ini dikenal dengan istilah *packages*. Dengan menggunakan XPackage, *Resources* dalam sebuah *packages* dapat dideskripsikan dengan mudah, properti dari *Resources* tersebut, dan hubungan *Resources* tersebut dengan *Resources* lain yang tersimpan dalam *packages* yang sama. Spesifikasi XPackage [13] tersusun dari standar XML, RDF, dan XML *Linking Language*.

4. **RSS 1.0 RDF *Site Summary*.**

   Internet menyediakan berbagai macam informasi dalam jumlah yang besar dan memiliki skala yang luas. Informasi yang tersedia ini dapat diperbaharui dalam waktu yang sangat cepat, hanya dalam hitungan detik. Hal ini menyebabkan

sebagian besar pengguna mengalami kesulitan dalam memilah-milah informasi yang dibutuhkan dan kegiatan ini menghabiskan banyak waktu sehingga produktivitas pengguna jauh menurun.

RSS 1.0 adalah sebuah aplikasi RDF yang bertujuan untuk mengatasi permasalahan tersebut. RSS 1.0 menyediakan cara yang sederhana untuk merepresentasikan informasi yang diperoleh dari Internet. RSS 1.0 adalah salah satu aplikasi RDF yang sangat banyak digunakan dalam dunia nyata. Standar RSS 1.0 diterbitkan oleh RSS-DEV Working Group yang dikeluarkan pada Mei 2001 [24].

Dengan menggunakan RSS, informasi dari Internet dapat dikategorikan dengan mudah. Dengan menggunakan aplikasi yang dinamakan **RSS Readers**, pengguna dapat menentukan apakah sebuah informasi sesuai dengan keinginan dan tidak perlu lagi bersusah payah untuk membuka halaman web. Informasi yang tersimpan dalam RSS juga dapat diperbaharui secara otomatis dalam jangka waktu tertentu tanpa harus mengunjungi halaman web tersebut.

Pada saat ini, RSS lebih dikenal sebagai *Really Simple Syndication* atau seringkali disebut sebagai RSS 2.0. Standar ini dikembangkan oleh RSS *Advisory Board* yang tidak menggunakan RDF sebagai kerangka kerjanya.

5. ***Gene Ontology Consortium* (GOC).**

   Salah satu bidang ilmu yang memanfaatkan RDF secara luas adalah bidang medis dan biologi. Bidang ini memiliki data dalam jumlah yang sangat besar, misalnya data tentang gen dan senyawa kimia yang terdapat dalam makhluk hidup. Data ini tersimpan dalam berbagai format yang menyulitkan para peneliti untuk memanfaatkan data tersebut. Untuk itu, diperlukan sebuah representasi yang dapat menangani perbedaan format tersebut.

Gene Ontology Consortium adalah badan yang pada awalnya terbentuk dari kolaborasi 3 (tiga) model basis data organisme, yaitu Flybase, *Saccharomyces Genome Database* (SGD), dan *Mouse Genome Database* (MGD). GOC bertujuan untuk mengembangkan spesifikasi yang dapat digunakan untuk merepresentasikan gen yang berasal dari basis data yang berbeda secara konsisten.

Saat ini, GOC telah mengembangkan 3 (tiga) buah ontologi yang mendeskripsikan gen dari segi proses biologis, komponen sel dan fungsi molekular. Sebuah gen dapat memiliki beberapa fungsi molekular dan terlibat dalam satu atau lebih proses biologis. Sebuah gen juga dapat diasosiasikan dengan beberapa komponen sel. Ontologi-ontologi tersebut didefinisikan dalam sebuah berkas XML yang secara konsisten diperbaharui setiap bulannya [14].

6. ***Friend Of A Friend* (FOAF).**

   FOAF adalah sebuah proyek yang bertujuan untuk mengembangkan spesifikasi untuk mendeksripsikan data individu, hubungan pertemanan antara individu dan pekerjaan individu tersebut. Secara sederhana FOAF adalah profil individu yang tersimpan dalam format yang dapat dimengerti oleh mesin (*machine-readable*). FOAF dimulai sejak tahun 2000 oleh Libby Miller dan Dan Brickley [12]. Dalam berkas FOAF, pengguna dapat mendeskripsikan data dirinya dan hubungan pertemanan dengan pengguna FOAF yang lain. Hubungan ini direpresentasikan dengan menggunakan penanda seperti FOAF URI, alamat e-mail, alamat halaman web pribadi, bahkan *account instant messenger*.

7. **MusicBrainz.**

   MusicBrainz adalah proyek berbasiskan komunitas yang menyimpan metadata musik. Metadata musik adalah informasi tentang artis, judul album, dan lagu-lagu yang terdapat dalam album tersebut. Semua informasi ini dikumpulkan dan diberikan kepada publik. Basis data ini dapat dimanfaatkan oleh pengguna untuk mencari informasi mengenai hasil karya musik terbaru yang dihasilkan oleh seorang artis, dan sebagainya. Aplikasi pemutar musik juga dapat memanfaatkan informasi dari MusicBrainz untuk kemudahan penggunanya dalam menikmati musik.

2.4 Struktur Data

Pada sub bab ini dijelaskan mengenai struktur data yang digunakan pada setiap RDF *Storage program*. Stuktur data yang dibahas meliputi *Hash Table*, B-*tree*, B+-*tree*, dan AVL-*tree*.

2.4.1 *Hash Table*

Hash table adalah struktur data yang berisi asosiasi antara *key* dan *value*. Struktur data ini mendukung operasi pencarian data yang efektif. Setiap *key* akan ditransformasikan dengan sebuah fungsi *hash*[7] yang menghasilkan sebuah indeks. Indeks ini digunakan sebagai penunjuk ke lokasi *value*.

[7] Fungsi yang memetakan sebuah nilai ke indeks yang menunjukkan lokasi nilai tersebut.

Hash table memiliki kompleksitas sebesar $O(1)$ untuk operasi memasukkan data. Sementara kompleksitas[8] rata-rata yang dibutuhkan untuk mencari data adalah sebesar $O(1)$. Sedangkan untuk *worst-case*, kompleksitas waktu adalah sebesar $O(n)$.

Walaupun *hash table* membutuhkan waktu rata-rata yang konstan dalam operasi pencarian, waktu yang dihabiskan dalam beberapa kasus tidaklah kecil. Proses evaluasi terhadap fungsi *hash* dapat memperlambat operasi tersebut. Untuk beberapa kasus, penggunaan struktur data yang lebih sederhana seperti *array* ataupun penggunaan *binary search tree* akan menghasilkan kinerja yang lebih cepat.

2.4.2 B-*tree*

B-*tree* adalah struktur data pohon yang menjaga data di dalamnya tetap terurut. Struktur data ini mendukung operasi pencarian, penambahan, penghapusan data dalam kompleksitas waktu logaritmik. B-*tree* banyak digunakan dalam program basis data dan sistem berkas.

B-*tree* diciptakan oleh Rudolf Bayer dan Ed McCreight. B-*tree* dengan derajat m (jumlah *key* di setiap *node*[9]) memiliki properti sebagai berikut:

- Setiap *node* memiliki anak dengan jumlah $\leq m$ buah.
- Setiap *node root* dan *leaves* memiliki anak dengan jumlah $\geq m$ buah.

[8] Representasi batas atas waktu yang dibutuhkan untuk menyelesaikan operasi.
[9] Elemen Pohon yang memiliki anakan.

- ▷ Setiap *root*[10] memiliki anak sedikitnya 2 buah.
- ▷ Semua *leaves*[11] memiliki level yang sama.
- ▷ Setiap *non-leaf node* dengan jumlah anak k buah memiliki *key* $k-1$ buah.

Dalam B-*tree*, setiap *node* internal dapat memiliki *node* anak dalam jumlah yang bervariasi, namun dibatasi dalam sebuah *range*. Saat penambahan data dilakukan, maka jumlah *node* anak akan bertambah. Jika jumlah *node* anak telah melewati batas, dilakukan operasi penggabungan dan pemisahan. B-*tree* tidak memerlukan dilakukannya operasi *re-balancing*[12] sebanyak struktur data pohon lainnya, namun B-*tree* menghabiskan lebih banyak ruang diska.

2.4.3 B+-*tree*

B+-*tree* adalah struktur data pohon yang ditemukan oleh Rudolf Bayer dan Edward M. McCreight pada tahun 1972. B+-*tree* memiliki keunggulan pada efisiensi ruang diska dibandingkan B-*tree*. Pada B+-*tree*, semua *record* disimpan pada level paling bawah dari pohon. Sementara *key* disimpan pada *node*.

Sebuah B+-*tree* dengan derajat b dengan level indeks sebesar h memiliki karakteristik sebagai berikut:

- ▷ Jumlah maksimum *record* yang dapat disimpan adalah $n = b^h$.

[10] Elemen pohon yang memiliki level paling atas, tidak memiliki orang tua.
[11] Elemen pohon level yang paling bawah. Tidak memiliki anakan.
[12] Operasi untuk menyeimbangkan sub pohon.

- ▷ Jumlah minimum *key* adalah $2(b/2)^{(h-1)}$.
- ▷ Besar ruang diska yang dibutuhkan untuk menyimpan pohon adalah $O(n)$.
- ▷ Operasi penambahan data membutuhkan waktu $O(log_b n)$.
- ▷ Operasi pencarian data membutuhkan waktu $O(log_b n)$.
- ▷ Operasi menghapus data membutuhkan waktu $0(log_b n)$.

B+-*tree* digunakan pada sistem berkas *New Technology File System* (NTFS), Reiser-FS, dan JFS2 sebagai indeks. Aplikasi lain yang menggunakan B+-*tree* adalah sistem basis data relasional. Pada sistem basis data, B+-*tree* digunakan sebagai indeks tabel.

2.4.4 AVL-*tree*

AVL-*tree* ditemukan oleh G.M. Adelson-Velsky dan E.M. Landis pada tahun 1962. Pada AVL-*tree*, tinggi setiap 2 (dua) sub pohon memiliki selisih *height* maksimal satu. Operasi pencarian, penambahan dan menghapus data memakan waktu $O(logn)$, dimana n adalah jumlah *node* yang tersimpan dalam pohon . Operasi penambahan dan menghapus data mengharuskan dilakukannya operasi *re-balancing* terhadap pohon.

Balance factor dari sebuah *node* adalah selisih antara ketinggian dari sub pohon kanan dan sub pohon kiri. Sebuah *node* dengan *balance factor* 1, 0, dan -1 dikatakan sebagai *node* yang *balanced*. *Node* dengan nilai *balance factor* selain itu maka dikatakan sebagai *node* yang tidak *balanced*. Jika demikian, pohon perlu *re-balancing*.

2.5 Penelitian Terkait

Dave Beckett, et al. dalam [5, 7] melaporkan hasil survei beberapa RDF *Storage program* yang berlisensi *free software* dan *open source*. Pada bagian ini dipaparkan mengenai beberapa RDF *Storage program* yang telah dimuat pada survei tersebut. Beberapa RDF *Storage program* yang dijelaskan telah digunakan secara luas dalam berbagai macam aplikasi *Semantic Web*.

2.5.1 Jena *Semantic Web Toolkit*

Kevin Wilkinson, et al. dalam [19] memperkenalkan Jena2, generasi kedua pengembangan dari Jena. Jena merupakan salah satu *toolkit* yang populer dalam pengembangan aplikasi *Semantic Web*. Jena dikembangkan menggunakan bahasa pemograman Java. Jena memiliki kemampuan menyimpan representasi graf RDF pada sistem basis data *relational* dengan bantuan JDBC *driver*. Jena mendukung beberapa sistem basis data relasional populer (misalnya PostgreSQL, MySQL, dan Oracle) dan memiliki kemampuan migrasi antar sistem basis data.

Generasi awal dari Jena (selanjutnya disebut dengan Jena1) memiliki beberapa kelemahan dalam kinerja seperti terlalu banyak melakukan operasi *join*, penyimpanan *statement* hanya dalam satu buah tabel, penggunaan *storage* yang meningkat saat *reification*, dan pemrosesan kueri yang tidak optimal. Fokus dari pengembangan Jena2 adalah kinerja dan skalabilitas yang penting dalam proses menerapkan teknologi *Semantic Web* dalam aplikasi dunia nyata. Jena2 memiliki kemampuan merepresen-

tasikan standar RDF yang direvisi, beberapa tambahan fitur dan arsitektur internal program yang baru.

Jena2 memiliki beberapa fitur kunci dalam arsitekturnya, antara lain dukungan representasi banyak graf RDF yang fleksibel untuk pengembang aplikasi. Fitur ini memudahkan pengembang untuk mengakses dan memanipulasi terhadap data RDF, sehingg dapat diakses dalam bentuk triplet. Dengan arsitektur ini, skema basis data yang merepresentasikan data RDF dalam digunakan Jena2 memiliki kinerja lebih baik dari Jena1. Rencana pengembangan lebih lanjut dari Jena2 adalah menyimpan literal dalam bentuk perintah SQL, dukungan terhadap OWL, dan kemampuan *reasoning*.

2.5.2 Sesame

Jeen Broekstra, et al. dalam [18] mengembangkan Sesame, sebuah arsitektur untuk penyimpanan dan pengkuerian informasi dari dalam berkas RDF dan RDF *Schema*. Sesame dikembangkan oleh Aidministrator Nederland b.v. sebagai bagian dari proyek *European IST Project On-To-Knowledge*[13]. Sesame menyediakan fasilitas untuk penyimpanan data RDF dan informasi skema secara permanen, ditambah akses ke dalam informasi tersebut melalui proses pengkuerian.

Untuk penyimpanan data RDF secara permanen, dibutuhkan sebuah media repositori yang *scalable*. Sebagai solusinya, diputuskan untuk menggunakan *Database Management System* (DBMS). DBMS dipilih karena telah digunakan sejak lama untuk

[13] http://www.ontoknowledge.org/

menyimpan data dalam jumlah besar. Salah satu kelemahan dari solusi ini adalah banyaknya jumlah DBMS yang tersedia di pasaran dan masing-masing DBMS memiliki mekanisme tersendiri untuk menyimpan data, yang akan berakibat pada ketergantungan pada jenis DBMS tertentu. Untuk menghindari masalah tersebut, Sesame menyediakan satu lapisan data yang dinamakan *Storage And Inference Layer* (SAIL). SAIL adalah API yang menyediakan fungsi-fungsi spesifik untuk RDF dan menerjemahkan fungsi tersebut ke dalam instruksi DBMS yang bersesuaian. Keuntungan yang diperoleh dari penggunaan API ini adalah keleluasaan untuk menggunakan DBMS yang berbeda-beda tanpa harus mengganti komponen Sesame yang lain.

Sesame memiliki tiga modul fungsional yang berjalan sebagai klien untuk SAIL, yaitu modul RQL *Query Engine*, modul RDF *admin*, dan modul RDF *export*. Sesame menyediakan beberapa metode untuk berkomunikasi dengan modul-modulnya, antara lain dengan menggunakan metode *Hypertext Transfer Protocol* (HTTP), metode *Simple Object Access Protocol* (SOAP), ataupun *Remote Method Invocation* (RMI). Dalam *paper* ini disebutkan rencana pengembangan lebih lanjut dari Sesame antara lain adalah dukungan *transaction rollback support* pada SAIL, kemampuan *versioning*, dukungan DAML+OIL, dan penambahan beberapa modul fungsional dan metode komunikasi yang baru.

2.5.3 *Yet Another RDF Storage* (YARS)

A. Harth dan S. Decker dalam [2] menyebutkan bahwa hampir semua skema indeks yang tersedia saat ini (**Jena**, **Sesame**, **rdfDB**, **Redland** dan **Kowari**) tidak mendu-

kung pemberian kueri untuk data dari web, sehingga kualitas hasil yang dikeluarkan sangat rendah untuk beberapa kasus. Kekurangan lain adalah ketidakmampuan untuk merepresentasikan konteks sebuah informasi, yang memegang peranan penting dalam mewujudkan *trusted Semantic Web*. Penelitian ini mengambil pendekatan menggunakan teknik basis data untuk menyimpan dan membuat indeks untuk data RDF yang dapat meningkatkan kualitas hasil luaran dan kemampuan representasi dibandingkan dengan skema indeks lain.

Pada penelitian ini dikembangkan skema indeks yang lengkap, meliputi *full-text index* untuk triplet dalam RDF yang mengandung konteks. Skema indeks ini dinamakan sebagai *Yet Another RDF Storage* (YARS), sebuah skema indeks yang memiliki keunggulan dalam penggunaan memori sehingga dapat digunakan untuk aplikasi *embedded*. Penelitian ini menggunakan dataset FOAF dalam bentuk N3. Untuk proses temu kembali informasi, dikembangkan bahasa kueri yang dinamakan **YARS Query Language**.

Struktur data yang dipilih untuk skema indeks ini adalah B-*tree* dengan pertimbangan kemampuan stuktur data ini dalam mendukung operasi *insert*, *delete* dan pencarian data tertentu. Struktur skema indeks yang digunakan akan mendukung evaluasi terhadap kueri (*Select-Project-Join*). Skema indeks yang dimiliki akan terdiri dari dua bagian, yaitu Leksikon dan *Quad Indexes*. Leksikon menyimpan representasi *string* dan *Object Identifier* (`OID`) dari setiap *node* RDF. Dalam Leksikon, terdapat tiga buah indeks, indeks `nodeid`, indeks `oidnode` yang menyimpan pemetaan *string* ke OID, dan *keyword* indeks sebagai *inverted index*. *Quad Index* menyimpan semua kemungkinan pola kueri terhadap data RDF. Penyimpanan pola ini bertujuan untuk menghindari operasi *join* yang membutuhkan memori besar.

Pengujian terhadap skema indeks ini dilakukan dengan memberikan empat jenis kueri yang memiliki pola akses dan karakteristik berbeda. Pengukuran dilakukan terhadap waktu yang dibutuhkan untuk memperoleh jawaban dari kueri yang diberikan. Perbandingan dilakukan terhadap **Redland**, **Sesame dengan MySQL**, dan **Sesame** yang menggunakan basis data bawaan. Kinerja dari YARS lebih baik karena penggunaan *quad index* akan menghindari operasi *join*, yang membuat kinerja skema lain yang diujikan menurun cukup tajam.

2.5.4 Kowari

David Makepeace, et al. [8] menjelaskan bahwa keberadaan dari skema indeks yang khusus untuk data RDF disebabkan oleh ketidakmampuan sistem basis data yang telah ada saat ini dalam merepresentasikan data RDF dalam jumlah yang besar. Untuk menyimpan metadata dalam basis data membutuhkan desain tabel yang memiliki sedikit kolom, namun dengan isi yang sangat banyak. Kelemahan utama adalah untuk kueri biasa akan membutuhkan operasi *join* terhadap hampir semua tabel (*nested join operations*). Jika kueri yang diberikan meningkat kompleksitasnya, maka kinerja dari sistem tersebut akan jatuh.

Penelitian ini merekomendasikan sebuah skema indeks yang diberi nama **Kowari**. Skema indeks ini terdiri dari beberapa komponen, seperti *query layer* dan RDF *Statement Storage layer*. Skema indeks ini menggunakan AVL-*tree* sebagai struktur data internal dari indeks yang menyimpan semua RDF *statement*. Terdapat tiga buah indeks AVL-*tree* yang digunakan, masing-masing untuk Subyek, Predikat, dan Obyek.

Pemilihan AVL-*tree* didasarkan pada kompleksitas waktu yang dibutuhkan untuk melakukan operasi pencarian, penambahan dan penghapusan data adalah $O(log\, n)$, dimana n adalah jumlah *node* dalam pohon tersebut. Kelebihan lain dari struktur data ini adalah properti pohon yang menjamin keseimbangan data dalam pohon tersebut. Setiap indeks akan menyimpan informasi alamat dari sebuah berkas berukuran besar yang akan diakses secara acak. Struktur ini memungkinkan ukuran AVL-*tree* tidak terlalu besar dan dapat disimpan dalam memori.

Penelitian ini merekomendasikan struktur data AVL-*tree* untuk digunakan dalam penyimpanan data RDF yang pada umumnya kueri yang diberikan melibatkan satu atau lebih elemen dari sebuah triplet. Dari percobaan yang dilakukan, penelitian ini menyimpulkan bahwa struktur data AVL-*tree* dapat digunakan untuk menyimpan ratusan juta *statement* RDF dan informasi tersebut dapat diakses dalam waktu yang mendekati *real-time*. Pengembangan lebih lanjut dari skema indeks akan lebih terfokus struktur indeks yang lebih kompleks, seperti membuat *hash table* yang berisikan triplet yang telah terurut.

2.5.5 3Store

S. Harris dan N. Gibbins dalam [25] mengembangkan skema indeks yang dinamakan **3Store**. 3Store adalah sebuah pustaka yang dikembangkan dalam bahasa pemograman C. Pustaka ini menyimpan representasi data RDF dalam basis data MySQL.

Skema indeks ini dikembangkan khusus untuk menyimpan data RDF yang memiliki ukuran sangat besar. Skema indeks ini telah diuji menggunakan dataset `hypen.info`. Dataset ini merupakan kumpulan data mengenai peneliti, proyek, publikasi, dan institusi yang tergabung dalam riset dalam ilmu komputer yang terdapat di Inggris. Skema indeks ini juga dirancang untuk memiliki tingkat efisiensi yang tinggi dalam mengevaluasi kueri dan menambah pengetahuan baru.

Penelitian ini juga melakukan evaluasi terhadap beberapa skema indeks lain, seperti **Jena**, **Sesame**, **Parka**, **Redland**, dan **TAP**. Skema indeks yang diujikan memiliki beberapa kelemahan, terutama dalam kemampuan untuk menyimpan data RDF berukuran besar. Hal ini ditemukan pada Jena dan Parka. Kelemahan waktu eksekusi kueri yang terlampau lambat dimiliki oleh Sesame, Redland tidak mampu melakukan *matching* terhadap graf RDF, dan TAP tidak menyediakan bahasa kueri seperti RDQL.

3Store dikembangkan dengan arsitektur yang terdiri dari tiga lapisan, yaitu bagian yang melakukan *parsing* terhadap berkas RDF, lapisan yang menyimpan representasi RDF tersebut dalam triplet, dan struktur basis data. Fokus dari 3Store adalah kemampuan untuk menyimpan data RDF dalam jumlah besar. 3Store menggunakan program yang telah ada seperti **Raptor**[14] sebagai RDF *parser* dan menerjemahkan triplet ke dalam bentuk perintah SQL.

Pada lapisan struktur data, 3Store menggunakan sebuah fungsi *hash* yang digunakan sebagai *primary key* untuk setiap URI, *Resources* dan literal. Hasil fungsi ini akan digunakan juga sebagai *foreign key* dalam tabel triplet. Dengan demikian, 3store dapat

[14] http://librdf.org/raptor

menjamin bahwa semua *records* akan memiliki ukuran yang sama. Dengan metode ini, 3Store dapat memanfaatkan kemampuan sistem basis data untuk mengoptimasi representasi data RDF tersebut.

2.5.6 *Indexing Schemes* menggunakan *Suffix Arrays*

Akiyoshi Mitono, et al. [1] merekomendasikan sebuah skema indeks untuk RDF dan RDF *Schema* yang berfokus pada *path expression* yang diekstrak dari data RDF dan RDF *Schema*. Dalam penelitian ini data yang digunakan dibatasi pada RDF dan RDF *Schema* yang tidak mengandung siklus.

Skema indeks melakukan ekstraksi terhadap data RDF dan RDF *Schema* menjadi 4 (empat) jenis *Directed Acyclic Graphs* (DAG), yaitu **Predicates in Schema, Predicates in Resource descriptions**, **Class inheritance**, dan **Propery inheritance**. Dari DAG akan dilakukan ekstraksi terhadap semua kemungkinan *path* yang ada di dalamnya. Struktur data yang digunakan adalah sebuah *suffix array* yang akan menjadi sebuah *full-text* indeks yang akan menyimpan semua kemungkinan *string* dalam berkas.

Pengujian dilakukan terhadap data Wordnet yang disimpan dengan menggunakan **RDFSuite**. Pengukuran dilakukan terhadap waktu yang dibutuhkan untuk memproses data dengan menggunakan indeks yang telah dikembangkan dan indeks yang dibuat oleh RDFSuite.

2.5.7 BRAHMS

Maciej Janik dan Krys Kochut [21] mengembangkan sebuah skema indeks yang dinamakan **BRAHMS** (*a work**B** ench **R**DF store **A**nd **H** igh performance **M**emory **S**ystem*). Skema indeks ini terfokus pada proses perolehan informasi , lebih khusus lagi adalah informasi semantik dari entitas yang ada dalam dataset. Pada dua entiti dalam berkas RDF yang memiliki relasi semantik satu sama lain, maka akan ditemukan *Semantic Associations* dalam kedua entiti tersebut. Dalam penerapannya di aplikasi dunia nyata, algoritma untuk menemukan keberadaan *Semantic Associations* akan sangat bermanfaat dalam memberantas kejahatan seperti pencucian uang, penipuan polis asuransi, dan sebagainya.

Semantic Asscoiations direpresentasikan sebagai sebuah *Resource* dalam berkas RDF. Permasalahan yang timbul dalam proses ini adalah penurunan kinerja yang cukup signifikan saat berkas RDF yang diproses berukuran besar. Salah satu solusi yang diambil ini adalah menyimpan semua struktur graf dalam memori, untuk menghindari penyimpanan dalam diska yang memiliki kecepatan akses jauh lebih lambat dibandingkan dengan memori. Konsekuensi dari pendekatan ini adalah representasi data RDF harus dibuat seringkas mungkin agar masih tersedia tempat untuk menyimpan seluruh struktur graf.

Struktur data yang digunakan dalam BRAHMS adalah *hash table*. *Hash table* ini menyimpan URI dengan *Resource* yang bersesuaian. Struktur data ini juga digunakan dalam indeks yang menyimpan semua informasi *node* yang akan digunakan dalam operasi algoritma dan kueri yang lebih kompleks. Indeks ini digunakan dalam proses pencarian *node neighborhood* yang cepat.

BRAHMS memiliki penyimpanan data internal sebagai berikut:

1. List yang menyimpan daftar triplet yang mengandung representasi numerik dari setiap *Resource*, *properties*, *class*, literal dengan indeks yang memungkinkan akses cepat.
2. List ID dari *Resources, properties, classes*, dan literal dengan URI yang bersesuaian.
3. List dari *Resources values* dan *literal values*.

Untuk menguji kinerja dari skema indeks ini, akan dilakukan perbandingan dengan skema indeks Jena, Sesame, dan Redland. Data RDF yang digunakan adalah data SWETO dan data *Synthetic* (*small* dan *big*). Hasil dari pengujian tersebut memperlihatkan bahwa penggunaan memori dan kinerja BRAHMS lebih baik dibandingkan skema indeks lainnya.

2.5.8 RDF-*Source related Storage System* (RDF-S3)

Karsten Tolle dan Fabian Wleklisnksi dalam [20] memperkenalkan sebuah skema indeks untuk menyimpan triplet RDF yang dinamakan RDF-S3. Kelebihan skema indeks ini dibandingkan dengan skema indeks lain adalah skema indeks ini menyimpan tentang asal informasi dari setiap triplet. Dengan demikian, pengguna dapat melakukan pelacakan kembali ke asal informasi yang memudahkan untuk melakukan verifikasi terhadap kebenaran sebuah informasi yang ada dalam berkas RDF. Hal ini

diimplementasikan dalam RDF-S3 dengan menambahkan *node* konteks. Penambahan ini dilakukan tanpa melakukan perubahan terhadap model data RDF. RDF-S3 terdiri dari dua buah komponen penyusun, komponen *loader*, dan komponen RDF-S3 API.

Dalam komponen *loader*, digunakan *ICS-Validating RDF Parser* (VRP) untuk menyimpan data RDF ke dalam repositori. Penggunaan VRP ini memberikan pengguna kemampuan untuk melakukan validasi pada level semantik sebelum menyimpannya ke basis data. Hal ini dilakukan sebagai salah satu upaya untuk meningkatkan kualitas data yang disimpan. Komponen RDF-S3 API berfungsi untuk merepresentasikan RDF secara murni ataupun dengan representasi RDF yang dilengkapi dengan informasi sumbernya. Keuntungan dari arsitektur ini, RDF-S3 memungkinkan operasi penghapusan dan *update* terhadap sumber informasi. Struktur penyimpanan internal dari RDF-S3 mengkombinasikan dua buah pendekatan (`GenRepr` dan `SpecRepr`) yang digunakan sebagai *redundant storage*. Setiap *Resource* URI akan direpresentasikan sebagai integer untuk mengurangi ukuran penyimpanan, yang disebabkan karena penggunaan *redundant storage*.

Skema indeks ini juga dilengkapi dengan bahasa kueri yang dinamakan *easy* RQL (eRQL), sebuah bahasa kueri yang didesain berdasarkan SPARQL. Pengembangan bahasa kueri ini didasarkan pada kebutuhan untuk mengakses representasi data yang terdapat dalam RDF-S3 dalam waktu yang cepat. eRQL didesain dalam bentuk yang sesederhana mungkin agar dapat digunakan dengan mudah oleh pengguna. Kueri yang diberikan tidak membutuhkan pengetahuan mengenai skema RDF yang tersimpan. Pengguna bahkan dapat memberikan kueri berupa *string* dan tetap mendapatkan jawaban yang sesuai. Sistem akan mengembalikan jawaban dalam bentuk triplet dan sumber informasinya. Sebagai tambahan, eRQL mendukung tiga mode,

yaitu *Statement-Mode*, *POI-Mode*, dan *Document Mode*.

2.5.9 *Lightweight Object Repository* (Lore)

Jason McHugh, et al. dalam [17] memperkenalkan Lore, adalah DBMS yang didesain khusus untuk mengatur *semistructured information*. Lore menggunakan model data yang sangat sederhana, yang disebut dengan *Object Exchange Model* (OEM). Lore memiliki bahasa kueri yang disebut sebagai Lorel (*Lore Language*). Lorel dikembangkan agar pengguna dapat melakukan operasi temu kembali informasi dan memperbaharui data yang tidak memiliki struktur yang tetap.

Sistem Lore memiliki arsitektur yang terdiri dari dua lapisan, yaitu lapisan *Query Compilation* dan lapisan *Data Engine*. Lapisan *Query Compilation* terdiri dari *parser*, *query plan generator*, dan *query optimizer*. Lapisan *Data Engine* tersusun dari *OEM Object Manager*, penyimpanan eksternal, dan lainnya. Sistem Lore berkomunikasi dengan aplikasi melalui *Application Programming Interface* (API).

2.5.10 KAON

Erol Bozsak, et al. dalam [10] memperkenalkan *Karlsruhe Ontology and Semantic Web Tool Suite* (KAON), sebuah *Semantic Web tool* yang dikembangkan dari proyek yang didanai oleh Uni Eropa[15]. KAON didesain untuk menyediakan infrastruktur on-

[15] http://kaon.semanticweb.org

tologi dan metadata. Infrastruktur ini dibutuhkan untuk membangun, menggunakan, dan mengakses aplikasi *Semantic Web* yang terdapat di web ataupun di *desktop*.

KAON memiliki arsitektur yang terdiri dari beberapa lapisan, lapisan *Application and Service*, lapisan *Middleware*, dan lapisan *Data and Remote Service*. Pada lapisan *Application and Service*, klien dapat berbentuk aplikasi java yang dibangun berdasarkan OntoMat dan aplikasi web yang berbasiskan KAON-PORTAL. Lapisan *Middleware* menyediakan abstraksi untuk mengakses ontologi. Pada lapisan *Data and Remote Service* menawarkan akses ke tempat penyimpanan eksternal dan menyediakan fasilitas *reasoning*.

KAON memiliki beberapa *tool* pendukung lain yang mengimplementasikan arsitektur tersebut. *Tool* pendukung tersebut terdiri dari dua jenis, yaitu *frontend* dan *backend*. *Frontend tool* terdiri dari KAON PORTAL, sebuah portal yang berbasiskan ontologi. Dalam portal ini terdapat OntoMat, sebuah kerangka kerja untuk mengembangkan aplikasi berbasiskan ontologi dan metadata. Pada *backend tool*, terdapat KAON API dan KAON *Server*. KAON API merupakan komponen inti pada lapisan *Middleware*. KAON API bertanggung jawab dalam menjaga konsistensi dari ontologi. Sementara KAON *Server* akan menangani proses penyimpanan data RDF yang dapat diakses oleh banyak pengguna dalam waktu yang bersamaan.