Uniwersytet w Białymstoku

Wydział Matematyki i Informatyki

Instytut Informatyki

Mapowanie grafów RDF na grafy właściwości

Łukasz Szeremeta

Promotor:
dr hab. Anna Gomolińska, prof UwB

Promotor pomocniczy:
dr inż. Dominik Tomaszuk

Białystok, 2017

Streszczenie

Obecny internet zdecydowanie nie jest tym, czym był na początku. Z medium do wymiany dokumentów przekształcił się w narzędzie praktycznie do wszystkiego – komunikacji na żywo, zakupów czy pracy zdalnej. Wiele usług internetowych wykorzystuje jeden z modeli przechowywania danych grafowych – Resource Description Framework (RDF) i grafy właściwości (ang. property graphs). Pierwszy z nich wykorzystywany jest m.in. w bazach wiedzy, czyli zbiory zawierające logicznie powiązane informacje z danych dziedzin. Drugi natomiast, m.in w systemach wielu instytucji finansowych do wczesnego wykrywania nadużyć, serwisach społecznościowych oraz transporcie lotniczym. Przy lawinowo rosnącej ilości danych, pogodzenie obu tych „światów", wydaje się być niezwykle istotne. W pracy zostaną omówione zagadnienia związane z obydwoma modelami: serializacje, języki zapytań, przegląd magazynów i baz danych. Zaprezentowane zostaną również metody i algorytmy transformacji oraz kompletne rozwiązanie łączące te dwie koncepcje.

Słowa kluczowe: Semantyczny Internet, RDF, SPARQL, grafy właściwości, bazy danych, serializacja danych, NoSQL

Abstract

The current internet is definitely different from the initial original. From the medium of exchange documents it has transformed into a tool for live communication, shopping, remote work and others. Many web services use graph models – Resource Description Framework (RDF) or property graphs. The first one is used e.g. in knowledge bases – collections, which contain logically related information. The second one is used e.g. in the systems of many financial institutions for early abuse detection, on social networking services, and for civil aviation. With the rapidly growing amount of data, dealing with both of these "worlds", seems to be extremely important. This work will cover issues related to both models such as serializations, query languages, graph stores and databases overview. The methods and transformation algorithms with a complete solution combining these two concepts will be also presented.

Keywords: Semantic Web, RDF, SPARQL, property graph, databases, data serializations, NoSQL

Spis treści

Spis rysunków

Spis tablic

Spis skrótów i akronimów

ACID – Atomicity, Consistency, Isolation, Durability

API – Application Programming Interface

BSBM – Berlin SPARQL Benchmark

CSV – Comma-Separated Values

FOAF – Friend of a Friend

HTTP – Hypertext Transfer Protocol

IRI – Internationalized Resource Identifier

JSON – JavaScript Object Notation

JSON-LD – JavaScript Object Notation for Linked Data

OWL – Web Ontology Language

REST – Representational State Transfer

RDF – Resource Description Framework

RDFS – RDF Schema

SKOS – Simple Knowledge Organization System

SPARQL – SPARQL Protocol and RDF Query Language[1]

SQL – Structured Query Language

Turtle – Terse RDF Triple Language

XML – Extensible Markup Language

YARS – Yet Another RDF Serialization

[1] Akronim rekurencyjny

Wstęp

Internet urósł do niesamowitych rozmiarów. Korzystając z jego dobrodziejstw nie tylko konsumujemy docierające do nas treści, ale również sami je tworzymy. Globalna sieć umożliwia dziś nie tylko przeglądanie statycznych stron internetowych, ale również komunikację audio i wideo w czasie rzeczywistym. Dziś czymś normalnym wydaje się zakup potrzebnych towarów w sklepie internetowym czy praca zdalna w wielu zawodach. Ilość informacji cały czas rośnie, a skuteczna i szybka ich analiza staje się być niezwykle ważna.

Resource Description Framework (RDF) i grafy właściwości (ang. property graphs) to dwa modele przechowywania danych grafowych. Pierwszy z nich używany jest w środowiskach Sementycznego Internetu (ang. Semantic Web), a drugi w wielu znanych grafowych bazach danych. Dzięki koncepcji Semantycznego Internetu komputery zaczynają „rozumieć". Głównym przykładem są bazy wiedzy, czyli zbiory zawierające logicznie powiązane informacje z danych dziedzin. Grafy właściwości używane są natomiast między innymi w systemach wielu instytucji finansowych do wczesnego wykrywania nadużyć, serwisach społecznościowych oraz transporcie lotniczym. Pogodzenie obu tych „światów" w lawinowo rosnącej ilości danych, wydaje się być niezwykle istotne.

Celem niniejszej pracy jest omówienie zagadnień związanych z Semantycznym Internetem oraz grafami właściwości, a także prezentacja rozwiązania łączącego obie te koncepcje.

Praca składa się z dwóch części – teoretycznej oraz praktycznej. W pierwszej omówione zostaną kwestie związane z Semantycznym Internetem, Danymi Połączonymi i grafami właściwości, a także zostanie dokonany przegląd serializacji, języków zapytań, magazynów grafów RDF i grafowych baz danych. W drugiej – zaprezentowane zostaną metody i algorytmy transformacji oraz kompletne rozwiązania łączące obie omówione koncepcje.

Praca zawiera trzy główne rozdziały. W rozdziale 1. przedstawione zostaną podstawowe pojęcia związane z Semantycznym Internetem takie jak trójka oraz graf

RDF. Oprócz tego zaprezentowany będzie również język zapytań RDF – SPARQL oraz przegląd i porównanie magazynów grafów RDF. Rozdział 2. będzie poświęcony zagadnieniom związanym z grafami właściwości. Zostanie tu również dokonany przegląd i porównanie serializacji oraz grafowych baz danych, a także omówiony język zapytań Cypher. Rozdział 3. przybliży z kolei kwestie związane z metodami i algorytmami transformacji, architekturą systemu testowego, danymi testowymi oraz wynikami przeprowadzonych eksperymentów. Praca kończy się wnioskami i podsumowaniem.

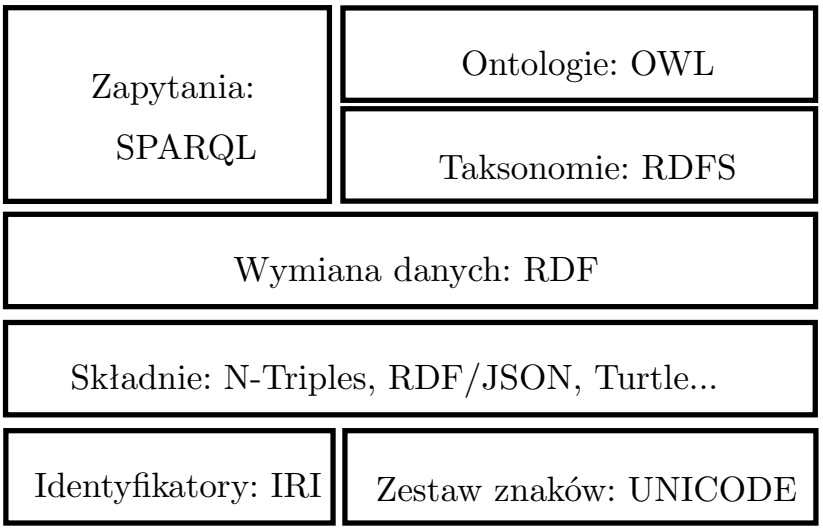
Rozdział 1

Semantyczny Internet

Pierwotnie internet był tworzony do wymiany dokumentów. Nikt prawdopodobnie nie przewidywał wtedy, że on się aż tak rozrośnie. Zwykłe strony internetowe nie są przystosowane, aby zawarte tam informacje mogły być bezproblemowo przetwarzane chociażby przez roboty wyszukiwarek. W niniejszym rozdziale zostaną omówione podstawowe zagadnienia związane z Semantycznym Internetem (ang. Semantic Web) [5] i Danymi Połączonymi (ang. Linked Data) [7]. Oba projekty wydają się być odpowiedzią na coraz większą liczbę bombardujących nas treści, umożliwiając stworzenie „internetu danych"[1].

Definicja 1.1 (Semantyczny Internet)**.** *Projekt stawiający sobie za cel stworzenie ram, umożliwiających opis otaczającego nas świata tak, aby wszelkie dane oraz relacje między nimi, mogły być zrozumiałe nie tylko dla ludzi, ale również maszyn.* □

[1] `https://www.ted.com/talks/tim_berners_lee_on_the_next_web/` [dostęp: 2017-01-15]



Rysunek 1.1: Stos Semantycznego Internetu

Stos Semantycznego Internetu (rysunek 1.1) przedstawia budowę Semantycznego Internetu (definicja 1.1). Ze względu na warstwową budowę, bywa również często nazywany potocznie „torcikiem semantycznym" (ang. Semantic Web Cake).

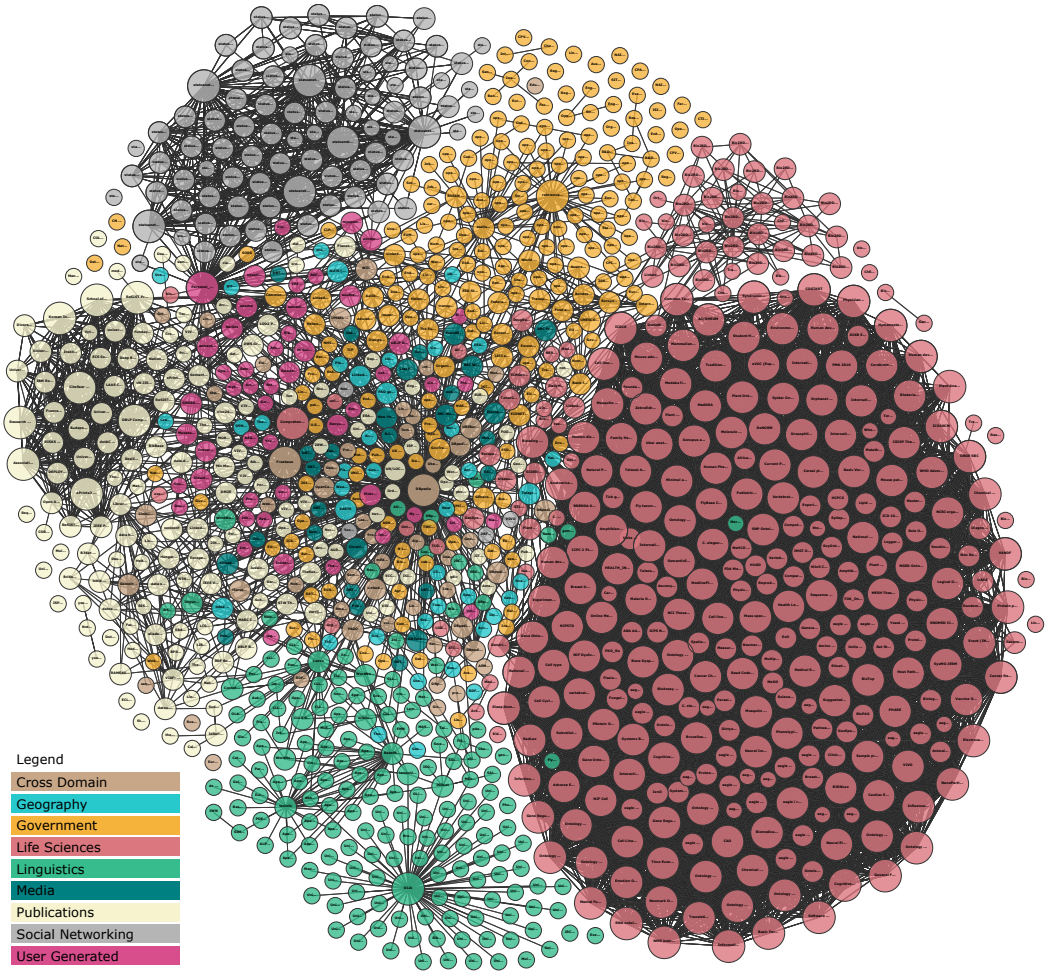
Tak jak wspomniano wcześniej, na stos Semantycznego Internetu [36] składa się wiele różnych warstw, gdzie niższe, są „podparciem" dla elementów znajdujących się powyżej. Innymi słowy, elementy umiejscowione u dołu stosu, są niejako fundamentem dla warstw wyższych.

Dolna warstwa „tortu" składa się z identyfikatorów (IRI) [23] oraz zestawu znaków (UNICODE [1]). Pierwszy z elementów, zostanie dokładniej omówiony w podrozdziale 1.1 niniejszej pracy, natomiast UNICODE można zdefiniować jako zestaw znaków, umożliwiający prezentację i przetwarzanie danych tekstowych zapisanych w wielu różnych językach. Kolejną warstwę zajmują serializacje, czyli sposoby zapisu informacji. Wśród nich można wymienić takie jak: N-Triples [16], Terse RDF Triple Language (Turtle) [48], TriG [51], N-Quads [15], RDF/XML [27], RDF/JSON [58], JavaScript Object Notation for Linked Data (JSON-LD) [39] i inne. Zostaną one omówione dokładniej w podrozdziale 1.2. Nad serializacjami umiejscowiony jest Resource Description Framework (RDF) [19], który zostanie bliżej omówiony w podrozdziale 1.1.

Na samej górze znajduje się SPARQL [31]. Jest to jeden z najpopularniejszych języków zapytań RDF [59]. Cechuje się on składnią bardzo podobną do Structured Query Language (SQL) [38], uzupełnioną o elementy specyficzne dla semantycznej sieci. Język zapytań pozwala na operacje CRUD na danych (dodawanie, usuwanie,

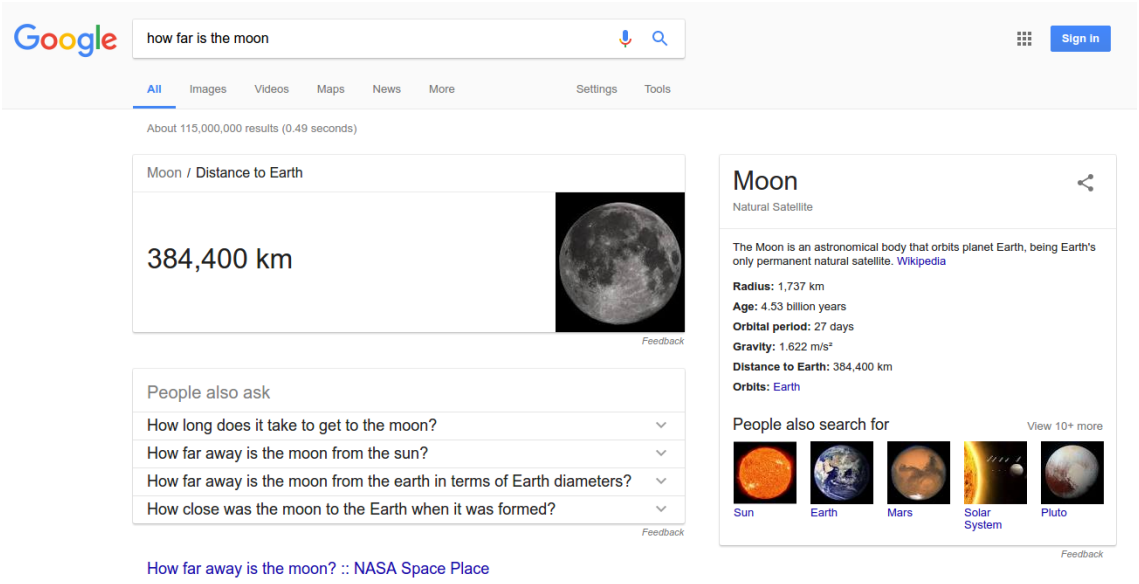
modyfikowanie oraz oczywiście wybieranie). Obok SPARQL, znajdują się taksonomie RDFS (RDF Schema) [12] oraz ontologie Web Ontology Language (OWL) [4]. Pierwszy z wymienionych, rozszerza RDF o pojęcia klas i własności, pozwalając opisywać relacje pomiędzy zgromadzonymi danymi. Pozwala również na hierarchię klas. OWL jest rozszerzeniem RDFS o bardziej zaawansowane konstrukcje takie jak relacja symetryczności czy przechodniości. Standard oprócz wymienionych, obsługuje również liczności.

Definicja 1.2 (Dane Połączone)**.** *Dane podzielone na części, które jednocześnie mogą być ze sobą powiązane pewnymi relacjami. Dane mogą być umieszczone w oddzielnych zbiorach danych, ale w praktyce nadal traktowane jako całość.* □



Rysunek 1.2: Przykład chmury danych połączonych (grafika na podstawie pracy: Andrejs Abele, John P. McCrae, Paul Buitelaar, Anja Jentzsch i Richard Cyganiak, licencja: CC-BY-SA)

Na rysunku 1.2 widoczne są poszczególne kategorie Danych Połączonych takie jak: geograficzne, rządowe, nauki przyrodnicze, lingwistyczne czy generowane przez użytkowników. Koncepcję Danych Połączonych można porównać do tej, znanej z tradycyjnej sieci, gdzie linki prowadzą do dokumentów. Tu jednak odnośniki wskazują na konkretne dane. Dzięki takiemu podejściu, można budować bardzo rozbudowane i konkretne zapytania, również takie, których nikt wcześniej nie opisał w „zwykłym internecie", ale odpowiedź na nie jest możliwa na podstawie samych, powiązanych ze sobą danych. Można również uzyskiwać błyskawiczne odpowiedzi na znacznie mniej skomplikowane pytania takie jak daty urodzin znanych osób oraz inne dane statystyczne. Rozwiązania oparte o tę koncepcję z powodzeniem działają. Przykładami są Google Knowledge Graph [53] oraz spełniające role osobistych asystentów: Aplikacja Google[2] wraz Google Assistant[3]. W przypadku pierwszego z wymienionych rozwiązań, oprócz wyników ze „zwykłych" stron internetowych, prezentowane są informacje dodatkowe, często będące bezpośrednią odpowiedzią na zadane pytanie (rysunek 1.3).



Rysunek 1.3: Przykład działania Google Knowledge Graph

Innym przykładem może być Open Graph [33] firmy Facebook. Pozwala ono webmasterom na dodawanie do swoich stron internetowych dodatkowych metadanych opisujących znajdujące się tam treści.

[2] `https://www.google.com/search/about/` [dostęp: 2017-01-17]
[3] `https://assistant.google.com/` [dostęp: 2017-01-17]

1.1 Trójka i graf RDF

W tym podrozdziale zostaną omówione zagadnienia związanie z trójką i grafem RDF.

Trójka RDF może być zbudowana z następujących elementów:

- referencji Internationalized Resource Identifier (IRI),
- literałów,
- węzłów pustych.

Referencje IRI są identyfikatorami określającymi konkretny zasób.

Przykład 1.1. *Przykład pokazuje referencję IRI wskazującą na opis Ziemi w DBpedii [3].*

```
<http://dbpedia.org/resource/Earth>
```

Literały to wartości leksykalne reprezentujące jakąś wartość, będące ciągami znaków wraz z przypisanym im typem danych identyfikowanym przez referencję IRI.

Przykład 1.2. *Przykład pokazuje literał z typem* `float`*.*

```
"121.57"^^http://www.w3.org/2001/XMLSchema#float
```

Wyróżnia się także węzły puste. Reprezentują one zasób, dla którego nie została określona referencja IRI ani literał.

Przykład 1.3. *Przykład pokazuje identyfikator węzła pustego.*

```
_:something
```

Język RDF umożliwia opisanie zasobów w postaci zdań składających się z *tematu*, *predykatu* i *obiektu*. Zdania nazwane są tu trójkami RDF. Temat jest określeniem opisywanego zasobu. Predykat określa specyficzne właściwości zasobu, a

także wyraża relację między tematem i obiektem. Wspomniany obiekt przechowuje wartość relacji. Trójkę RDF można więc zdefiniować następująco:

Definicja 1.3 (Trójka RDF)**.** *Załóżmy, że $\mathcal{R}$ to zbiór referencji IRI, $\mathcal{P}$ to zbiór węzłów pustych. $\mathcal{L}$ jest zbiorem literałów. Wtedy* trójka RDF *u będzie zdefiniowana jako trójka $u = \langle t, p, o \rangle$, gdzie $t \in \mathcal{R} \cup \mathcal{P}$ nazywa się tematem, $p \in \mathcal{R}$ nazywa się predykatem, a $o \in \mathcal{R} \cup \mathcal{P} \cup \mathcal{L}$ nazywa się obiektem.* □

Przykład 1.4. *Przykład pokazuje trójkę RDF złożoną z tematu, predykatu i obiektu w serializacji Turtle [48].*

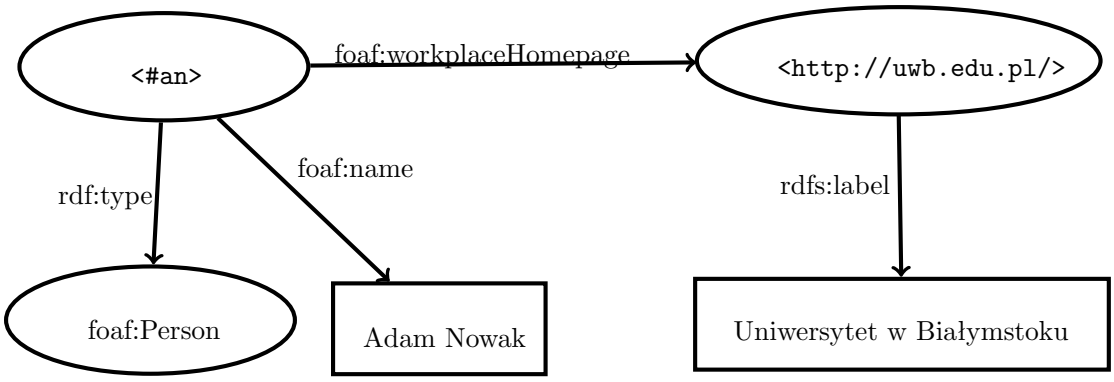
```
1  @prefix foaf: <http://xmlns.com/foaf/0.1/> .
2
3  <http://example.net/me#an> foaf:name "Adam Nowak" .
```

Graf RDF stanowi kolekcję trójek RDF. Można go określić jako multigraf skierowany z etykietami (patrz rozdział 2). W tym przypadku tematy i obiekty trójki to węzły. Graf RDF określa się również jako graf danych strukturalnych, gdzie trójka $\langle t, p, o \rangle$ widziana jest jako krawędź $t \xrightarrow{p} o$.

Definicja 1.4 (Graf RDF)**.** *Załóżmy, że $\mathcal{R}$ to zbiór referencji IRI, $\mathcal{P}$ to zbiór węzłów pustych, $\mathcal{L}$ jest zbiorem literałów, a $\mathcal{O} = \mathcal{R} \cup \mathcal{P} \cup \mathcal{L}$ i $\mathcal{S} = \mathcal{R} \cup \mathcal{P}$. Wtedy $G \subset \mathcal{S} \times \mathcal{R} \times \mathcal{O}$ to zbiór wszystkich* trójek RDF *zwany* grafem RDF. □

Przykład 1.5. *Przykład pokazuje graf RDF wykorzystujący profil FOAF (ang. Friend of a Friend) [13]. Poniższy graf zawiera cztery trójki RDF w serializacji Turtle. Graf został zaprezentowany na rysunku 1.4.*

```
1        @prefix foaf: <http://xmlns.com/foaf/0.1/> .
2
3  <#an> rdf:type foaf:Person .
4  <#an> foaf:name "Adam Nowak" .
5  <#an> foaf:workplaceHomepage <http://uwb.edu.pl/> .
6  <http://uwb.edu.pl/> rdfs:label "Uniwersytet w Białymstoku" .
```



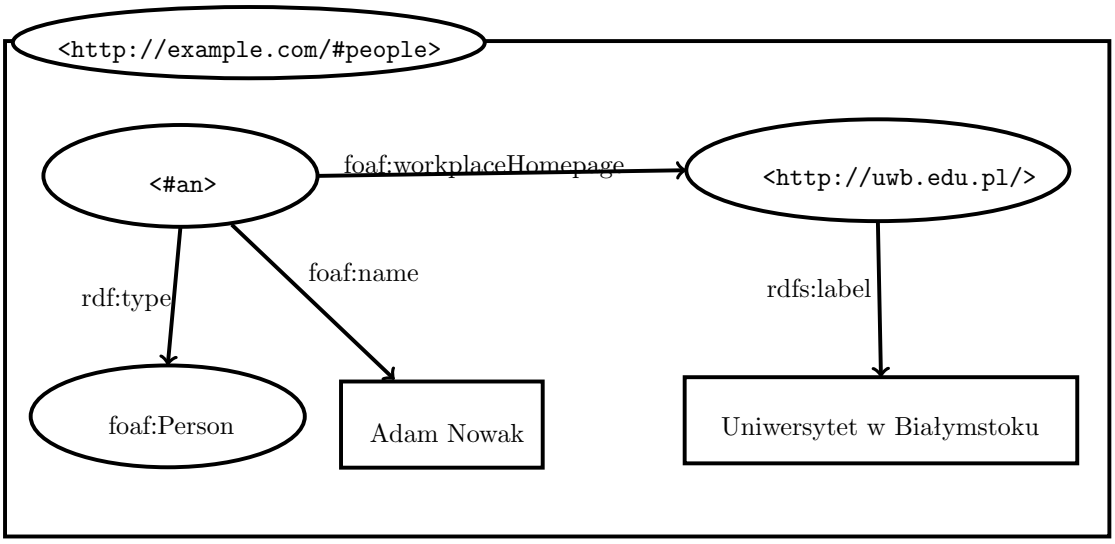
Rysunek 1.4: Graf RDF

RDF można rozszerzyć o koncepcję grafów nazwanych, dzięki czemu graf może przyjąć swoją własną nazwę.

Definicja 1.5 (Graf nazwany)**.** Graf nazwany *to para* $\langle u, G \rangle$*, gdzie* $u \in \mathcal{R} \cup \mathcal{P}$ *to nazwa grafu a* G *to graf RDF.* □

Przykład 1.6. *Przykład pokazuje graf nazwany wykorzystujący profil FOAF. Graf ten posiada własną nazwę* `http://example.com/#people` *i zawiera cztery trójki RDF zapisane w serializacji TriG [51]. Graf został zaprezentowany na rysunku 1.5.*

```
1  @base <http://example.net/> .
2  @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
3  @prefix foaf: <http://xmlns.com/foaf/0.1/> .
4
5  <http://example.com/#people> {
6    <#an> rdf:type foaf:Person .
7    <#an> foaf:name "Adam Nowak" .
8    <#an> foaf:workplaceHomepage <http://uwb.edu.pl/> .
9    <http://uwb.edu.pl/> rdfs:label "Uniwersytet w Białymstoku" .
10 }
```



Rysunek 1.5: Graf nazwany

1.2 Przegląd serializacji RDF

W tym podrozdziale zostaną omówione najpopularniejsze serializacje RDF. Serializacja to sposób opisu danych oraz łączących je relacji w plikach tekstowych. Każda z serializacji posiada swoje specyficzne cechy dzięki którym może być lepsza lub gorsza od innych.

Jedną z najprostszych serializacji RDF [50] jest *N-Triples*. Wiersze opisujące trójkę RDF składają się z tematu, predykatu i obiektu oddzielonych białym znakiem. Zdanie, a zarazem linia, kończy się znakiem kropki. Komentarze zaczynają się od znaku `#` i mogą występować zarówno po zakończeniu poszczególnych zdań jak i w nowych liniach. Referencje IRI otaczane są z lewej znakiem `<`, a z prawej `>`, natomiast literały z obu stron znakami `"`. Czyste literały mogą mieć ustalony język w jakim są zapisane. Odpowiedni tag języka jest poprzedzany znakiem `@`. Do literałów z typem dodawany jest poprzedzony znakami `^^` IRI, będący opisem danego typu danych. Identyfikatory węzłów pustych oznaczane są jako `_:` wraz z etykietą, będącą ciągiem znaków. W formie kanonicznej N-Triples komentarze nie są dozwolone, a elementy zdania muszą być oddzielane pojedynczą spacją.

Przykład 1.7. *Przykład pokazuje trójki RDF zapisane w serializacji N-Triples:*

```
1 <http://example.net/me#an> <http://xmlns.com/foaf/0.1/name> "Adam Nowak" .
2 <http://example.net/me#an> <http://xmlns.com/foaf/0.1/gender> "male"@en .
```

Serializacja *N-Quads* jest tożsama z serializacją N-Triples z tym, że oprócz tematu, predykatu i obiektu, każda linia zawiera również etykietę grafu RDF. Zdanie tak samo jak w N-Triples kończy się kropką.

Przykład 1.8. *Przykład pokazuje trójki RDF z nazwą grafu zapisane w serializacji N-Quads:*

```
1 <http://example.net/#an> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <
    http://xmlns.com/foaf/0.1/Person> <http://example.com/adam> .
2 <http://example.net/#an> <http://xmlns.com/foaf/0.1/name> "Adam Nowak" <http://
    example.com/adam> .
3 <http://example.net/#an> <http://xmlns.com/foaf/0.1/workplaceHomepage> <http://
    uwb.edu.pl/> <http://example.com/adam> .
```

Turtle jest nadzbiorem serializacji N-Triples (przykład 1.7). Wprowadza większe możliwości skracania składni. Dzięki temu zapis może być bardziej kompaktowy. Pierwszymi ze zmian są bazowe i relatywne IRI. Bazowy IRI, określany za pomocą dyrektywy `@base`, może być automatycznie doklejany do relatywnych IRI. Na podobnej zasadzie działają również prefiksy definiowane za pomocą dyrektywy `@prefix`. Do odpowiedniego prefiksu odwołuje się za pomocą określonej wcześniej w definicji prefiksu etykiety. Turtle wprowadza również `;` służące do oddzielania wielu predykatów oraz `,` – do rozdzielania wielu obiektów przypisanych do danego tematu. W przypadku predykatów można używać `a` jako krótszej formy dla pełnego IRI `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`.

Przykład 1.9. *Przykład pokazuje trójki z przykładu 1.7 wraz z dodatkowym tagiem języka zapisane w serializacji Turtle:*

```
@base <http://example.net/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<me#an>
  foaf:name "Adam Nowak" ;
  foaf:gender "male"@en, "mężczyzna"@pl .
```

Serializacja *TriG* rozszerza Turtle o obsługę grafów nazwanych (definicja 1.5).

Przykład 1.10. *Przykład pokazuje graf nazwany* `<http://example.com/adam>` *wraz z przykładowymi trójkami RDF z przykładu 1.8 w serializacji TriG:*

```
@base <http://example.net/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

<http://example.com/adam> {
  <#an> rdf:type foaf:Person .
  <#an> foaf:name "Adam Nowak" .
  <#an> foaf:workplaceHomepage <http://uwb.edu.pl/> .
}
```

RDF/XML to serializacja oparta o Extensible Markup Language (XML) [11]. Oznacza to, że dziedziczy również pewne cechy formatu na którym jest oparta. W przypadku XML jest to duża czytelność dla człowieka, a także bogata ilość gotowych bibliotek programistycznych do jego obsługi.

Przykład 1.11. *Przykład pokazuje graf nazwany* `<http://example.com/adam>` *wraz z przykładowymi trójkami RDF z przykładu 1.8 w serializacji RDF/XML.*

```
1  <rdf:RDF
2    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3    xmlns:foaf="http://xmlns.com/foaf/0.1/">
4
5  <rdf:Description rdf:about="http://example.net/#an">
6    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
7    <foaf:name>Adam Nowak</foaf:name>
8    <foaf:workplaceHomepage rdf:resource="http://uwb.edu.pl/"/>
9  </rdf:Description>
10
11 </rdf:RDF>
```

Głównym elementem dokumentu XML widocznego na przykładzie 1.11 jest `rdf:RDF`. Wewnątrz można określić prefiksy będące przestrzenią nazw XML. Zdefiniowane prefiksy widoczne są w liniach 2. i 3. przykładu. Element `rdf:Description` reprezentuje wierzchołek, a `rdf:type`, `foaf:name` i `foaf:workplaceHomepage` to właściwości.

Serializacja *RDF/JSON* również posiada wsparcie dla grafów nazwanych. Jest ona zgodna z formatem JavaScript Object Notation (JSON) [10], którego cechą szczególną jest duża czytelność, połączona z bardziej zwięzłą, niż w przypadku XML, składnią. Ze względu na stosunkowo nieduże rozmiary przesyłanych danych, JSON jest z powodzeniem używany w różnych usługach sieciowych.
Definicje zawierające opisy tematu, predykatu i obiektu są od siebie wyraźnie oddzielone. Opisy mogą zawierać różne klucze. `type` określa typ wartości, a `value` samą wartość. Oba wymienione klucze są obowiązkowe. Serializacja obsługuje takie typy danych jak IRI (`uri`), literały (`literal`), literały z typem (`typed-literal`), a także węzły puste (`bnode`). Dodatkowo używa się kluczy takich jak `lang` oraz `datatype`. Pierwszy z nich określa język i może występować tylko, gdy mamy do czynienia z literałem. Drugi w kolejności określa typ danych dla literału z typem. Dzięki polu `context` można wzbogacić trójkę RDF (definicja 1.3) o kontekst.
Referencje IRI można używać zarówno w przypadku tematu, predykatu, obiektu jak i specjalnego pola odpowiedzialnego za kontekst. Węzeł pusty może pojawić się w temacie i obiekcie. Literał oraz literał z typem może być natomiast użyty tylko w obiekcie.

Przykład 1.12. *Przykład pokazuje wybraną trójkę z przykładu 1.7 zapisaną w serializacji RDF/JSON*[4].

```
{[{
    "subject": {
      "type" : "uri",
      "value" : "http://example.net/me#an"
    },
    "predicate": {
      "type" : "uri",
      "value" : "http://xmlns.com/foaf/0.1/name"
    },
    "object": {
      "type" : "literal",
      "value" : "Adam Nowak"
    }
  }]}
```

Inną serializacją opartą o format JSON jest *JSON-LD*[5]. Pozwala na całkiem proste wzbogacenie „zwykłych" dokumentów JSON o elementy związane z danymi połączonymi (definicja 1.2).
Tag `@context` opisuje kontekst innych zawartych kluczy. Dzięki temu tagi nie muszą być za każdym razem jednoznaczne – można posługiwać się skróconą nazwą, której kontekst jest wytłumaczony w innym dokumencie. Takie podejście mocno zwiększa czytelność, a także sprawia, że dokument JSON może mieć mniejszą objętość przy zachowaniu takiej samej liczby danych. Upraszcza to również adaptację istniejących systemów opartych o JSON.
Innym kluczem jest `@id`, który pozwala na jednoznaczne wskazanie na opisywane dane w innym miejscu. Jest to przydatne, jeżeli na przykład chcemy wspomnieć o konkretnej osobie. Innymi przydatnymi kluczami są `@type`, `@language` oraz `@value`. Pierwszy z nich wskazuje jednoznacznie typ, drugi określa język, a trzeci definiuje samo wyrażenie. `@graph` używany jest do opisu grafów nazwanych.

[4] `uri` traktowany jako IRI
[5] `http://json-ld.org/` [dostęp: 2017-01-30]

Przykład 1.13. *Przykład pokazuje trójki z przykładu 1.10 zapisane w serializacji JSON-LD:*

```
{
  "@context": {
    "Person": "http://xmlns.com/foaf/0.1/Person",
    "name": "http://xmlns.com/foaf/0.1/name",
    "homepage": "http://xmlns.com/foaf/0.1/workplaceHomepage"
  },
  "@id": "http://example.com/adam",
  "@type": "Person",
  "name": "Adam Nowak",
  "homepage": "http://uwb.edu.pl/"
}
```

JSON-LD jest używany między innymi w Application Programming Interface (API) wspomnianego wcześniej Google Knowledge Graph[6].

Nazwa	**Prefiksy**	**Rodzina Turtle**	**Oparta o XML**	**Oparta o JSON**	**Normalizacja**
N-Triples	nie obsługuje	tak	nie	nie	tak
N-Quads	nie obsługuje	tak	nie	nie	nie
Turtle	obsługuje	tak	nie	nie	nie
TriG	obsługuje	tak	nie	nie	nie
RDF/XML	obsługuje	nie	tak	nie	nie
RDF/JSON	nie obsługuje	nie	nie	tak	tak
JSON-LD	obsługuje	nie	nie	tak	częściowo

Tablica 1.1: Porównanie serializacji RDF

1.3 Język zapytań SPARQL

W tym podrozdziale zostanie omówiony SPARQL będący językiem zapytań i protokołem używanym w Semantycznym Internecie. Na początku zdefiniowane zostaną

[6] `https://developers.google.com/knowledge-graph/` [dostęp: 2017-01-30]

istotne pojęcia, a następnie omówione składnie wybierania jak i modyfikowania danych.

Definicja 1.6 (Zmienna SPARQL)**.** *Dopasowuje dowolny element (zasób lub literał) w zbiorze danych RDF. Zaczyna się od znaku zapytania.* □

Definicja 1.7 (Wzorzec trójki)**.** *Załóżmy, że $\mathcal{R}$ to zbiór referencji IRI, $\mathcal{P}$ to zbiór węzłów pustych. $\mathcal{L}$ jest zbiorem literałów, a $\mathcal{V}$ to zmienna SPARQL. Wtedy* wzorzec trójki *u będzie zdefiniowany jako trójka $wt = \langle t_w, p_w, o_w \rangle$, gdzie $t_w \in \mathcal{R} \cup \mathcal{P} \cup \mathcal{V}$ nazywa się tematem wzorca trójki, $p_w \in \mathcal{R} \cup \mathcal{V}$ nazywa się predykatem wzorca trójki, a $o_w \in \mathcal{R} \cup \mathcal{P} \cup \mathcal{L} \cup \mathcal{V}$ nazywa się obiektem wzorca trójki.* □

Inaczej mówiąc, wzorzec trójki to trójka RDF (definicja 1.3), której każda ze składowych (temat, predykat, obiekt) może być zastąpiona zmienną.

Przykład 1.14. *Przykład pokazuje wzorce trójki. W linii nr 1 umieszczono wzorzec dopasowujący trójkę z określonym predykatem i obiektem. Wzorzec zaprezentowany w linii nr 2 dopasowuje dowolną trójkę. W linii nr 3 widoczny jest wzorzec dopasowujący jawnie określoną trójkę RDF.*

```
1  ?person foaf:name "Adam Nowak" .
2  ?subject ?predicate ?object .
3  <http://example.net/me#an> foaf:name "Adam Nowak" .
```

Definicja 1.8 (Podstawowy wzorzec grafu)**.** *Podstawowym wzorcem grafu (ang. Basic Graph Pattern) jest zbiór wzorców trójek.* □

Wybieranie danych

Składnia wybierania danych jest podobna do tej znanej z SQL. Została ona jednak dostosowana do specyfiki Semantycznego Internetu. Początek zapytania to opcjonalna deklaracja prefiksów. Zastosowanie ich jest dokładnie takie samo jak w przypadku serializacji (podrozdział 1.2) – prefiksy pozwalają na skrócenie zapisu. Obowiązkowym elementem zapytania jest klauzula wynikowa określająca używaną formę zapytania (`SELECT`, `CONSTRUCT`, `ASK` lub `DESCRIBE`) i zwracane dane. Poniżej opcjonalnie można wybrać zbiór danych na którym będzie wykonywane zapytanie. Służą do tego klauzule `FROM` oraz `FROM NAMED`, gdzie drugą z nich stosuje się do grafów

nazwanych. Następnie znajduje się sekcja `WHERE`, gdzie określa się szczegółowe warunki zapytania. Ostatnie miejsca w składni zapytania zajmują modyfikatory takie jak `GROUP BY`, `HAVING`, `ORDER BY`, `LIMIT`, `OFFSET`, `BIND` i `VALUES`. Dwóch pierwszych używa się przy agregacji danych, trzeci służy do sortowania wyników, dwa kolejne do ograniczania wyników zapytania, `BIND` pozwala na przypisanie wartości do zmiennej, natomiast ostatni, `VALUES`, pozwala na przypisanie wielu zmiennych w jednym bloku. Użycie modyfikatorów jest opcjonalne.
Dostępne są następujące składnie wybierania danych:

- `SELECT` – wybiera i zwraca surowe dane,
- `CONSTRUCT` – konstruuje graf RDF,
- `ASK` – zwraca prawdę, gdy został znaleziony przynajmniej jeden pasujący wynik lub fałsz w przeciwnym przypadku,
- `DESCRIBE` – zwraca opis zasobu.

Poniżej przedstawiono przykłady wymienionych form zapytań.

Przykład 1.15. *Przykład pokazuje zapytanie* ***SELECT****. Zapytanie zwróci osoby, których stroną domową miejsca pracy jest* ***http://uwb.edu.pl/****.*

```
1  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2  SELECT ?person
3  WHERE {
4    ?person foaf:workplaceHomepage <http://uwb.edu.pl/> .
5  }
```

Bardzo podobną do `SELECT` formą zapytania jest `CONSTRUCT`. Zamiast zwracania jedynie pasujących zmiennych, `CONSTRUCT` zwraca graf RDF.

Przykład 1.16. *Przykład pokazuje zapytanie* `CONSTRUCT`*. Zapytanie zwróci graf RDF złożony z osób, których stroną domową miejsca pracy jest* `http://uwb.edu.pl/`*.*

```
1 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2 CONSTRUCT {
3   ?person foaf:workplaceHomepage <http://uwb.edu.pl/> .
4 }
```

Inną przydatną konstrukcją jest składnia `ASK`. Zapytanie zwraca prawdę, gdy znaleziono przynajmniej jeden wynik spełniający kryteria oraz fałsz w przeciwnym przypadku. Zapytanie `DESCRIBE` zwraca informacje o danym zasobie.

Aktualizacja danych

Wybieranie to nie jedyne możliwości języka zapytań SPARQL. Istotne są także operacje dodawania, zmiany oraz usuwania danych.

Definicja 1.9 (Zbiór danych RDF)**.** *Załóżmy, że* $G, G_1, G_2, \ldots, G_n$ *to grafy RDF, a* $r_1, r_2, \ldots, r_n$ *to różne od siebie referencje IRI. Wtedy zbiór danych RDF to zbiór* $\{G, (r_1, G_1), (r_2, G_2), \ldots, (r_n, G_n)\}$*, w którym* G *jest nazywany grafem domyślnym, a* (r_i, G_i) *grafem nazwanym.* □

Oprócz wybierania danych, możliwe są różnego rodzaju operacje na zbiorach danych RDF. Język SPARQL [28] udostępnia następujące operacje aktualizacji danych:

- `INSERT DATA` – dodaje konkretne trójki do grafu,
- `DELETE DATA` – usuwa konkretne trójki z grafu,
- `INSERT` – dodaje trójki do grafu (bazując na warunkach)
- `DELETE` – usuwa trójki z grafu (bazując na warunkach)
- `LOAD` – odczytuje dokument z podanego IRI i umieszcza wszystkie trójki we wskazanym grafie
- `CLEAR` – usuwa wszystkie trójki we wskazanym grafie

Aby dodać konkretne trójki do grafu, należy użyć klauzuli `INSERT DATA`. W dalszej części zapytania określa się trójki w serializacji Turtle (przykład 1.9). Na samym początku można podać graf nazwany do którego trójki zostaną dodane. W przypadku niewyspecyfikowania konkretnego grafu, trójki zostaną dopisane do grafu domyślnego. Identyczną do `INSERT DATA` składnię ma `DELETE DATA`. Różnica polega wyłącznie na innej klauzuli oraz odmiennych efektach wykonania zapytania. W przypadku wykonania tej drugiej, wybrane trójki zostaną usunięte.
Składnie `INSERT` i `DELETE` różnią się od `INSERT DATA` oraz `DELETE DATA` możliwością określania warunków umieszczenia lub usunięcia danych. Wymagania określane są za pomocą klauzuli `WHERE`. Używając w jednym zapytaniu, jednocześnie `DELETE` i `INSERT` ze wspólną klauzulą `WHERE`, uzyskuje się efekt aktualizacji danych. `WITH` określa graf na którym zostaną wykonane operacje. Jest to klauzula opcjonalna.

Przykład 1.17. *Przykład pokazuje połączone zapytania `INSERT` i `DELETE`. W wyniku wykonania zapytania, nastąpi zmiana imion i nazwisk wszystkich Adamów Nowaków z grafu określonego przez klauzulę `WITH`.*

```
1  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2
3  WITH <http://example/g/people>
4  DELETE { ?person foaf:name 'Adam Nowak' }
5  INSERT { ?person foaf:name 'Adam Jan Nowak' }
6  WHERE
7  {
8    ?person foaf:name 'Adam Nowak'
9  }
```

Składnie `LOAD` oraz `CLEAR` służą do wykonywania masowych zmian w konkretnych grafach RDF. Pierwsza z nich pozwala na załadowanie i dodanie wszystkich trójek z podanego adresu IRI (przykład 1.1) do konkretnego grafu (podanego za pomocą klauzuli `INTO GRAPH` lub domyślnego).

Przykład 1.18. *Przykład pokazuje zapytanie* ***LOAD****. W wyniku wykonania zapytania, do grafu* ***http://example/g/people****, zostaną dodane wszystkie trójki z* ***http://example.com/jkowalski****.*

```
LOAD <http://example.com/jkowalski> INTO GRAPH <http://example/g/people>
```

Zapytanie z `CLEAR` usuwa wszystkie trójki we wskazanym przez IRI grafie (słowo kluczowe `GRAPH`), grafie domyślnym (`DEFAULT`; zachowanie może być różne w zależności od implementacji) lub wszystkich grafach nazwanych (`NAMED`). Specyfikator `ALL` usuwa wszystkie trójki we wszystkich grafach.

Przykład 1.19. *Przykład pokazuje zapytanie* ***CLEAR****. W wyniku wykonania zapytania, zostaną usunięte wszystkie trójki z grafu* ***http://example/g/people****.*

```
CLEAR GRAPH <http://example/g/people>
```

1.4 Przegląd magazynów grafów RDF

W niniejszym podrozdziale zostaną omówione magazyny grafów RDF. Przedstawione zostaną rozwiązania w pełni komercyjne (z zamkniętym kodem źródłowym i płatnymi edycjami) oraz rozwiązania bezpłatne i otwartoźródłowe.

RDF4J[7], znany również jako Sesame [14], jest otwartoźródłowym frameworkiem RDF opartym o język Java. Umożliwia nie tylko przechowywanie, ale także wiele innych operacji, takich jak parsowanie czy przeszukiwanie danych. Możliwe jest jego zintegrowanie z innymi magazynami grafów RDF takimi jak: Ontotext GraphDB[8], Stardog[9] czy CumulusRDF [40]. Framework w pełni wspiera język zapytań SPARQL 1.1 omówiony w podrozdziale 1.3. Pozwala nie tylko na wybieranie danych, ale również ich aktualizację. RDF4J posiada wsparcie dla wszystkich najpopularniejszych serializacji RDF.

Apache Jena[44] to otwartoźródłowy framework stworzony specjalnie dla rozwiązań związanych z Semantycznym Internetem i Danymi Połączonymi. W przeciwień-

[7] `http://rdf4j.org/about/` [dostęp: 2017-03-25]
[8] `https://ontotext.com/products/graphdb/` [dostęp: 2017-03-25]
[9] `http://www.stardog.com/` [dostęp: 2017-03-25]

stwie do RDF4J, posiada wbudowane wsparcie dla OWL. Cechą charakterystyczną jest podział na komponenty, odpowiadające za poszczególne funkcjonalności. Framework obsługuje SPARQL 1.1 oraz zapewnia ulepszone przeszukiwanie tekstu za pomocą biblioteki Apache Lucene[10].

H_2RDF [46] to otwartoźródłowy, w pełni rozproszony magazyn grafów RDF. Powstał w opaciu o bazę danych Apache HBase [29, 62]. Zaproponowane rozwiązanie łączy w sobie NoSQLową bazę danych z techniką MapReduce [20]. Jako język zapytań używany jest SPARQL. System wprowadza autorskie rozwiązania w obsłudze złączeń. H_2RDF+ [47] to ulepszona wersja H_2RDF.

AllegroGraph[11] to rozbudowany, własnościowy magazyn grafów RDF. Rozwiązanie zapewnia pełną zgodność z ACID przez co obsługuje takie działania jak *Commit*, *Rollback* czy *Checkpointing*. Językiem zapytań jest SPARQL 1.1. Wszystkie zmieniane trójki są automatycznie indeksowane. Dostępne są również zaawansowane możliwości indeksacji tekstu.

Virtuoso [24] to hybrydowe rozwiązanie łączące w sobie cechy relacyjnych oraz kolumnowych baz danych. Rozwiązanie umożliwia przechowywania jednocześnie nie tylko trójek RDF, ale także standardowych tabel SQL. Możliwy jest import trójek RDF w serializacjach N3, Turtle oraz RDF/XML. Virtuoso wspiera SPARQL jako język zapytań. Klauzule SPARQL można używać wewnątrz standardowych klauzul SQL. Dodatkowo, jeżeli jest to tylko możliwe, rozwiązanie stara się mapować typy danych RDF na odpowiedniki znane z SQL.

Oracle Spatial and Graph[12] jest płatnym dodatkiem do własnościowej bazy danych Oracle Database. Jako język zapytań wykorzystywany jest SPARQL 1.1, ale także SQL. Obok OWL 2 wspierane jest także Simple Knowledge Organization System (SKOS) i reguły definiowane przez użytkownika. System wspiera również indeksację tekstu. Oracle Spatial and Graph można zintegrować z innymi omawianymi wcześniej rozwiązaniami – Apache Jena oraz Sesame.

RTriples [55, 54] to relatywnie prosty magazyn grafów RDF oparty o NoSQL-ową, dokumentową bazę danych RethinkDB[13] oraz micro-framework Sinatra[14]. Magazyn nie posiada obsługi SPARQL, jednak zapewnia alternatywny język zapytań, pozwalający na całkiem zaawansowane wybieranie danych ze zbioru. Wspieranych jest

[10] `https://lucene.apache.org/core/` [dostęp: 2017-03-25]

[11] `http://allegrograph.com/allegrograph/` [dostęp: 2017-03-25]

[12] `http://www.oracle.com/technetwork/database-options/spatialandgraph/overview/spatialandgraph-1707409.html` [dostęp: 2017-03-27]

[13] `https://rethinkdb.com/` [dostęp: 2017-03-25]

[14] `http://www.sinatrarb.com/` [dostęp: 2017-03-25]

5 formatów danych wyjściowych – RDF/JSON [57], N-Triples, Comma-Separated Values (CSV) [52], prosty format tekstowy oraz tabela HTML.
W tabeli 1.2 zostały porównane wszystkie powyższe magazyny grafów RDF:

Nazwa	**Licencja**	**Magazyn danych**	**Język programowania**	**Serializacje**	**Języki zapytań**
RDF4J	Eclipse Distribution License (EDL) 1.0	zewnętrzny	Java	RDF/XML, Turtle, N-Triples, N-Quads, JSON-LD, TriG, TriX, RDF Binary	SPARQL 1.1
Apache Jena	Apache License 2.0	zewnętrzny	Java	Turtle, RDF/XML, N-Triples, JSON-LD, RDF/JSON, TriG, N-Quads, TriX, RDF Binary	SPARQL 1.1
H_2RDF, H_2RDF+	GNU GPL 3	HBase	Java	brak danych	SPARQL
AllegroGraph	własnościowa	natywny	Common Lisp	N-Triples, Turtle, RDF manifest, RDF/XML,N-Quad, TriX, TriG, NQX	SPARQL 1.1 i własny język zapytań
Virtuoso	GNU GPL 2 i własnościowa	hybrydowy (relacyjny i kolumnowy)	C	Turtle, JSON, XML, N-Triples, TriG, CSV	SPARQL 1.1, SQL, XQuery
Oracle Spatial and Graph	własnościowa	grafowy	Java	RDF/XML, N-Triples, N-Quads, TriG, Turtle	SPARQL 1.1, SQL
RTriples	MIT	dokumentowy	Ruby	RDF/JSON, N-Triples, CSV, txt, HTML	własny język zapytań

Tablica 1.2: Porównanie magazynów grafów RDF

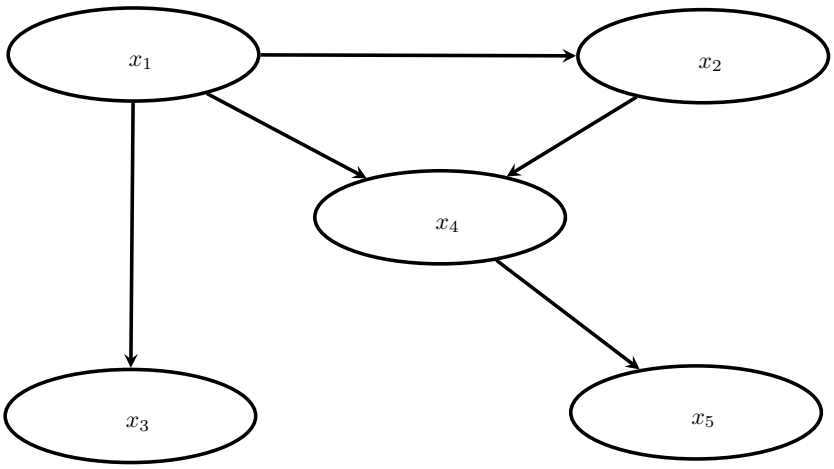
Rozdział 2

Grafy właściwości

W niniejszym rozdziale zostaną przedstawione podstawowe pojęcia związane z grafami. Omówione zostaną grafy właściwości, a także serializacje i grafowy język zapytań Cypher.

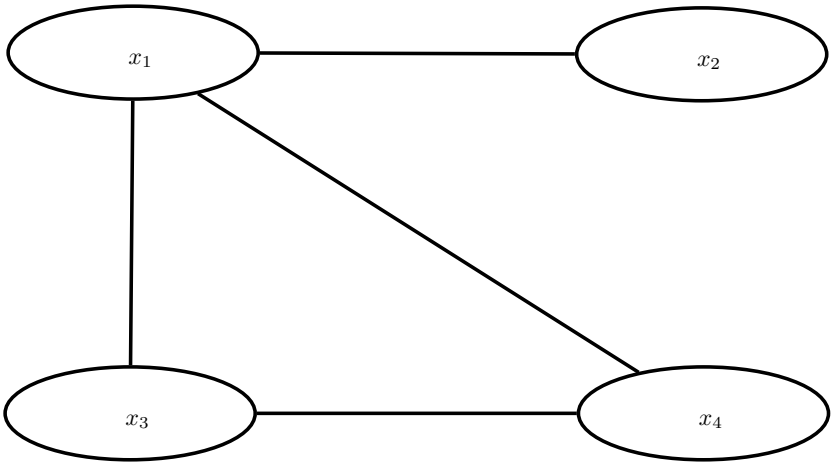
Definicja 2.1 (Graf)**.** *Graf to struktura matematyczna będąca parą zbiorów wierzchołków V i krawędzi K łączących dwa wierzchołki.* □

Definicja 2.2 (Graf skierowany)**.** *Graf skierowany G_s to graf będący parą $G_s = (V, K_s)$, gdzie V to uporządkowany zbiór par wierzchołków, a K_s to zbiór skierowanych krawędzi $K_s = (v_p, v_k)$ z wierzchołka początkowego $v_p \in V$ do końcowego $v_k \in V$.* □



Rysunek 2.1: Graf skierowany z pięcioma wierzchołkami i krawędziami

Definicja 2.3 (Graf nieskierowany)**.** *Załóżmy, że V to zbiór wierzchołków, a K to zbiór krawędzi. Wtedy graf nieskierowany to para $G_n = (V, K)$, gdzie $K \subseteq \{\{v_p, v_k\} : v_p, v_k \in V\}$.* □

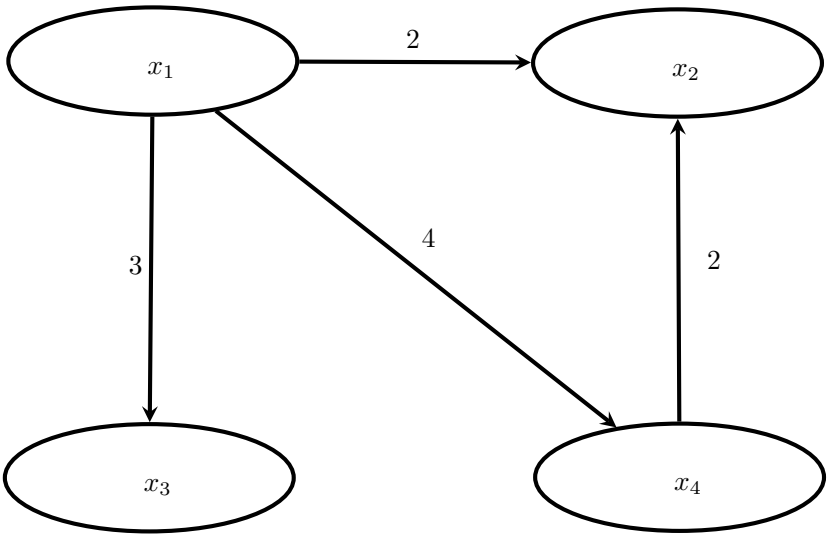


Rysunek 2.2: Graf nieskierowany z czterema wierzchołkami i krawędziami

Grafy pozwalają na prostsze przedstawienie różnych typów danych w różnych dziedzinach. Graf skierowany to graf, którego krawędzie mają groty skierowane w kierunku któregoś z pozostałych wierzchołków grafu lub siebie samych. Graf nieskierowany to graf, którego krawędzie nie mają grotów skierowanych w kierunku któregoś z pozostałych wierzchołków grafu.
Oprócz grafów skierowanych i nieskierowanych wyróżnia się także grafy z etykietami (rysunek 2.3) oraz multigrafy (rysunek 2.4).

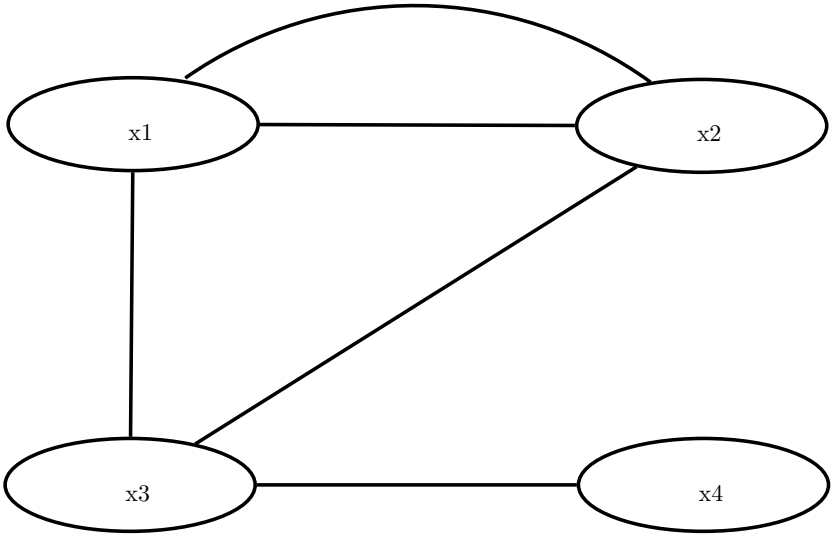
Definicja 2.4 (Graf z etykietowanymi wierzchołkami)**.** *Graf z etykietowanymi wierzchołkami G_{ew} to graf będący trójką $G_{ew} = (V, K, E)$, gdzie V to zbiór wierzchołków, K to zbiór krawędzi, a E to zbiór etykiet będącymi ciągami znaków, przyporządkowanych poszczególnym wierzchołkom grafu.* □

Definicja 2.5 (Graf z etykietowanymi krawędziami)**.** *Graf z etykietami G_{ek} to graf będący trójką $G_{ek} = (V, K, E)$, gdzie V to zbiór wierzchołków, K to zbiór krawędzi, a E to zbiór etykiet będącymi ciągami znaków, przyporządkowanych poszczególnym krawędziom grafu.* □



Rysunek 2.3: Graf skierowany z etykietami

Definicja 2.6 (Multigraf)**.** *Załóżmy, że V to zbiór wierzchołków, K to zbiór krawędzi, a n to funkcja określająca krotność krawędzi. Wtedy multigraf będzie zdefiniowany jako trójka $G_m = (V, K, n)$, gdzie jest przynajmniej jeden wierzchołek $v \in V$, do którego istnieje dojście z przynajmniej dwóch krawędzi $k \in K$.* □



Rysunek 2.4: Multigraf

2.1 Graf właściwości

W niniejszym podrozdziale omówiony zostanie graf właściwości będący szczególnym rodzajem grafu. Za [60] przytoczono definicję grafu właściwości.

Definicja 2.7 (Graf właściwości)**.** *Graf właściwości to graf będący krotką* $G_w = \langle V, K, S, W, h_k, t_k, e_v, e_k, w_v, w_k \rangle$*, gdzie:*

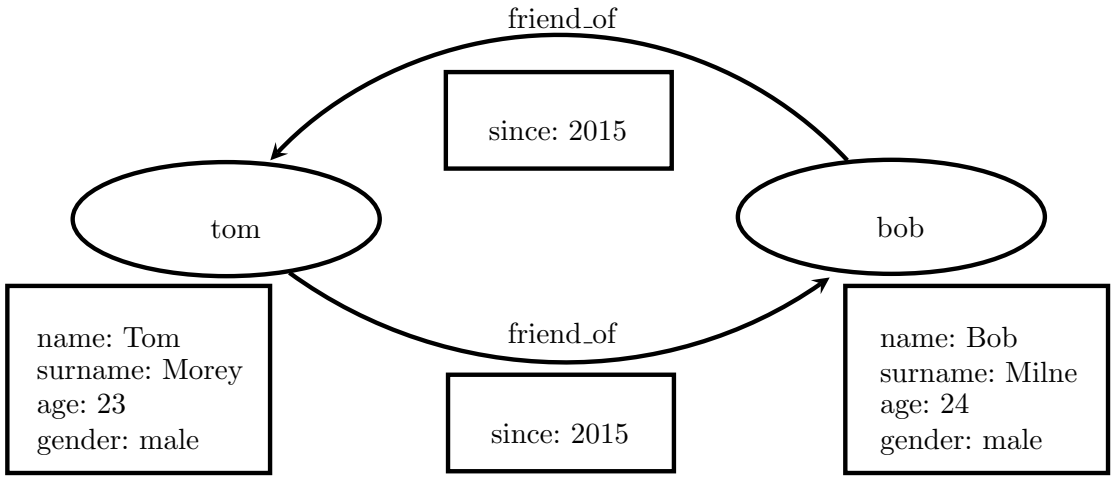
- V *to niepusty zbiór wierzchołków,*
- K *to zbiór krawędzi,*
- S *to niepusty zbiór ciągów znakowych,*
- W *zawiera właściwość w formie* $w = \langle t, v \rangle$*, gdzie* $t \in S$ *i* $v \in S$*,*
- $h_k : K \to V$ *to funkcja dostarczająca wejście dla każdej z krawędzi,*
- $t_k : K \to V$ *to funkcja dostarczająca wyjście dla każdej z krawędzi,*
- $e_v : V \to S$ *to funkcja mapująca każdy wierzchołek do etykiety,*
- $e_k : K \to S$ *to funkcja mapująca każdą krawędź do etykiety,*
- $w_v : V \to 2^W$ *to funkcja przypisująca wierzchołki do ich właściwości,*

- $w_k : V \to 2^W$ *to funkcja przypisująca krawędzie do ich właściwości.*

□

Można powiedzieć, że graf właściwości to graf skierowany (definicja 2.2) w którym:

- wierzchołki mogą być etykietowane (definicja 2.4), mogą zawierać właściwości,
- krawędzie są etykietowane (definicja 2.5) i mogą zawierać właściwości,
- właściwości są parami klucz-wartość, gdzie klucze są ciągami znaków, a wartości mogą być różnego typu.



Rysunek 2.5: Graf właściwości z dwoma wierzchołkami i krawędziami

Przykład 2.1. *Widoczny na rysunku 2.5 graf właściwości posiada następujące elementy:*

- $V = \{v_1, v_2\}$,
- $K = \{k_1, k_2\}$,
- $S = \{friend\_of, tom, bob, name, Tom, Bob, surname, Morey, Milne, age, 23, 24, gender, male, since, 2015\}$,
- $h_k(k_1) = bob$,
- $h_k(k_2) = tom$,
- $t_k(k_1) = tom$,
- $t_k(k_2) = bob$,
- $e_v(v_1) = tom$,
- $e_v(v_2) = bob$,
- $e_k(k_1) = friend\_of$,
- $e_k(k_2) = friend\_of$,
- $w_v(v_1) = \{\langle name, Tom\rangle, \langle surname, Morey\rangle, \langle age, 23\rangle, \langle gender, male\rangle\}$,
- $w_v(v_2) = \{\langle name, Bob\rangle, \langle surname, Milne\rangle, \langle age, 24\rangle, \langle gender, male\rangle\}$,
- $w_k(k_1) = \{\langle since, 2015\rangle\}$,
- $w_k(k_2) = \{\langle since, 2015\rangle\}$.

2.2 Przegląd serializacji grafowych

GEXF[1] to format oparty o XML. Cechą charakterystyczną są jednoznaczne nazwy atrybutów, którymi można definiować dany element grafu. Serializacja składa się z kilku części. Pierwsza z nich służy do opisu stworzonego dokumentu. Określane są tu takie elementy jak data ostatniej modyfikacji czy autor. Kolejna sekcja to opis

[1] `https://gephi.org/gexf/format/index.html` [dostęp: 2017-04-24]

grafu. Ona także dzieli się na kolejne podelementy – definicje wierzchołków (`nodes`) i krawędzi (`edges`). Nie jest wymagane, aby blok opisujący krawędzie znajdował się zawsze na końcu – może być on umieszczony w dowolnym miejscu sekcji opisującej graf.

Przykład 2.2. *Przykład pokazuje graf z wagami zapisany w serializacji GEXF.*

```
<gexf xmlns="http://www.gexf.net/1.3" version="1.3" xmlns:viz="http://www.gexf.
    net/1.3/viz" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:
    schemaLocation="http://www.gexf.net/1.3 http://www.gexf.net/1.3/gexf.xsd">
  <graph defaultedgetype="directed" mode="static">
    <nodes>
      <node id="0" label="etykieta0"/>
      <node id="1" label="etykieta1"/>
      <node id="2" label="etykieta2"/>
    </nodes>
    <edges>
      <edge id="0" source="1" target="2"></edge>
      <edge id="1" source="2" target="0" weight="4.0"></edge>
      <edge id="2" source="1" target="0" weight="5.0"></edge>
    </edges>
  </graph>
</gexf>
```

Serializacja *GML* [35] była projektowana z myślą bycia jak najbardziej uniwersalną. Format stara się być więc jak najbardziej elastyczny i umożliwia dodawanie dowolnych dodatkowych danych do opisów grafów, a także wierzchołków i krawędzi. Te dodatkowe elementy mogą być obsługiwane przez aplikacje lub pomijane. Dokument w formacie GML ma strukturę drzewiastą. W elemencie `graph` opisywane są elementy grafu, a w `node` i `edge` osobno każdy wierzchołek i krawędź.

Przykład 2.3. *Przykład pokazuje graf zapisany w serializacji GML.*

```
graph
[
  directed 1
  node
  [
    id 0
    label "etykieta0"
  ]
  node
  [
    id 1
    label "etykieta1"
  ]
  node
  [
    id 2
    label "etykieta2"
  ]
  edge
  [
    id 0
    source 1
    target 2
    value 1.0
  ]
  edge
  [
    id 1
    source 2
    target 0
    value 4.0
  ]
  edge
  [
    id 2
    source 1
    target 0
    value 5.0
  ]
]
```

GraphML [9] to kolejna serializacja oparta o język XML [11]. Cechuje się stosunkowo czytelną składnią i możliwością tworzenia własnych atrybutów opisujących poszczególne elementy grafu, a także niestandardowych typów danych. Każdy wierzchołek i graf musi posiadać identyfikator podany za pomocą atrybutu `id`, a krawędź oprócz tego również określone źródło (atrybut `source`) i cel (atrybut `target`). Format posiada również wsparcie dla zagnieżdżonych grafów.

Przykład 2.4. *Przykład pokazuje graf z wagami zapisany w serializacji GraphML. Podczas definicji dodatkowego atrybutu, domyślna waga została ustalona na 1.0.*

```
<graphml xmlns="http://graphml.graphdrawing.org/xmlns">
  <key attr.name="label" attr.type="string" for="node" id="label"/>
  <key attr.name="Edge Label" attr.type="string" for="edge" id="edgelabel"/>
  <key attr.name="weight" attr.type="double" for="edge" id="weight"/>
  <graph edgedefault="directed">
    <node id="0">
      <data key="label">etykieta0</data>
    </node>
    <node id="1">
      <data key="label">etykieta1</data>
    </node>
    <node id="2">
      <data key="label">etykieta2</data>
    </node>
    <edge id="0" source="1" target="2">
      <data key="weight">1.0</data>
    </edge>
    <edge id="1" source="2" target="0">
      <data key="weight">4.0</data>
    </edge>
    <edge id="2" source="1" target="0">
      <data key="weight">5.0</data>
    </edge>
  </graph>
</graphml>
```

Ucinet DL to serializacja używana w programie Ucinet[2]. Serializacja pozwala na wprowadzenie danych w kilku różnych formatach. Pierwszym z nich jest `fullmatrix`.

[2] `https://sites.google.com/site/ucinetsoftware/home` [dostęp: 2017-04-23]

W tym przypadku graf zapisuje się za pomocą kwadratowej macierzy sąsiedztwa M_{ij} o ilości wierszy i kolumn równej liczbie wierzchołków, gdzie i odwzorowuje wierzchołek startowy, a j – wierzchołek końcowy poszczególnych krawędzi. Jeżeli dana krawędź istnieje, M_{ij} przyjmuje wartość 1. W przeciwnym przypadku na skrzyżowaniu wierszy i kolumn wpisuje się 0. Serializacja obsługuje także definiowanie wag poszczególnych krawędzi. Wtedy zamiast 1 wstawia się odpowiednią liczbę z tym, że nie dozwolona jest waga 0. Obsługiwane są także liczby zmiennoprzecinkowe. Wymagane jest również podanie liczby krawędzi. Etykiety wierzchołków są opcjonalne. Można je wymienić po umieszczonym w nowej linii nagłówku `labels:`, oddzielając każdą nazwę przecinkiem. Inne formaty to m.in `Edgelist1` oraz `Nodelist1`. Pierwszy z nich pozwala na wypisanie w nowych liniach par wierzchołków, które są ze sobą połączone zamiast macierzy sąsiedztwa. W `Nodelist1` to krok pośredni pomiędzy obydwoma. W każdej linii podaje się numer lub nazwę wierzchołka oraz połączone z nim wierzchołki.

Przykład 2.5. *Przykład pokazuje graf z wagami i etykietami zapisany w serializacji UcinetDL w formacie* ***fullmatrix****.*

```
dl
format = fullmatrix
n = 3
labels:
etykieta0,etykieta1,etykieta2
data:
0.0 0.0 0.0
5.0 0.0 1.0
4.0 0.0 0.0
```

GDF jest formatem wykorzystywanym przez narzędzie GUESS[3]. Posiada wsparcie dla atrybutów zarówno w przypadku krawędzi jak i wierzchołków. Wykorzystuje konwencję znaną z formatu Comma-Separated Values (CSV), gdzie poszczególne wartości oddzielone są przecinkami. Opisy wierzchołków i krawędzi są umieszczone w oddzielnych sekcjach.

[3] `http://graphexploration.cond.org/` [dostęp: 2017-04-23]

Przykład 2.6. *Przykład pokazuje graf z wagami zapisany w serializacji GDF.*

```
nodedef> name VARCHAR,label VARCHAR
0,"etykieta0"
1,"etykieta1"
2,"etykieta2"
edgedef> node1,node2,weight DOUBLE,directed BOOLEAN
1,2,1.0,true
2,0,4.0,true
1,0,5.0,true
```

Nazwa	**Format**	**Wagi krawędzi**	**Atrybuty**	**Domyślne wartości atrybutów**
GEXF	XML	obsługuje	obsługuje	nie obsługuje
GML	tekstowy	obsługuje	obsługuje	obsługuje
GraphML	XML	obsługuje	obsługuje	obsługuje
Ucinet DL	tekstowy	obsługuje	obsługuje	nie obsługuje
GDF	tekstowy	obsługuje	obsługuje	obsługuje

Tablica 2.1: Porównanie serializacji grafowych

2.3 Język zapytań Cypher

Cypher[4] to deklaratywny język zapytań. Używany jest przede wszystkim w bazie danych Neo4j. Powołany do życia projekt OpenCypher[5], którego celem jest dostarczenie kompletnej, otwartej dokumentacji, pozwolił jednak na zaimplementowanie tego języka zapytań również w innych systemach. Cypher w swojej strukturze jest bardzo podobny do SQL [38]. Do tworzenia zapytań używa się specjalnych klauzul, których nazwy jednak nie są bezpośrednią kopią tych znanych z SQL. Pewne dodatkowe zmiany musiały także nastąpić ze względu na nieco inną specyfikę relacyjnych i grafowych baz danych. Mimo to, osoba używająca wcześniej SQL, a nawet

[4] `https://neo4j.com/docs/developer-manual/current/cypher/` [dostęp: 2017-05-10]
[5] `http://www.opencypher.org/` [dostęp: 2017-05-10]

SPARQL (omówiony w podrozdziale 1.3), powinna bez większego problemu dostrzec różnice i cechy wspólne wymienionych języków zapytań.

Wybieranie danych

Do wybierania danych w Cypherze służy głównie klauzula `MATCH`, która jest niejako odpowiednikiem `SELECT` z SQL-owych języków zapytań. Za jej pomocą określany jest wzorzec do wyszukania w bazie danych. Klauzula `WHERE` określa szczegółowe warunki zapytania. Jest ona częścią składową całego zapytania z `MATCH` i pochodnymi, nie można jej użyć osobno, ale nie jest obowiązkowa do konstrukcji zapytania wybierającego dane. Finalną klauzulą jest `RESULT`. Określa ona, jakie dane powinny być zwrócone w wyniku zapytania. Oprócz wymienionych, można użyć także dodatkowych modyfikatorów takich jak `ORDER BY` czy `LIMIT`. Pierwsza z nich służy do sortowania, a druga do ograniczania ilości wyników. Wymienione wyżej modyfikatory, jeżeli są obecne, muszą wystąpić po klauzuli `RETURN` lub `WITH`. Wspomniane `WITH` służy do łączenia części zapytań.

Przykład 2.7. *Przykład pokazuje zapytanie*[6] *`MATCH`. Zapytanie zwróci obsadę filmów, których tytuły zaczynają się na literę „Z”. Wyniki zostaną posortowane względem tytułów filmów, a liczba wyników będzie ograniczona do pierwszych dziesięciu.*

```
1  MATCH (actor:Person)-[:ACTED_IN]->(movie:Movie)
2  WHERE movie.title STARTS WITH "Z"
3  RETURN movie.title AS title, collect(actor.name) AS cast
4  ORDER BY title ASC LIMIT 10;
```

Aktualizacja danych

Oprócz wybierania danych, istotne są także operacje tworzenia i modyfikowania danych. W przypadku języka Cypher służą do tego przede wszystkim następujące klauzule:

- `LOAD CSV` – importuje wskazany plik CSV do bazy danych,
- `CREATE` – pozwala stworzyć nowy wierzchołek oraz relację,

[6] Przykład na podstawie: `https://neo4j.com/developer/cypher/` [dostęp: 2017-05-10]

- `SET` – pozwala na aktualizację etykiet, właściwości i powiązania wierzchołków,
- `DELETE` – usuwa elementy grafu bez powiązań (nie zostaną usunięte jeżeli nie są jawnie wskazane do usunięcia),
- `DETACH DELETE` – usuwa elementy grafu wraz z powiązaniami,
- `REMOVE` – usuwa właściwości i etykiety.

Przykład 2.8. *Przykład pokazuje zapytanie* ***CREATE****, w wyniku którego zostanie dodany do bazy danych wierzchołek z etykietą* **`Person`** *i trzema właściwościami –* `name`, `surname` *oraz* `city_of_born`.

```
CREATE (n:Person { name: 'Adam', surname: 'Nowak',
city_of_born: 'Bialystok' })
```

Przykład 2.9. *Przykład pokazuje zapytanie* ***SET****, w wyniku którego wszystkim osobom o nazwisku Nowak zostanie usunięta właściwość* `city_of_born`. *Zapytanie zwróci także wierzchołek po aktualizacji.*

```
MATCH (n:Person{ surname: 'Nowak'})
SET n.city_of_born = NULL
RETURN n
```

Przykład 2.10. *Przykład pokazuje zapytanie* ***DETACH DELETE****, w wyniku którego zostaną usunięte z bazy osoby o nazwisku Nowak, których miejscem urodzenia jest Białystok wraz ze wszystkimi powiązaniami.*

```
MATCH (n:Person { surname: 'Nowak', city_of_born: 'Bialystok'})
DETACH DELETE n
```

Funkcje

W zapytaniach Cyphera można korzystać ze sporego zestawu predefiniowanych funkcji takich jak:

- *funkcje boolowskie* – zwracające prawdę lub fałsz, np. `all()`, `any()` czy `exists()`,
- *funkcje skalarne* – zwracające pojedynczą wartość, np. `last()`, `properties()` czy `toFloat()`,
- *funkcje agregujące* – przyjmujące jako argumenty wiele wartości i zwracające pojedynczy wynik, np. `avg()`, `count()` czy `collect()`,
- *funkcje wypisujące* – zwracające listę wyników, np. `filter()`, `labels()`, `range()`,
- *funkcje matematyczne* – zbiór funkcji matematycznych; w tym funkcje numeryczne (np. `abs()`), logarytmiczne (np. `sqrt()`) i trygonometryczne (np. `cos()`),
- *funkcje ciągów znakowych* – używane do operowania na łańcuchach znaków, np. `lTrim()`, `split()` czy `reverse()`,
- *funkcje przestrzenne* – służą do operowania na danych przestrzennych, np. `distance()`.

Przykład 2.11. *Przykład pokazuje wykorzystanie funkcji* `count()`*. Zapytanie zwróci liczbę osób, które urodziły się w Białymstoku.*

```
1 MATCH (n:Person {city_of_born: 'Bialystok'})
2 RETURN count(*)
```

2.4 Przegląd grafowych baz danych

Neo4j [61] to dostępna na otwartej licencji grafowa baza danych. Oprócz bezpłatnej edycji społecznościowej, dostępna jest także płatna wersja komercyjna z bardziej zaawansowanymi mechanizmami, niedostępnymi w wersji bezpłatnej. Edycja komercyjna umożliwia na przykład replikację danych w klastrach, lepsze zarządzanie wydajnością, śledzenie zapytań, monitorowanie czy wykonywanie kopii zapasowych

na działającym systemie. Baza jest multiplatformowa. Stosowanym językiem zapytań jest Cypher (podrozdział 2.3). Baza jest w pełni zgodna z zestawem właściwości – Atomicity, Consistency, Isolation, Durability (ACID) oraz posiada wsparcie dla wielu języków programowania takich jak Java, C# czy Python. Neo4j ma możliwość prostej integracji z innymi rozwiązaniami włączając w to bazę MongoDB [17], Apache Spark [65] czy silnik wyszukiwania ElasticSearch [30]. Bazę można również umieścić w komercyjnych usługach „chmurowych" takich jak Amazon EC2[7] czy Microsoft Azure[8].

OrientDB [56] jest bazą danych posiadającą obsługę różnych modeli m.in. grafowego oraz dokumentowego. Edycja biznesowa zawiera dodatkowe narzędzia, ułatwiające wdrażanie i zarządzanie bazą danych, a także rozszerzone wsparcie techniczne. Podobnie jak wymieniona wcześniej Neo4j, baza danych OrientDB jest multiplatformowa. Językiem zapytań jest OrientDB SQL, którego składnia jest identyczna z SQL. Producent wprowadził do niego jednak niewielkie zmiany. Udostępniane jest także oficjalne Application Programming Interface (API) dla najpopularniejszych języków programowania i frameworków. Przy wykorzystywaniu bazy w architekturze rozproszonej, wspierana jest replikacja danych. Baza obsługuje transakcje.

ArangoDB[9], podobnie do OrientDB, jest bazą posiadającą wsparcie dla wielu różnych modeli – klucz/wartość, dokumentowego oraz grafowego. Wersja *Basic* różni się od w pełni darmowej gwarantowanym wsparciem technicznym. Edycja *Enterprise* wyróżnia się komercyjną licencją, dodatkowymi możliwościami oprogramowania, lepszym niż w Basic wsparciem technicznym oraz ofertą prywatnych szkoleń. Używanym językiem zapytań jest AQL ze składnią przypominającą nieco SQL. W porównaniu do OrientDB SQL, wprowadzone w składni SQL zmiany są jednak dużo bardziej idące. Dodana została na przykład obsługa pętli, a niektóre operacje wykonywane są przez inne niż standardowe klauzule SQL. W przypadku wyboru grafowego modelu danych, można również używać Gremlina jako języka zapytań. ArangoDB [21] wspiera transakcje definiowane przez użytkownika.

Sparksee[10] to komercyjna grafowa baza danych. Obok płatnych licencji znajduje się również darmowa licencja testowa z ograniczonymi możliwościami, a także bezpłatne licencje edukacyjne oraz deweloperskie możliwe do uzyskania po kontakcie z firmą. Dostępne są interfejsy programistyczne (API) dla wielu innych języków

[7] `https://aws.amazon.com/ec2/` [dostęp: 2017-04-13]

[8] `https://azure.microsoft.com/` [dostęp: 2017-04-13]

[9] `https://www.arangodb.com/` [dostęp: 2017-04-13]

[10] `http://www.sparsity-technologies.com/` [dostęp: 2017-04-13]

programowania włączając w to .NET, C++, Python czy Java. Rozwiązanie umożliwia eksport danych w formatach GraphML, Grahviz oraz YGRAPHML. Baza jest multiplatformowa, a jej specjalna wersja działa również na urządzeniach mobilnych (Android, BlackBerry 10 oraz iOS). Jest także pełne wsparcie dla ACID.

JanusGraph[11] to w pełni darmowa, otwartoźródłowa grafowa baza danych. Posiada wsparcie dla tworzenia kopii zapasowych na działającym systemie, co często jest dostępne dopiero w płatnych edycjach konkurencyjnych systemów. JanusGraph ponadto wyróżnia się wysoką integralnością z innymi dostępnymi rozwiązaniami. Magazynem danych może być Apache Cassandra [64], Apache HBase [62] lub Oracle Berkeley DB [45]. Wybór przynajmniej jednego z nich jest wymagany. Bazę można prosto zintegrować się m.in. z Elasticsearch oraz Apache Lucene w celu przyspieszenia indeksowania i umożliwienia tworzenia bardziej złożonych zapytań. Do analizy i przetwarzania grafów może być użyte jedno z trzech rozwiązań – Apache Spark [65], Apache Giraph [41] lub Apache Hadoop[6]. Używanym językiem zapytań jest Gremlin [49]. JanusGraph posiada wsparcie dla ACID.

[11] `http://janusgraph.org/` [dostęp: 2017-04-13]

Nazwa	**Licencja**	**Model danych**	**Język programowania**	**Serializacje**	**Języki zapytań**
Neo4j	GNU GPL 3/AGPL 3/własnościowa	grafowy (grafy właściwości)	Java	CSV, GraphML	Cypher oraz inne poprzez rozszerzenia
OrientDB	Apache License 2.0 i własnościowa	multimodelowy (grafy właściwości, dokumentowy, klucz/wartość)	Java	JSON, zrzut SQL, CSV, GraphML	OrientDB SQL, Gremlin
ArangoDB	Apache License 2.0 i własnościowa	multimodelowy (grafy właściwości, klucz/wartość, dokumentowy)	C++, JavaScript	JSON, CSV	AQL
Sparksee	własnościowa oraz licencja testowa	grafowy (grafy właściwości)	C++	GraphML, Grahviz	interfejsy programistyczne dla wielu języków programowania
JanusGraph	Apache License 2.0	grafowy (grafy właściwości)	Java	GraphML	Gremlin

Tablica 2.2: Porównanie grafowych baz danych

Rozdział 3

Transformacja RDF do grafów właściwości

W niniejszym rozdziale zostaną przedstawione metody transformacji grafów RDF do grafów właściwości. Zaprezentowane również będzie proponowane rozwiązanie bazujące na, omówionej w podrozdziale 2.4, grafowej bazie danych Neo4j, a także sposób utworzenia i zawartość zbiorów testowych. Rozdział zakończy się omówieniem wyników wykonanych eksperymentów.

3.1 Metody transformacji

Yet Another RDF Serialization (YARS) [60] została wykorzystana jako serializacja pośrednia pomiędzy grafami RDF omówionymi w podrozdziale 1.1, a grafami właściwości (podrozdział 2.1). Jest to odzwierciedlone także w składni, która jest podobna zarówno do serializacji RDF jak i Cyphera z grafowych języków zapytań. YARS posiada trzy wyodrębnione części. W pierwszej z nich określane są prefiksy, w drugiej wierzchołki, a w trzeciej relacje. Prefiksy, podobnie jak w przypadku niektórych serializacji RDF np. Turtle (podrozdział 1.2), pozwalają na skrócenie zapisu. Zamiast długich IRI używa się krótszych kluczy znakowych. Deklaracja prefiksów w YARS musi być zawsze na początku. Nazwy wierzchołków i relacji mogą być natomiast umieszczone w dowolnym miejscu dokumentu, ale po prefiksach. Wspomniana serializacja została dodatkowo rozszerzona o możliwość określania tagów języka i typów literałów.

Przykład 3.1. *Przykład pokazuje trójki RDF z przykładu 1.7 zapisane w serializacji YARS. Aby skrócić zapis, użyto prefiksów.*

```
1  :foaf: <http://xmlns.com/foaf/0.1/>
2  (a {value:<http://example.net/me#an>})
3  (b {value:"Adam Nowak"})
4  (a)-[:foaf:name]->(b)
5  (c {value:"male", lang:"en"})
6  (a)-[:foaf:gender]->(c)
```

Algorytm generowania danych w serializacji YARS mógłby wyglądać następująco:

Data: graf RDF G
Result: graf RDF zapisany w serializacji YARS Y

1 utwórz pusty zbiór identyfikatorów i wartości prefiksów P
2 utwórz pusty zbiór tematów i obiektów z odpowiadającymi im haszami H
3 **foreach** $g \in G$ **do**
4 pobierz i zapisz temat s
5 pobierz i zapisz predykat p
6 pobierz i zapisz obiekt o
7 pobierz identyfikator (p_{id}) i wartość prefiksu (p_n)
8 **if** $p_{id} \notin P$ **then**
9 dodaj prefiks do zbioru P
10 wygeneruj hash s_h dla wartości tematu s
11 **if** $s_h \notin H$ **then**
12 dodaj temat wraz z haszem do zbioru H
13 wygeneruj hash o_h dla wartości obiektu o
14 **if** $o_h \notin H$ **then**
15 dodaj obiekt wraz z haszem do zbioru H
16 utwórz relację złożoną z tematu, predykatu i obiektu
17 wróć na początek Y
18 wypisz zbiór prefiksów P
19 **return** Y

Niektóre ze standardowych operatorów algebry relacyjnej można scharakteryzować

podobnie zarówno w przypadku algebry SPARQL [18], jak i algebry Cyphera [43, 42]. Tymi operatorami są selekcja oraz projekcja.

Definicja 3.1 (Selekcja)**.** *Załóżmy, że R to relacja, a w to warunek. Wtedy selekcja to operator $\sigma_w(R)$ zwracający relację spełniającą wskazany warunek.* □

Definicja 3.2 (Projekcja)**.** *Załóżmy, że R to relacja, a $a_1, \ldots, a_n$ to lista atrybutów. Wtedy projekcja (rzut) to operator $\pi_{a_1,\ldots,a_n}(R)$ zwracający w wyniku relację powstałą po wyborze wskazanych atrybutów.* □

Definicja 3.3 (Złączenie naturalne)**.** *Załóżmy, że R_1 i R_2 to relacje. Wtedy złączenie naturalne to operator $R_1 \bowtie R_2$ zwracający w wyniku iloczyn kartezjański relacji R_1 i R_2 z pominięciem krotek o tych samych nazwach.* □

Poniżej zostały zdefiniowane specyficzne operatory algebry Cyphera opracowane na podstawie [43, 42].

Definicja 3.4 (Pobranie wierzchołków)**.** *Załóżmy, że z to pojedynczy atrybut, a $e_1, \ldots, e_n$ to lista etykiet. Wtedy pobranie wierzchołków to operacja $\bigcirc_{(z:e_1,\ldots,e_n)}$ zwracająca w wyniku relację R zawierającą wierzchołki, które posiadają wszystkie etykiety z listy $e_1, \ldots, e_n$.* □

Definicja 3.5 (Rozszerzenie w górę)**.** *Załóżmy, że R to relacja, a_1 i a_2 to nowe atrybuty, p to istniejący atrybut, a $d_1, \ldots, d_n$ i $k_1, \ldots, k_n$ to etykiety. Wtedy rozszerzenie w górę to operator $\uparrow^{(p:k_1,\ldots,k_n)}_{(a_2)} [a_1 : d_1, \ldots, d_n](R)$, którego rezultatem jest dodanie atrybutu a_2 do krawędzi na skierowanej ścieżce od p do a_1.* □

Przykład 3.2. *Przykład pokazuje zapytanie SPARQL* `SELECT ?s ?p WHERE {?s ?p <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/Review>}` *zapisane w algebrze SPARQL.*

$(\pi_{s \leftarrow ?subject, p \leftarrow ?predicate}(\sigma_{?object = http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/Review}(Triples))$

Przykład 3.3. *Przykład pokazuje zapytanie* `MATCH (s)-[p]->(o)WHERE o.value = "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/Review" RETURN s, p` *równoważne z zapytaniem z przykładu 3.2 zapisane w algebrze Cyphera.*

$\pi_{s,p}(\sigma_{o.value = http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/Review}(\uparrow^o_s [e : p] \bigcirc_s) Triples)$

Podobnie można przekształcić inne zapytania zawierające konstrukcje z języków SPARQL i Cypher.
Poniżej przedstawiono algorytm mapowania zapytania w języku SPARQL na zapytanie Cypher.

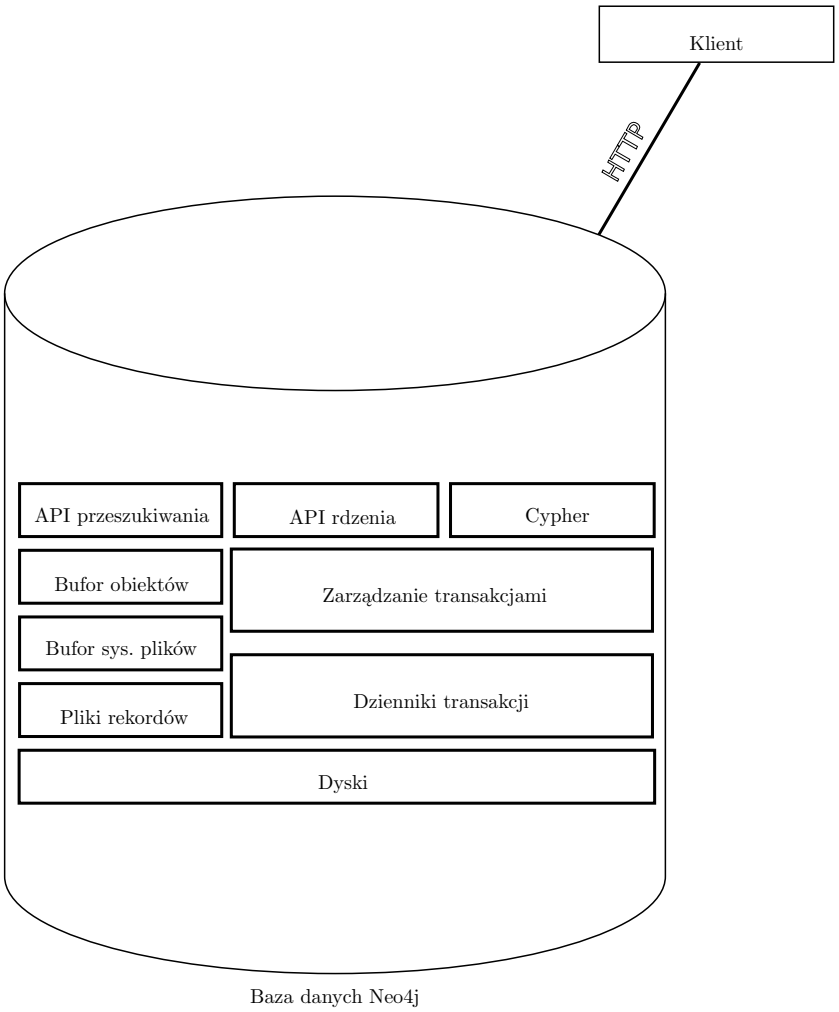
Data: zapytanie SPARQL Q
Result: zapytanie Cypher C
utwórz algebrę Cypher A_C
tłumacz Q na algebrę SPARQL A_Q
foreach $a_q \in A_Q$ **do**
    if *temat LUB obiekt jest zmienną* **then**
        dodaj elementy do projekcji A_C
    else
        dodaj elementy do selekcji A_C
    dodaj predykat do rozszerzenia A_C
    dodaj obiekt do pobieranie_wierzcholka A_C
    if *istnieje następny element w kolekcji* a_q **then**
        dodaj naturalne złączenie A_C
    if *operacja inna niż selekcja LUB projekcja* $\in A_Q$ **then**
        dołącz operację A_C
tłumacz A_C na C
return C

3.2 Architektura systemu testowego

Prezentowane rozwiązanie[1] oparte jest o grafową bazę danych Neo4j omówioną w podrozdziale 2.4 i otwartoźródłowy plugin autorstwa Niclas Hoyer[2]. Wspiera zarówno omówiony w podrozdziale 2.3 język zapytań Cypher jak i używany w Semantycznym Internecie SPARQL omówiony w podrozdziale 1.3. W przypadku wybierania danych, oprócz składni `SELECT` wspierane są również `CONSTRUCT` i `ASK`. Możliwy jest także prosty import, zamiana i usuwanie danych za pomocą języka zapytań Cypher i REST-owych [26] mechanizmów, a oprócz tego podstawowe wnioskowanie

[1] `https://github.com/lszeremeta/neo4j-sparql-extension-yars` [dostęp: 2017-07-10]
[2] `https://github.com/niclashoyer/neo4j-sparql-extension` [dostęp: 2017-06-15]

OWL2 [37]. Dzięki doimplementowaniu własnego parsera serializacji YARS[3] (przykład 3.1) do frameworka Sesame (omówionego w podrozdziale 1.4) oraz zmodyfikowaniu API Sesame[4], możliwy stał się import danych w tym formacie.
Na rysunku 3.1 została przedstawiona architektura systemu testowego. Serwer składa się z API przeszukiwania, API rdzenia, a także, omówionego w podrozdziale 2.3, języka zapytań Cypher. Oprócz tego istotnymi elementami są bufory obiektów i systemu plików, pliki rekordów, a także elementy odpowiedzialne za zarządzanie transakcjami. Klient łączy się z serwerem za pomocą protokołu Hypertext Transfer Protocol (HTTP) [25].



Rysunek 3.1: Architektura systemu testowego

Można znaleźć podobne pomysły na osiągnięcie wymienionych wyżej celów. Nie

[3] `https://github.com/lszeremeta/sesame-rio-yars` [dostęp: 2017-07-10]
[4] `https://github.com/lszeremeta/sesame-rio-api` [dostęp: 2017-07-10]

są one jednak pozbawione pewnych wad. Autorzy w [34] porównują wydajność różnych rozwiązań przechowywania danych. W artykule omawiają magazyny grafów RDF takie jak Virtuoso i Blazegraph[5], a także relacyjną bazę PostgreSQL [22] oraz grafową – Neo4j. Niestety autorzy rozważają wyłącznie specyficzny model danych z WikiData [63]. Autorzy w [32] skupiają się na formalnym zdefiniowaniu obu modeli i znalezieniu sposobu transformacji grafów RDF na grafy właściwości. Zauważają, że grafowe bazy danych, po odpowiednich modyfikacjach, mogłyby z powodzeniem przechowywać dane połączone i umożliwiać do nich dostęp za pomocą najbardziej popularnych grafowych języków zapytań. W tym celu autorzy prezentują rozszerzoną, niestandardową serializację Turtle, SPARQL* będącą rozszerzeniem SPARQL oraz RDF* będące rozszerzeniem RDF. Niestety RDF* i SPARQL*, oprócz tego, że nie jest standardem, posiadają tylko jedną implementację.

3.3 Zbiory danych testowych

Wykorzystany Berlin SPARQL Benchmark (BSBM) [8] zawiera dane e-commerce. Wśród nich znajdują się zestawy produktów oferowane przez różnych producentów. Produkty mają przypisane również swoje cechy. Benchmark zawiera także opinie konsumentów o produktach oraz oferty. Generowane dane mogą być z łatwością skalowane do żądanej ilości produktów. W BSBM wbudowana jest obsługa wielu serializacji RDF omówionych w podrozdziale 1.2 takich jak N-Triples czy Turtle. Dzięki implementacji konwertera[6], możliwe stało się wygenerowanie danych testowych w formacie YARS.
Model danych składa się z takich klas jak:

- `Product` – opisy produktów,
- `ProductType` – opisy typów produktów,
- `ProductFeature` – opisy cech produktów,
- `Producer` – opisy producentów,
- `Vendor` – opisy sprzedawców,
- `Offer` – opisy ofert,

[5] `https://www.blazegraph.com/` [dostęp: 2017-06-10]
[6] `https://github.com/lszeremeta/yars-conv` [dostęp: 2017-07-10]

- `Review` – opisy recenzji produktów,
- `Person` – opisy osób.

Produkty (`Product`) są podzielone na różne kategorie (`ProductType`), mają wiele cech (`ProductFeature`), są produkowane przez wielu producentów (`Producer`) oraz sprzedawane przez wielu sprzedawców (`Vendor`). Opinie na temat produktów złożone są z tytułu oraz opisu.
W tabeli 3.1 przedstawiono szczegółowe dane na temat ilości poszczególnych elementów w każdym z testowych zbiorów.

Zbiór	**Trójki**	**Produkty**	**Typy produktów**	**Cechy produktów**	**Oferty**	**Osoby**	**Recenzje**
10	5007	10	7	289	200	6	100
20	8498	20	7	289	400	10	200
30	12022	30	7	289	600	16	300
40	16981	40	13	569	800	21	400
50	22729	50	21	999	1000	26	500
60	26263	60	21	999	1200	30	600
70	29847	70	21	999	1400	37	700
80	33354	80	21	999	1600	41	800
90	36879	90	21	999	1800	46	900
100	40377	100	21	999	2000	50	1000

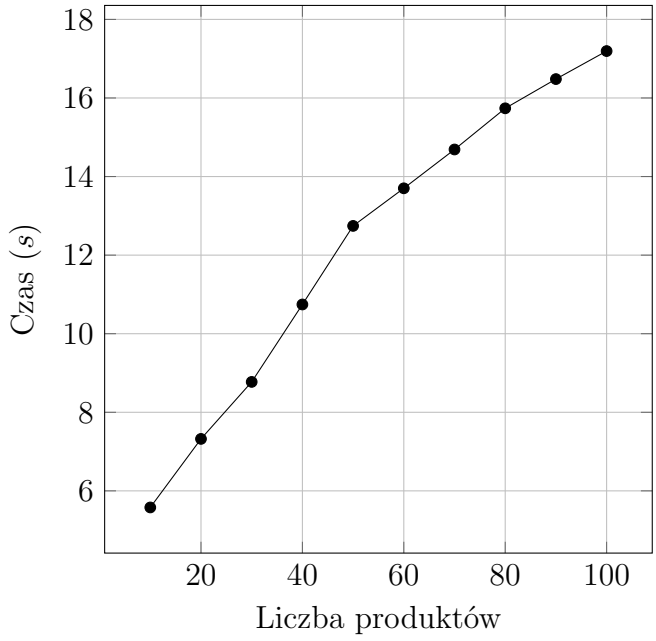
Tablica 3.1: Zawartość zbiorów testowych

3.4 Eksperymenty

W tym podrozdziale zostaną przedstawione wyniki eksperymentów – testów czasu importowania danych testowych do bazy danych oraz długości wykonywania zapytań. Jako zbiory testowe zostały wykorzystane dane wygenerowane za pomocą opisanego w podrozdziale 3.3 Berlin SPARQL Benchmark (BSBM). Generator nie obsługuje proponowanej serializacji YARS (przykład 3.1), dlatego został zaimplementowany odpowiedni konwerter z serializacji Turtle (przykład 1.9). Pozwoliło to stworzyć testowe zbiory danych w serializacji YARS. Wszystkie eksperymenty zo-

stały wykonane na komputerze z czterordzeniowym procesorem AMD A6-6310 APU (od 1,80 GHz do 2,40 GHz) z grafiką AMD Radeon R4 oraz 8GB pamięci RAM.

Na rysunku 3.2 zostały zaprezentowane czasy importowania poszczególnych zbiorów danych do bazy danych. Czasy rosną potęgowo przy współczynniku determinacji $R^2 \approx 0,99$. Każdy test został wykonany dziesięciokrotnie na świeżo uruchomionej bazie nieprzechowującej żadnych danych. Podane niżej czasy wykonania są średnią arytmetyczną z dziesięciu pomiarów. Przedstawione wyniki wydają się być akceptowalne, nawet w przypadku dużych zbiorów.



Rysunek 3.2: Testy importowania danych testowych do bazy

Kolejne eksperymenty dotyczyły czasu wykonywania wybranych zapytań SPARQL. Każde zapytanie było wykonywane dziesięciokrotnie, po każdorazowym restarcie bazy danych. Pozwoliło to wykluczyć wpływ wcześniejszych zapytań testowych na wynik końcowy. Podane czasy wykonania są średnią arytmetyczną z dziesięciu pomiarów.

Zapytania testowe zostały przedstawione w tabeli 3.2. Zapytania od pierwszego do siódmego to zapytania typu `SELECT`. Pierwsze dwa to zapytania z jednym wzorcem trójki (definicja 1.7) z tym, że w zapytaniu Q2 dodane są klauzule `LIMIT` i `OFFSET`. Q3 i Q4 to zapytania z dwoma wzorcami gdzie w Q4 występuje złączenie. Q5 to zapytanie z klauzulą `ORDER BY`, a Q6 to zapytanie agregujące z `GROUP BY`. Q7 to zapytanie zawierające podzapytanie, a ostatnie to zapytanie konstruujące nowy

graf wynikowy (`CONSTRUCT`). W tabeli 3.3 zostały zaprezentowane sprawdzane cechy w zapytaniach.

Identyfikator	Zapytanie
Q1	`SELECT ?s ?p WHERE {?s ?p <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/Offer>}`
Q2	`SELECT ?s ?p WHERE {?s ?p <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/Offer>} LIMIT 10 OFFSET 10}`
Q3	`SELECT ?s ?p ?s2 ?p2 WHERE {?s ?p <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/Offer> . ?s2 ?p2 <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer1/Product37>}`
Q4	`SELECT ?x WHERE {<http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromRatingSite1/Review1> <http://purl.org/stuff/rev#reviewer> ?x . ?x <http://purl.org/dc/elements/1.1/publisher> <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromRatingSite1/RatingSite1>}`
Q5	`SELECT ?s ?p WHERE {?s ?p <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/Product>} ORDER BY DESC(?s)`
Q6	`SELECT (COUNT(?offer)AS ?count)WHERE {?offer <http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/price> ?price } GROUP BY ?price ORDER BY DESC(COUNT(?offer))`
Q7	`SELECT ?reviewer WHERE {?reviewer <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> . FILTER EXISTS {?reviewer <http://xmlns.com/foaf/0.1/name> ?name}}`
Q8	`CONSTRUCT {?vendor <http://xmlns.com/foaf/0.1/workplaceHomepage> ?site} WHERE {?vendor <http://xmlns.com/foaf/0.1/homepage> ?site}`

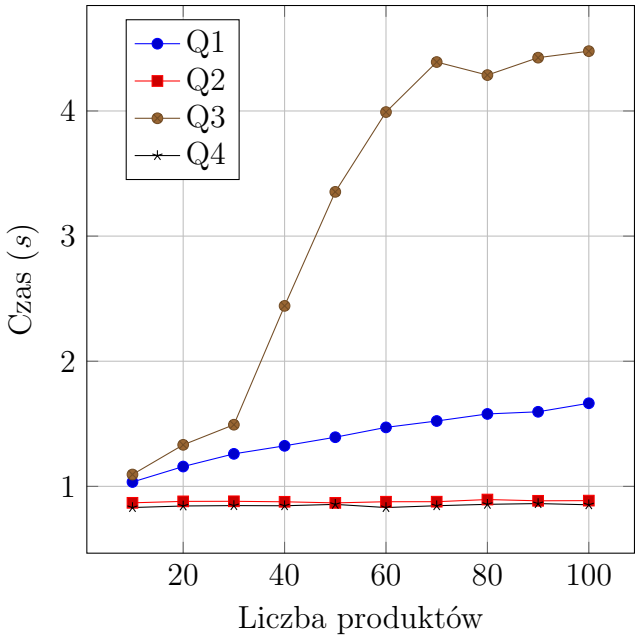
Tablica 3.2: Zapytania testowe

Na wykresie 3.3 widoczne są czasy wykonywania zapytań Q1, Q2, Q3 i Q4. Czas wykonania zapytania Q1 rośnie potęgowo przy współczynniku determinacji $R^2 \approx 0,99$, a w przypadku Q3 również potęgowo przy $R^2 \approx 0,92$. Można zauważyć, że wykonanie zapytania Q2 i Q4 zajmuje podobną ilość czasu dla każdego zbioru

	Wybieranie pojedyncze	**Wybieranie złożone**	**Porządek**	**Limit**	**Przesunięcie**	**Grupowanie**	**Tworzenie**
Q1	tak	nie	nie	nie	nie	nie	nie
Q2	tak	nie	nie	tak	tak	nie	nie
Q3	tak	nie	nie	nie	nie	nie	nie
Q4	nie	tak	nie	nie	nie	nie	nie
Q5	tak	nie	tak	nie	nie	nie	nie
Q6	tak	nie	tak	nie	nie	tak	nie
Q7	tak	tak	nie	nie	nie	nie	nie
Q8	tak	nie	nie	nie	nie	nie	tak

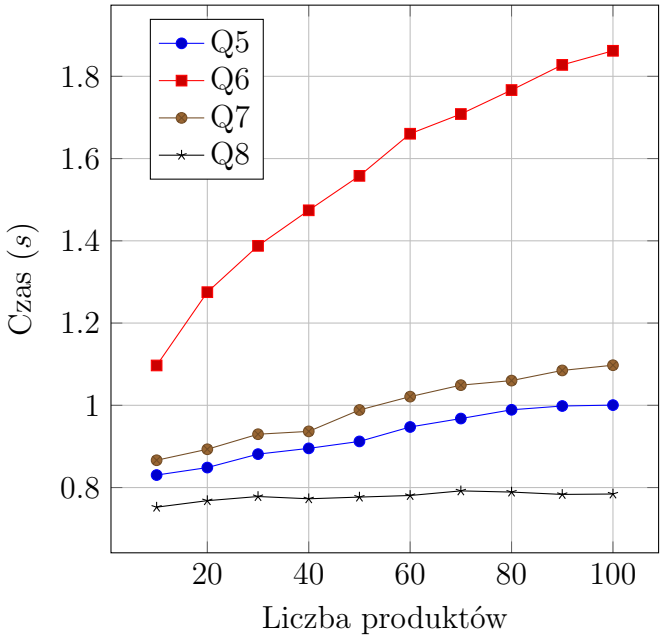
Tablica 3.3: Sprawdzane cechy w zapytaniach

testowego. Jest to poniżej sekundy bez względu na wielkość testowanych zbiorów danych. Najwięcej czasu zajmuje wykonanie zapytania Q3. W przypadku zestawów testowych składających się z powyżej 60 produktów, zapytanie to wykonuje się ponad 4 sekundy i zwraca dużą liczbę wyników. Na przykładzie zapytań Q1 i Q2 można zaobserwować, jak ograniczenie ilości zwracanych wyników przekłada się na krótszą długość wykonywania zapytania.



Rysunek 3.3: Czasy wykonywania zapytań Q1, Q2, Q3 i Q4

Wykres 3.4 przedstawia czasy wykonania zapytań Q5, Q6, Q7 i Q8. Czas wykonania zapytań Q5 i Q7 rośnie liniowo przy współczynniku determinacji $R^2 \approx 0,98$, w przypadku Q6 potęgowo przy $R^2 \approx 1$. Można zauważyć, że najkrócej wykonującym się zapytaniem jest Q8 konstruujące nowy graf za pomocą składni `CONSTRUCT`. Wynika to z faktu, ze tylko to zapytanie wykorzystuje pojedynczy wzorzec trójki. Najdłużej wykonywało się natomiast zapytanie agregujące Q6. Można także zauważyć, że wzrost czasu wykonania zapytań Q5 i Q7 okazał się podobny.



Rysunek 3.4: Czasy wykonywania zapytań Q5, Q6, Q7 i Q8

Większość zapytań wykonała się w czasie mniejszym niż 2 sekundy. Jedynie zapytanie Q3 zwracające dużą ilość wyników wykonywało się dłużej. W jego przypadku zanotowano również największe wzrosty czasu wykonania w zależności od wielkości zbioru testowego.

Podsumowanie

Resource Description Framework (RDF) i grafy właściwości stwarzają odmienne sposoby przechowywania danych grafowych. Pierwszy z nich używany jest przede wszystkim w Semantycznym Internecie, a drugi w wielu innych systemach bazodanowych. W ostatnich latach możemy zauważyć ogromny wzrost ilości danych, gdzie próba pogodzenia obu tych „światów", wydaje się w takim kontekście niezwykle istotna.

Celem niniejszej pracy było omówienie zagadnień związanych z Semantycznym Internetem oraz grafami właściwości, a także prezentacja rozwiązania łączącego obie te koncepcje. Cel ten został osiągnięty w całości.

W ramach niniejszej pracy omówiono dwie różne koncepcje przechowywania danych grafowych, ich serializacje, języki zapytań oraz systemy w których są wykorzystywane. Zaprezentowano również metody i algorytmy transformacji oraz rozwiązania pozwalające na połączenie obu omówionych koncepcji. Jest to przede wszystkim serializacja YARS oraz system oparty o grafową bazę danych obsługujący, znany z Semantycznego Internetu, język zapytań SPARQL.

Testy wydajnościowe zostały przeprowadzone na wygenerowanych przy pomocy Berlin SPARQL Benchmark danych testowych. Dzięki zaimplementowanemu konwerterowi, możliwa stała się konwersja utworzonych zbiorów na pośrednią serializację YARS.

Wyniki eksperymentów pokazują spory potencjał zaproponowanego rozwiązania. Niemal wszystkie testowane zapytania wykonują się poniżej 2 sekund bez względu na wielkość zbioru testowego. Również czas importu danych w formacie YARS można uznać za zadowalający nawet w przypadku większych zbiorów.

Przyszłe prace powinny koncentrować się na rozszerzeniu wnioskowania RDFS i OWL2 przy wykorzystaniu serializacji Notation 3 Logic [2] oraz dalszych krokach mających na celu poprawę wydajności (m.in. optymalizacja zapytań).

Bibliografia

[1] The Unicode® Standard Version 9.0 – Core Specification. Technical report, Unicode Consortium, Mountain View, CA, June 2016. `http://www.unicode.org/versions/Unicode9.0.0/`.

[2] Dörthe Arndt, Ruben Verborgh, Jos De Roo, Hong Sun, Erik Mannens, and Rik Van de Walle. Semantics of Notation3 Logic: A solution for implicit quantification. In *Proceedings of the 9th International Web Rule Symposium*, August 2015.

[3] Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. DBpedia: A Nucleus for a Web of Open Data. *The semantic web*, pages 722–735, 2007.

[4] Jie Bao, Elisa F. Kendall, Deborah L. McGuinness, and Peter F. Patel-Schneider. OWL 2 Web Ontology Language Quick Reference Guide (Second Edition). W3C recommendation, World Wide Web Consortium, December 2012. `http://www.w3.org/TR/2012/REC-owl2-quick-reference-20121211/`.

[5] Tim Berners-Lee, James Hendler, Ora Lassila, et al. The semantic web. *Scientific american*, 284(5):28–37, 2001.

[6] Milind Bhandarkar. MapReduce programming with apache Hadoop. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–1. IEEE, 2010.

[7] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data-the story so far, 2009.

[8] Christian Bizer and Andreas Schultz. The Berlin SPARQL Benchmark, 2009.

[9] Ulrik Brandes, Markus Eiglsperger, Ivan Herman, Michael Himsolt, and M Scott Marshall. GraphML progress report structural layer proposal. In *International Symposium on Graph Drawing*, pages 501–512. Springer, 2001.

[10] Tim Bray. The JavaScript Object Notation (JSON) Data Interchange Format. RFC 7159, RFC Editor, March 2014. `https://tools.ietf.org/html/rfc7159`.

[11] Tim Bray, Eve Maler, Jean Paoli, François Yergeau, John Cowan, and Michael Sperberg-McQueen. Extensible Markup Language (XML) 1.1 (Second Edition). W3C recommendation, W3C, August 2006. `http://www.w3.org/TR/2006/REC-xml11-20060816`.

[12] Dan Brickley and R.V. Guha. RDF Schema 1.1. W3C recommendation, World Wide Web Consortium, February 2014. `http://www.w3.org/TR/2014/REC-rdf-schema-20140225/`.

[13] Dan Brickley and Libby Miller. FOAF Vocabulary Specification 0.99, January 2014. `http://xmlns.com/`.

[14] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *International semantic web conference*, pages 54–68. Springer, 2002.

[15] Gavin Carothers. RDF 1.1 N-Quads. W3C recommendation, World Wide Web Consortium, February 2014. `http://www.w3.org/TR/2014/REC-n-quads-20140225/`.

[16] Gavin Carothers and Andy Seabourne. RDF 1.1 N-Triples. W3C recommendation, World Wide Web Consortium, February 2014. `http://www.w3.org/TR/2014/REC-n-triples-20140225/`.

[17] Kristina Chodorow. *MongoDB: The Definitive Guide: Powerful and Scalable Data Storage*. O'Reilly Media, Inc., 2013.

[18] Richard Cyganiak. A relational algebra for SPARQL. *Digital Media Systems Laboratory HP Laboratories Bristol. HPL-2005-170*, 35, 2005.

[19] Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax. W3C recommendation, World Wide Web Consortium, February 2014. `http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/`.

[20] Jeffrey Dean and Sanjay Ghemawat. MapReduce: a flexible data processing tool. *Communications of the ACM*, 53(1):72–77, 2010.

[21] Lucas Dohmen, PDR Klamma, and Frank Celler. *Algorithms for Large Networks in the NoSQL Database ArangoDB*. PhD thesis, Bachelors thesis, RWTH Aachen, Aachen, 2012.

[22] Korry Douglas and Susan Douglas. *PostgreSQL: a comprehensive guide to building, programming, and administering PostgresSQL databases*. SAMS publishing, 2003.

[23] Martin Duerst and Michel Suignard. Internationalized Resource Identifiers (IRIs). Technical report, The Internet Engineering Task Force, January 2005. `http://www.ietf.org/rfc/rfc3987.txt`.

[24] Orri Erling and Ivan Mikhailov. RDF Support in the Virtuoso DBMS. In *Networked Knowledge-Networked Media*, pages 7–24. Springer, 2009.

[25] Roy Fielding and Julian Reschke. Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230, RFC Editor, June 2014. `https://tools.ietf.org/html/rfc7230`.

[26] Roy Thomas Fielding. REST: architectural styles and the design of network-based software architectures. *Doctoral dissertation, University of California*, 2000.

[27] Fabien Gandon and Guus Schreiber. RDF 1.1 XML Syntax. W3C recommendation, W3C, February 2014. `http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/`.

[28] Paula Gearon, Alexandre Passant, and Axel Polleres. SPARQL 1.1 Update. W3C recommendation, W3C, March 2013. `http://www.w3.org/TR/2013/REC-sparql11-update-20130321/`.

[29] Lars George. *HBase: The Definitive Guide: Random Access to Your Planet-Size Data*. O'Ŕeilly Media, Inc., 2011.

[30] Clinton Gormley and Zachary Tong. *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine*. O'Reilly Media, Inc., 2015.

[31] Steve Harris and Andy Seaborne. SPARQL 1.1 Query Language. W3C recommendation, World Wide Web Consortium, March 2013. http://www.w3.org/TR/2013/REC-sparql11-query-20130321/.

[32] Olaf Hartig. Reconciliation of RDF* and property graphs. *arXiv preprint arXiv:1409.3288*, 2014.

[33] Austin Haugen. The Open Graph Protocol Design Decisions. In *International Semantic Web Conference*, pages 338–338. Springer, 2010.

[34] Daniel Hernández, Aidan Hogan, Cristian Riveros, Carlos Rojas, and Enzo Zerega. Querying Wikidata: Comparing SPARQL, Relational and Graph Databases. In *International Semantic Web Conference (2)*, pages 88–103, 2016.

[35] Michael Himsolt. GML: A portable graph file format. Technical report, 1997. http://www.fim.uni-passau.de/fileadmin/files/lehrstuhl/brandenburg/projekte/gml/gml-technical-report.pdf.

[36] Pascal Hitzler, Markus Krotzsch, and Sebastian Rudolph. *Foundations of semantic web technologies*. CRC Press, 2011.

[37] Ian Horrocks, Boris Motik, Zhe Wu, Achille Fokoue, and Bernardo Cuenca Grau. OWL 2 Web Ontology Language Profiles (Second Edition). W3C recommendation, W3C, December 2012. http://www.w3.org/TR/2012/REC-owl2-profiles-20121211/.

[38] Information technology – Database languages – SQL – Part 1: Framework (SQL/Framework). Standard, International Organization for Standardization, Geneva, CH, December 2016.

[39] Gregg Kellogg, Markus Lanthaler, and Manu Sporny. JSON-LD 1.0. W3C recommendation, W3C, January 2014. http://www.w3.org/TR/2014/REC-json-ld-20140116/.

[40] Gunter Ladwig and Andreas Harth. CumulusRDF: Linked Data Management on Nested Key-Value Stores. In *The 7th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2011)*.

[41] Claudio Martella, Roman Shaposhnik, Dionysios Logothetis, and Steve Harenberg. *Practical graph analytics with Apache Giraph*. Springer, 2015.

[42] József Marton and Gábor Szárnyas. Formalisation of openCypher Queries in Relational Algebra (Extended Version). 2017.

[43] József Marton, Gábor Szárnyas, and Dániel Varró. Formalising opencypher Graph Queries in Relational Algebra. *arXiv preprint arXiv:1705.02844*, 2017.

[44] Brian McBride. Jena: A semantic web toolkit. *IEEE Internet computing*, 6(6):55–59, 2002.

[45] Michael A Olson, Keith Bostic, and Margo I Seltzer. Berkeley DB. In *USENIX Annual Technical Conference, FREENIX Track*, pages 183–191, 1999.

[46] Nikolaos Papailiou, Ioannis Konstantinou, Dimitrios Tsoumakos, and Nectarios Koziris. H2RDF: adaptive query processing on RDF data in the cloud. In *Proceedings of the 21st International Conference on World Wide Web*, pages 397–400. ACM, 2012.

[47] Nikolaos Papailiou, Dimitrios Tsoumakos, Ioannis Konstantinou, Panagiotis Karras, and Nectarios Koziris. H 2 RDF+: an efficient data management system for big RDF graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 909–912. ACM, 2014.

[48] Eric Prud'hommeaux and Gavin Carothers. RDF 1.1 Turtle. W3C recommendation, World Wide Web Consortium, February 2014. `http://www.w3.org/TR/2014/REC-turtle-20140225/`.

[49] Marko A Rodriguez. The Gremlin Graph Traversal Machine and Language. In *Proceedings of the 15th Symposium on Database Programming Languages*, pages 1–10. ACM, 2015.

[50] Guus Schreiber and Yves Raimond. RDF 1.1 Primer. W3C note, W3C, June 2014. `http://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/`.

[51] Andy Seaborne and Gavin Carothers. RDF 1.1 TriG. W3C recommendation, W3C, February 2014. `http://www.w3.org/TR/2014/REC-trig-20140225/`.

[52] Yakov Shafranovich. Common Format and MIME Type for Comma-Separated Values (CSV) Files. RFC 4180, RFC Editor, October 2005. `http://www.rfc-editor.org/rfc/rfc4180.txt`.

[53] Thomas Steiner, Ruben Verborgh, Raphaël Troncy, Joaquim Gabarro, and Rik Van de Walle. Adding Realtime Coverage to the Google Knowledge Graph. In *Proceedings of the 2012th International Conference on Posters & Demonstrations Track-Volume 914*.

[54] Łukasz Szeremeta. Buforowanie trójek w magazynach grafów RDF. Bachelor's Thesis, University of Bialystok, Poland, 2015. `http://repozytorium.uwb.edu.pl/jspui/handle/11320/3160`.

[55] Łukasz Szeremeta and Dominik Tomaszuk. Document-oriented RDF graph store. *Studia Informatica*, 38(2):31–43, 2017.

[56] Claudio Tesoriero. *Getting Started with OrientDB*. Packt Publishing Ltd, 2013.

[57] Dominik Tomaszuk. Flat triples approach to RDF graphs in JSON. In *Proceedings of W3C Workshop-RDF Next Steps. World Wide Web Consortium*. `https://www.w3.org/2009/12/rdf-ws/papers/ws02`.

[58] Dominik Tomaszuk. Document-oriented triplestore based on RDF/JSON. *Logic, Philosophy and Computer Science*, (22(35)):125–140, 2010.

[59] Dominik Tomaszuk. *New methods for accessing data in RDF graph structures*. PhD thesis, Warsaw University of Technology, 2014.

[60] Dominik Tomaszuk. RDF Data in Property Graph Model. In *Metadata and Semantics Research: 10th International Conference, MTSR 2016, Göttingen, Germany, November 22-25, 2016, Proceedings*, pages 104–115. Springer, 2016.

[61] Rik Van Bruggen. *Learning Neo4j*. Packt Publishing Ltd, 2014.

[62] Mehul Nalin Vora. Hadoop-HBase for large-scale data. In *Computer science and network technology (ICCSNT), 2011 international conference on*, volume 1, pages 601–605. IEEE, 2011.

[63] Denny Vrandečić and Markus Krötzsch. Wikidata: a free collaborative knowledgebase. *Communications of the ACM*, 57(10):78–85, 2014.

[64] Guoxi Wang and Jianfeng Tang. The nosql principles and basic application of cassandra model. In *Computer Science & Service System (CSSS), 2012 International Conference on*, pages 1332–1335. IEEE, 2012.

[65] Matei Zaharia, Reynold S Xin, Patrick Wendell, Tathagata Das, Michael Armbrust, Ankur Dave, Xiangrui Meng, Josh Rosen, Shivaram Venkataraman, Michael J Franklin, et al. Apache Spark: A unified engine for big data processing. *Communications of the ACM*, 59(11):56–65, 2016.