
Uniwersytet w Białymstoku

Wydział Matematyki i Informatyki

Instytut Informatyki

# Buforowanie trójek w magazynach grafów RDF

Łukasz Szeremeta

Promotor:
dr inż. Dominik Tomaszuk

Białystok, 2015

**Streszczenie**

Internet, używany przez nas na co dzień, nie był tworzony do komfortowej analizy danych, a raczej wymiany dokumentów. Odpowiedzą na ten stan rzeczy, zdaje się być projekt Semantycznego Internetu oraz Danych Połączonych. Dzięki temu podejściu do globalnej pajęczyny, możemy konstruować konkretne zapytania oraz uzyskiwać przydatne informacje bez większego wysiłku. W pracy zostaną omówione zagadnienia związane z Semantycznym Internetem, Danymi Połączonymi, RDF-em będącym językiem opisu modelu danych, różnymi serializacjami RDF oraz koncepcjami buforowania danych w bazach danych. Oprócz tego zostanie zaprezentowany RTriples – kompletny magazyn grafów RDF zbudowany w oparciu o bazę danych NoSQL, przystosowaną do zastosowań związanych z Semantycznym Internetem i Danymi Połączonymi. Całość została uzupełniona o testy wydajnościowe zaprezentowanego rozwiązania.

**Słowa kluczowe:** Semantyczny Internet, Dane Połączone, RDF, JSON, buforowanie, indeksy, NoSQL, RethinkDB, RTriples, języki zapytań, bazy danych

**Abstract**

Internet used by us every day was not originally created for convenient data analysis, but rather for exchange documents. The answer for this state of affairs seems to be Semantic Web and Linked Data projects. Using this approach to World Wide Web, we can construct certain search queries and obtain useful information without much effort. This work discussed issues related to the Semantic Web, Linked Data, RDF (language describing data model), different RDF serializations and database data caching concepts. In addition, I present RTriples – complete RDF graph store based on NoSQL database, adapted for use in Semantic Web and Linked Data environments. The whole is supplemented by performance tests of presented solution.

**Keywords:** Semantic Web, Linked Data, RDF, JSON, caching, indexes, NoSQL, RethinkDB, RTriples, query languages, databases

# Spis treści

# Wstęp

Każdy z nas korzysta z Internetu. Globalna pajęczyna pozwala na bezpośredni kontakt z osobami rozsianymi po całym świecie, wspólną pracę czy rozrywkę, a to tylko ułamek dostępnych możliwości.

Dotychczasowy model Internetu ma też jednak pewne wady. Od początku był projektowany w celu łatwiejszej wymiany dokumentów. Mamy prosty i niemal bezproblemowy dostęp do różnego rodzaju serwisów internetowych. Wszelkie dane są jednak dostępne albo jako strony HTML, albo opublikowane w zewnętrznych formatach. Biorąc pod uwagę zwiększającą się codziennie ilość informacji, ich analiza, nie jest wcale łatwa, zarówno przez ludzi, jak i maszyny.

Semantyczny Internet i Dane Połączone starają się być odpowiedzią na te bolączki. Operując na niemal surowych danych, mamy możliwość konstruowania zdecydowanie bardziej konkretnych zapytań, w odpowiedzi otrzymując, przydatne dla nas odpowiedzi. To wszystko bez większego wysiłku. Dane mogą być dodatkowo wzbogacone o kontekst, który jest znany nawet z codziennego życia. Praktyczne zastosowania znajdują się same – od szybkiego wypisania osób urodzonych w danym dniu, po wyszukiwanie białek mających pożądane właściwości w zastosowaniach medycznych[1]. Na tak konkretne zapytania, prawdopodobnie nie otrzymamy odpowiedzi w standardowych wyszukiwarkach. W ich przypadku mniej ważny jest kontekst, a jedynie obecność poszukiwanej treści na którejś z podstron.

Celem pracy jest omówienie zagadnień związanych z Semantycznym Internetem i Danymi Połączonymi, przedstawienie i klasyfikacja różnych technik buforowania, a także prezentacja kompletnego magazynu grafów RDF.

Niniejsza praca składa się z części teoretycznej oraz praktycznej. W pierwszej z nich zostaną omówione zagadnienia dotyczące Semantycznego Internetu, Danych Połączonych, RDF jako języka opisu modelu danych, różnych serializacji RDF oraz koncepcji buforowania danych w bazach danych. W części drugiej, zostanie zapre-

[1] `http://www.ted.com/talks/tim_berners_lee_on_the_next_web` [dostęp: 2015-08-06]

zentowany kompletny magazyn grafów RDF – RTriples, zbudowany w oparciu o dokumentową bazę danych RethinkDB, przystosowaną do zastosowań związanych z Semantycznym Internetem i Danymi Połączonymi.

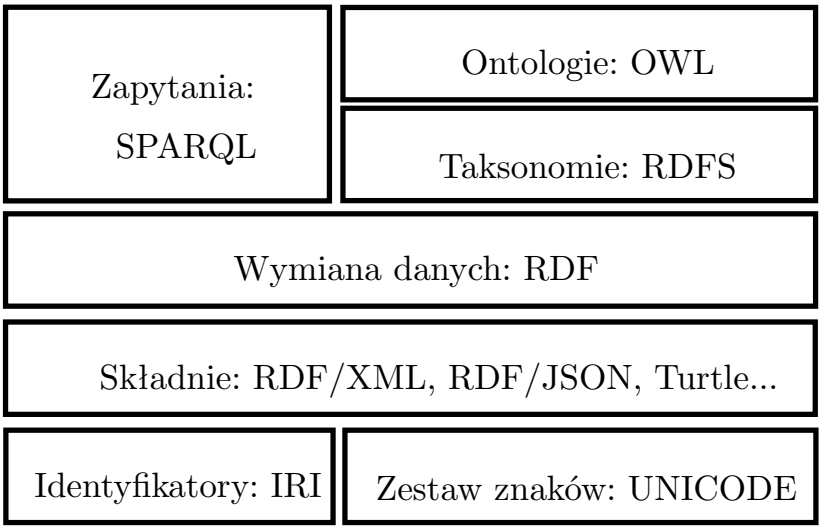
Praca została podzielona na trzy główne rozdziały. W rozdziale 1. zostanie omówiony szereg zagadnień związanych z Semantycznym Internetem takich jak: Stos Semantycznego Internetu, model RDF, graf RDF oraz magazyn grafów RDF. Rozdział 2. będzie poświęcony zagadnieniom buforowania danych. Zostanie tu również dokonana klasyfikacja strategii i algorytmów buforowania. Rozdział 3. przybliży natomiast kwestie związane z architekturą systemu testowego, danymi testowymi czy wynikami przeprowadzonych eksperymentów. Praca kończy się wnioskami i podsumowaniem.

# Rozdział 1

# Semantyczny Internet

Zacznijmy najpierw od omówienia czym są Semantyczny Internet (ang. Semantic Web) [6] oraz Dane Połączone (ang. Linked Data) [7].

**Definicja 1.1** (Semantyczny Internet)**.** *Projekt mający za zadanie stworzenie standardu opisu rzeczywistości (treści, osób, powiązań pomiędzy nimi itd.), tak aby te dane mogły być bezproblemowo przetwarzane przez maszyny wraz z zachowaniem odpowiedniego, łączącego te dane kontekstu.* □



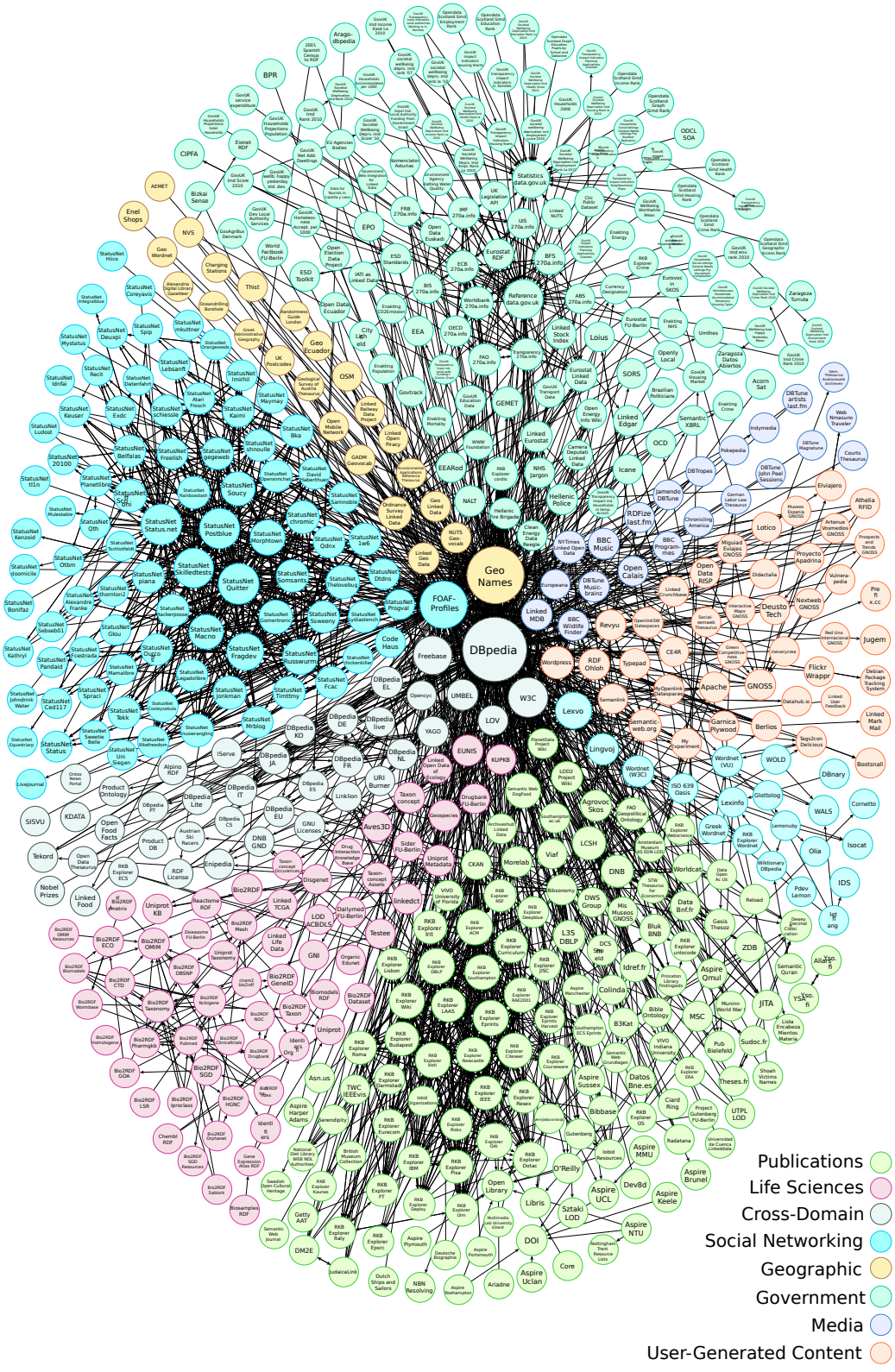

Rysunek 1.1: Stos Semantycznego Internetu

Stos Semantycznego Internetu (rysunek 1.1) ukazuje strukturę semantycznej sieci (definicja 1.1). Schemat ten bywa często nazywany potocznie jako torcik Semantycznego Internetu (ang. Semantic Web Cake) ze względu na to, że podobnie jak to danie, ma budowę warstwową. Poniżej zostaną omówione jego poszczególne elementy.

Stos Semantycznego Internetu [26] składa się z warstw. Każde kolejne piętro opiera się na elementach warstwy wcześniejszej. Na pierwszą warstwę składają się - identyfikator IRI [19] oraz zestaw znaków UNICODE [1]. Pojęcie identyfikatora IRI, zostanie omówione w podrozdziale 1.1. UNICODE to natomiast komputerowy zestaw znaków, umożliwiający reprezentowanie i przetwarzanie tekstu w wielu językach. Kolejny poziom stanowią składnie, zwane również serializacjami. W swojej pracy skupię się głównie na omówieniu RDF/JSON [49], jednak istnieje też wiele innych serializacji. Za przykładowe można wymienić: N-Triples [15], Turtle [37], TriG [43], N-Quads [14], RDFa [46] i inne. Piętro wyżej znajduje się RDF [17], który zostanie omówiony w podrozdziale 1.1.

Istotnym elementem jest SPARQL [23], będący jednym z najpopularniejszych języków zapytań RDF [50]. Jego cechą charakterystyczną jest składnia bardzo podobna do SQL pozwalająca na szereg operacji CRUD takich jak dodawanie, wybieranie, modyfikowanie oraz usuwanie zgromadzonych danych. Tuż obok języka zapytań, na rysunku 1.1, możemy dostrzec taksonomie RDFS (RDF Schema) i ontologie OWL. RDFS (RDF Schema) [9] rozszerza RDF o klasy i własności. Pozwala opisywać relacje pomiędzy powiązanymi zasobami. Zezwala on również na hierarchię klas. OWL (ang. Web Ontology Language) [3] rozszerza RDFS o bardziej zaawansowane konstrukcje, na przykład relację przechodniości, symetryczności czy inwersji. Wraz z OWL 2, oprócz relacji zwrotności i asymetryczności, została dodana również obsługa liczności.

**Definicja 1.2** (Dane połączone)**.** *Dane rozrzucone na kawałki, które jednocześnie są ze sobą połączone w pewnej relacji. Przeglądając informacje o konkretnym obiekcie, możliwe jest przejście do kolejnych zbiorów danych, które dostarczają większej ilości szczegółów.* □




Rysunek 1.2: Przykład chmury danych połączonych

Koncepcja Danych połączonych [24] jest nieco analogiczna do linków obecnych na stronach internetowych z tą różnicą, że „zwykłe" linki nie niosą ze sobą dodatkowych informacji (typów relacji), które mogłyby być bezproblemowo przetworzone przez maszyny. Dzięki Sematycznemu Internetowi i Danym Połączonym możliwe jest budowanie zapytań typu „urodziny prezydenta Stanów Zjednoczonych" czy „użytkownicy Internetu w Polsce". Tego typu zapytania, mogą być już poprawnie interpretowane przez maszyny. Przykładem może być Knowledge Graph firmy Google używany w wyszukiwarce i Google Now, będący osobistym asystentem. Innym rozwiązaniem jest Open Graph firmy Facebook, pozwalający programistom stron internetowych udostępniać dodatkowe informacje o danej witrynie internetowej.

## 1.1 Trójka RDF

Elementarne składniki modelu RDF odnoszą się do zasobów. Elementy te składają się z trzech rozłącznych podzbiorów:

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

Referencje IRI są globalnymi identyfikatorami, które mogą być używane do określania dowolnego zasobu.

**Przykład 1.1.** *Przykład prezentuje referencję IRI, która została użyta do określenia domu w DBpedii*[1].

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

Literały natomiast to wartości leksykalne. Dzielą się na czyste literały oraz literały z typem. Te pierwsze są zbiorem czystych ciągów znaków z opcjonalnym znacznikiem języka. Literały z typem składają się z leksykalnego ciągu znaków oraz typu danych, które są identyfikowane przez referencję IRI.

**Przykład 1.2.** *Przykład prezentuje literał prosty bez znacznika języka, literał prosty ze znacznikiem języka i literał z typem*

```
"Jabłko" "Jabłko"@pl
"3.14"^^http://www.w3.org/2001/XMLSchema#float
```

Ostatnim podzbiorem są węzły puste, które są zmiennymi stosowanymi do oznaczania pewnego zasobu, dla którego nie jest podany literał lub referencja IRI.

**Przykład 1.3.** *Przykład prezentuje węzeł pusty.*

```
_:x
```

Model danych RDF jest podobny do diagramów klas, jest oparty na idei tworzenia twierdzeń na temat zasobów internetowych w postaci zdań temat-predykat-obiekt. Zdania te w terminologii RDF nazwane są trójkami RDF. Temat określa opisywany zasób. Predykat określa cechy i aspekty zasobu oraz wyraża relację między tematem a obiektem. Natomiast obiekt przechowuje wartość tej relacji. Zgodnie ze specyfikacją języka RDF poniżej definiujemy trójkę RDF.

**Definicja 1.3** (Trójka RDF)**.** *Załóżmy, że $\mathcal{I}$ to zbiór wszystkich referencji IRI, $\mathcal{B}$ to nieskończony zbiór węzłów pustych. $\mathcal{L}$ jest zbiorem literałów, wtedy* trójka RDF *$t$ będzie zdefiniowana jako trójka $t = \langle s, p, o \rangle$, gdzie $s \in \mathcal{I} \cup \mathcal{B}$ jest nazywane tematem, $p \in \mathcal{I}$ jest nazywane predykatem i $o \in \mathcal{I} \cup \mathcal{B} \cup \mathcal{L}$ jest nazywane obiektem.* □

**Przykład 1.4.** *Przykład prezentuje trójkę RDF składającą się z tematu, predykatu i obiektu*

```
<http://example.net/me#jk>   foaf:name   "Jan Kowalski" .
```

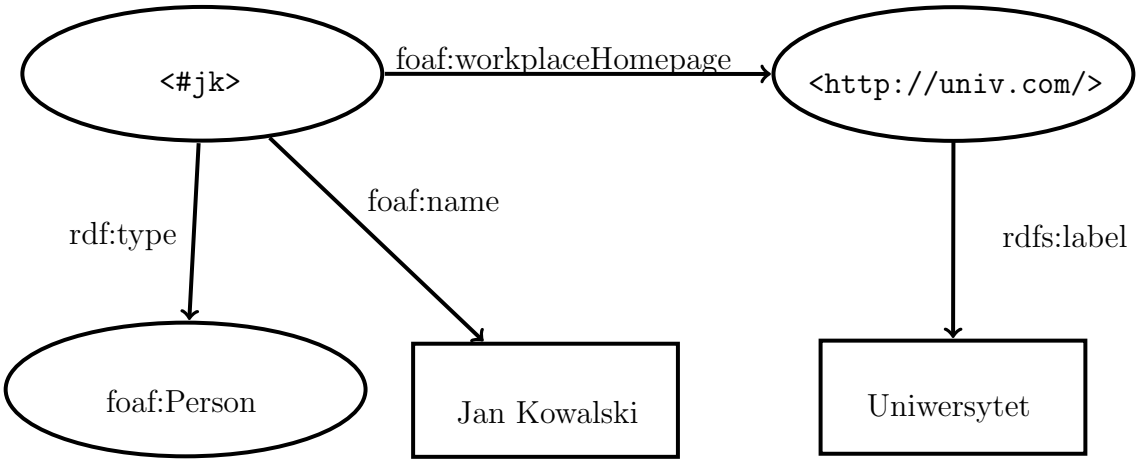
## 1.2 Graf RDF

Kolekcja trójek RDF stanowi graf RDF. Jest to multigraf skierowany z etykietami, gdzie węzłami są tematy i obiekty trójki RDF. Często graf RDF jest określany jako graf danych strukturalnych, gdzie trójka $\langle s, p, o \rangle$ może być widziana jako krawędź $s \xrightarrow{p} o$.

**Definicja 1.4** (Graf RDF)**.** *Niech* $\mathcal{O} = \mathcal{I} \cup \mathcal{B} \cup \mathcal{L}$ *i* $\mathcal{S} = \mathcal{I} \cup \mathcal{B}$*, wtedy* $G \subset \mathcal{S} \times \mathcal{I} \times \mathcal{O}$ *jest zbiorem wszystkich* trójek RDF *i nazywamy go* grafem RDF. □

**Przykład 1.5.** *Przykład prezentuje graf RDF opisujący profil FOAF [10]. Ten graf zawiera cztery trójki RDF:*

```
<#jk>   rdf:type    foaf:Person    .
<#jk>   foaf:name   "Jan Kowalski" .
<#jk>   foaf:workplaceHomepage <http://univ.com/> .
<http://univ.com/>  rdfs:label "Uniwersytet"      .
```



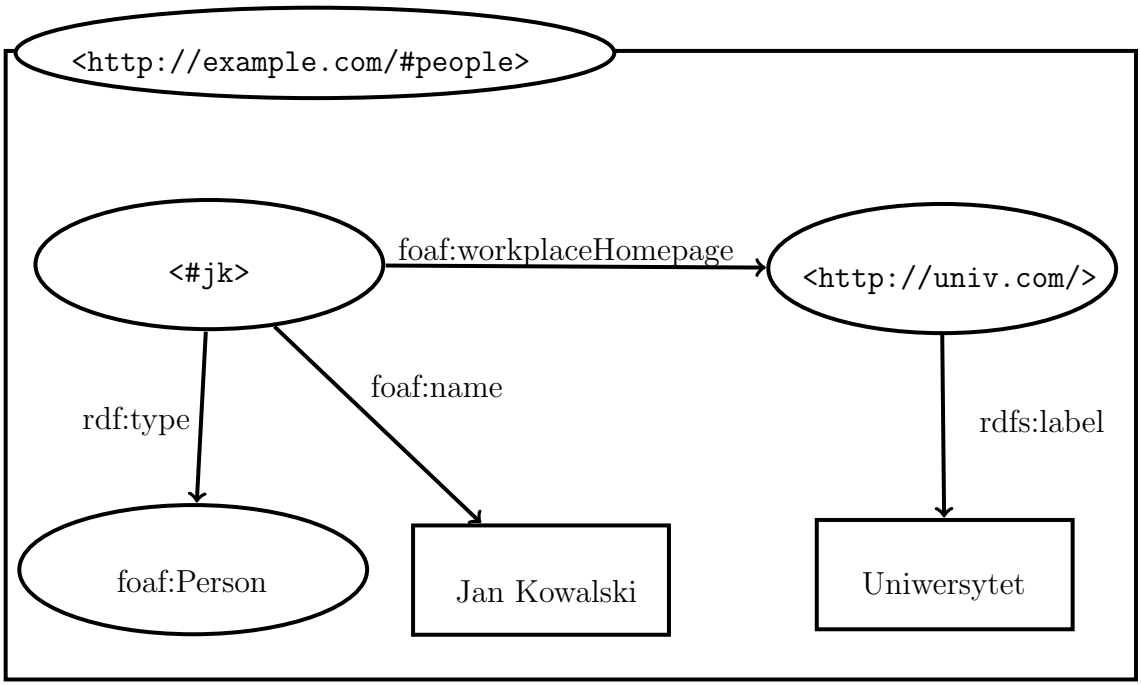

Rysunek 1.3: Graf RDF

Semantyka i składnia RDF może zostać rozszerzona o koncepcję grafów nazwanych. Podstawą modelu danych grafów nazwanych jest mechanizm nazywania grafów.

**Definicja 1.5** (Graf nazwany)**.** Graf nazwany *NG to para* $\langle u, G \rangle$*, gdzie* $u \in \mathcal{I} \cup \mathcal{B}$ *i jest nazwą grafu a G jest grafem RDF.* □

**Przykład 1.6.** *Przykład prezentuje graf nazwany zawierający profil FOAF. Graf ten posiada nazwę* `http://example.com/#people` *i zawiera cztery trójki RDF:*

```
<http://example.com/#people> {
  <#jk>   rdf:type    foaf:Person    .
  <#jk>   foaf:name   "Jan Kowalski" .
  <#jk>   foaf:workplaceHomepage <http://univ.com/> .
  <http://univ.com/>  rdfs:label "Uniwersytet"      .
}
```




Rysunek 1.4: Graf nazwany

Jedną z serializacji języka RDF obsługującą grafy nazwane jest RDF/JSON [48]. Składnia ta jest przeznaczona do łatwej integracji z wdrożonymi systemami wykorzystującymi język JSON. Publikowanie danych RDF w tym formacie sprawia, iż są

one dostępne dla twórców internetowych bez potrzeby instalowania dodatkowych bibliotek i innego oprogramowania do modyfikacji dokumentów. Przykład serializacji RDF/JSON znajduje się w przykładzie 1.7.

**Przykład 1.7.** *Przykład prezentuje serializację RDF/JSON*[2] *tożsamą z przykładem 1.4.*

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

Klucze `type` i `value` są obowiązkowe. Pierwszy z nich określa typ wartości, a drugi przyjmowaną wartość. Obsługiwane wartości typów danych to `uri`, `bnode`, `literal` oraz `typed-literal`. Dodatkowe klucze to `lang`, określający język oraz `datatype` określający typ. Pierwszy z nich może występować tylko, gdy klucz `type` został określony jako literał, a drugi – tylko w przypadku `typed-literal`. Serializacja pozwala na wzbogacenie trójki RDF (definicja 1.3) o kontekst za pomocą pola `context`.

[2] `uri` traktowany jest jako IRI

Poniższa tabela przedstawia reguły użycia poszczególnych typów danych w każdym z pól:

| typ | temat | predykat | obiekt | kontekst |
|---|---|---|---|---|
| referencja IRI (`uri`) | tak | tak | tak | tak |
| węzeł pusty (`bnode`) | tak | nie | tak | nie |
| literał (`literal`) | nie | nie | tak | nie |

Tablica 1.1: Zestawienie typów danych w serializacji RDF/JSON

## 1.3 Magazyn grafów RDF

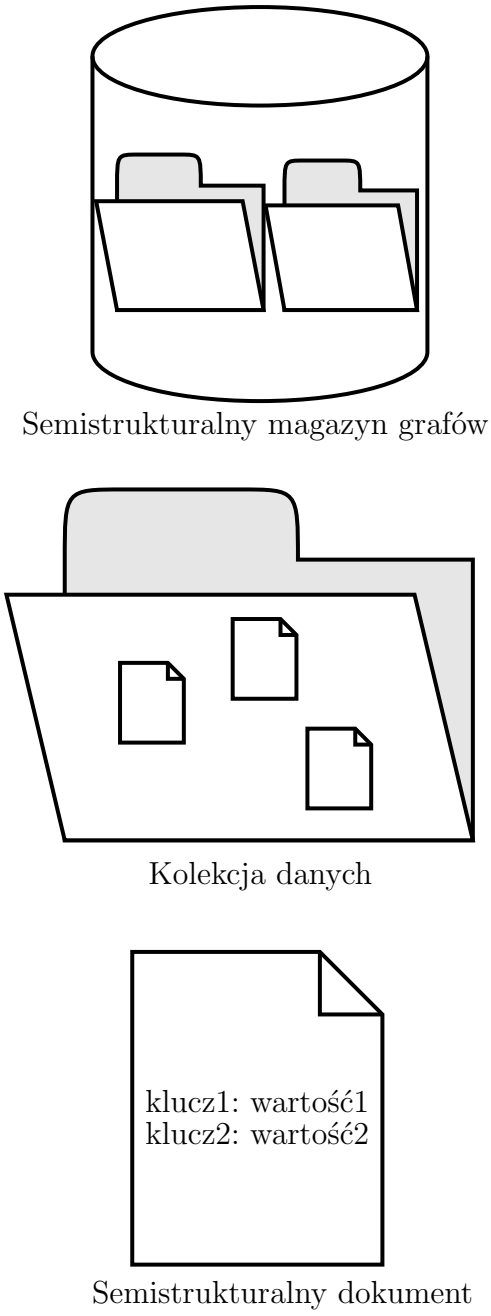
Proponowany semistrukturalny magazyn grafów RDF składa się z kolekcji danych, które zawierają w sobie semistrukturalne dokumenty (rysunek 1.5).

**Definicja 1.6** (Semistrukturalny dokument)**.** *Załóżmy, że para klucz/wartość to* $P = \langle k, v \rangle$*, gdzie* $k \in \mathcal{L}$ *oznacza klucz, a* $v$ *oznacza wartość klucza, który może zawierać wartość logiczną, liczbową, ciąg znaków lub kolejną parę klucz/wartość.* Semistrukturalny dokument $D$ *możemy zdefiniować jako* $D = \{P_1, P_2, \ldots, P_n\}$ *dla* $n \geqslant 1$. □

Zakładamy również, że semistrukturalny dokument reprezentuje dokładnie jeden graf RDF $G$ oraz może zawierać nazwane grafy $NG$.

**Definicja 1.7** (Kolekcja danych)**.** Kolekcja danych *to krotka* $C = \langle id, [D_1, D_2, \ldots, D_i] \rangle$*, gdzie* $id \in \mathcal{L}$ *to nazwa kolekcji, a* $[D_1, D_2, \ldots, D_i]$ *to lista semistrukturalnych dokumentów.* $i \geqslant 1$. □

**Definicja 1.8** (Semistrukturalny magazyn grafów RDF)**.** Semistrukturalny magazyn grafów RDF *to* $GS = \{C_1, C_2, \ldots, C_m\}$*, gdzie każde* $C_m$ *to kolekcje danych,* $m \geqslant 0$. □




Rysunek 1.5: Relacja pomiędzy semistrukturalnym dokumentem, kolekcją danych, a semistrukturalnym magazynem grafów RDF

## 1.4 Prace powiązane

Obecne prace są skupione głównie na znalezieniu jak najlepszego sposobu składowania i przetwarzania danych w języku RDF. Jak przyznają autorzy w [29], RDF może

być z powodzeniem stosowany do zapisywania danych semistrukturalnych w różnego typu aplikacjach, jednakże efektywne przetwarzanie tych informacji, możliwe jest dopiero w chmurze obliczeniowej. Kolejny artykuł [16] dokonuje porównania różnych baz danych NoSQL przetwarzających dane RDF. Autorzy zwracają uwagę na to, że bazy NoSQL mają spory potencjał w przetwarzaniu danych RDF, ale zauważają jednak problemy związane z optymalizacją zapytań.

W artykule [49] można natomiast przeczytać o wykorzystaniu dokumentowej bazy danych MongoDB[3] jako magazynu grafów RDF. Autor przedstawia również przykładowe algorytmy mapowania zapytań SPARQL do używanej serializacji RDF/JSON (przykład 1.7).

Autorzy pracy [38] prezentują ciekawe połączenie nierelacyjnej, rozproszonej bazy danych NoSQL, wzorowanej na Google BigTable – Apache HBase z techniką MapReduce [18]. Inną propozycją jest Rya [39] – skalowalna baza danych RDF używająca Accumulo[4] zbudowana na podstawie Google Bigtable. Implementacja wymienionej bazy jako wtyczki do OpenRDF Sesame [12] pozwala na obsługę zapytań SPARQL oraz różnych serializacji RDF.

Na skalowalność nacisk kładą również autorzy [30] oraz [38]. Pierwsi prezentują bazujący na HBase[5] magazyn trójek Jena-HBase, a drudzy – PigSPARQL, czyli system, który daje możliwość przetwarzania złożonych zapytań SPARQL w klastrze MapReduce. Artykuł [41] prezentuje natomiast podejście rozwiązujące problemy dopasowywania wzorców grafu w technice MapReduce przy użyciu zaawansowanej algebry.

Autorzy [22] proponują rozproszony system oceny/optymalizacji zapytań na dużych ilościach danych RDF – Partout oraz algorytmy podziału danych na części. Hose i Schenkel [27] w swojej pracy prezentują natomiast sposób na równoważenie obciążenia bazujący na partycjonowaniu danych i replikowaniu trójek. Kolejnym godnym uwagi pomysłem jest Trinity.RDF [56] – rozproszona, skalowalna baza danych RDF. Rozwiązanie to nie opiera się na operacji łączenia. Innym podejściem wykazali się autorzy [35]. Przedstawiają oni opracowanie specjalnego algorytmu, który pozwala na optymalizowanie złączenia w bazie danych H2RDF korzystającej z MapReduce.

W pracy [42] zaprezentowano natomiast grafową bazę danych SHARD zbudo-

[3] `https://www.mongodb.org/` [dostęp: 2015-07-10]

[4] `https://accumulo.apache.org/` [dostęp: 2014-08-16]

[5] `http://hbase.apache.org/` [dostęp: 2014-08-11]

waną na podstawie Apache Hadoop[6], gdzie specjalne algorytmy iteracyjne stanowią podstawę skalowalności całego systemu. Szereg technik pozwalających na zapewnienie skalowalności zapytań SPARQL przy dużych grafach RDF zaproponowano natomiast w pracy [28]. Jednym z pomysłów jest podział zapytań SPARQL na wysokowydajnościowe części, które wykorzystują to, w jaki sposób dane są podzielone w klastrze. Inny sposób to umieszczenie danych pomiędzy węzłami w sposób, który pomaga przyspieszyć przetwarzanie zapytań poprzez lokalne optymalizacje.

Autorzy w [2] prezentują platformę Amanda opartą o Amazon Web Services (AWS) i podejściu Software as a Service (SaaS). Proponowane rozwiązanie pozwala na przechowywanie danych w sieci, w szczególności dokumentów XML i grafów RDF. Artykuł pokazują też jak zbudować i wykonywać operacje na ich magazynie grafów RDF. W artykule [13] zbadano natomiast problem indeksowania danych RDF przechowywanych w chmurze Amazon. Użyto w tym celu dostarczonej przez AWS dla niedużych danych, bazy danych SimpleDB [40]. Na tym samym oprogramowaniu skupili się również autorzy [47]. Zaproponowali oni Stratustore, który działa jako back-end dla Jena Semantic Web framework [33] i przechowuje dane w SimpleDB. Inną bazą klucz-wartość jest bazująca na Apache Cassandra[7] – CumulusRDF prezentowana w [32]. Autorzy opracowali dwa schematy indeksu dla RDF w celu wsparcia wyszukiwania Danych Połączonych oraz podstawowego wyszukiwania trójek według wzorca.

[6] `http://hadoop.apache.org/` [dostęp: 2014-08-17]

[7] `http://cassandra.apache.org/` [dostęp: 2014-08-16]

# Rozdział 2

# Buforowanie danych

Na początku przyjrzyjmy się czym tak właściwie są dane. Definicji tego terminu mogłoby się znaleźć wiele. Możemy śmiało przyjąć, że:

**Definicja 2.1** (Dane)**.** Dane *to wszystkie pojedyncze elementy otaczającego nas świata, które można w jakikolwiek sposób przetworzyć [21].* □

Dane mogą być połączone ze sobą w relacje, a zbiór danych nazwiemy informacją. Zwykle mamy do czynienia z większą ilością danych. W takim przypadku, ważnym aspektem może być buforowanie danych.

## 2.1 Buforowanie w bazach danych

Dane, a właściwie zbiory danych mogą być gromadzone w bazach danych.

**Definicja 2.2** (Baza danych)**.** *Niech* $\mathcal{D}_0, \mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3 \ldots \mathcal{D}_i$*. Baza danych to* $DB = \{\mathcal{D}_0, \mathcal{D}_1, \ldots, \mathcal{D}_i\}$*, gdzie* $i \geqslant 0$*, to pojedyncze jednostki danych.* □

Umieszczenie danych w bazach danych pozwala na sprawniejszą ich późniejszą obróbkę (przetwarzanie).

**Definicja 2.3** (System zarządzania bazami danych)**.** System zarządzania bazami danych*, zwany też systemem bazodanowym, to program komputerowy (jeden lub więcej) pozwalający na zarządzanie bazą danych. Działa on zazwyczaj w architekturze klient-serwer.* □

**Definicja 2.4** (Buforowanie danych)**.** Buforowanie danych *to zbiór metod pozwalających ograniczyć powtórne przetwarzanie tych samych danych m.in. w celu przyspieszenia przetwarzania danych oraz zmniejszenia obciążenia sprzętu, a także infrastruktury sieciowej.* □

W przypadku systemów zarządzania bazami danych, które muszą obsługiwać dużą liczbę zapytań, wykorzystanie jakichkolwiek metod buforowania staje się koniecznością. Dzięki temu wzrasta responsywność i zmniejsza się zużycie zasobów. Za bardziej konkretny przykład, możemy przytoczyć skrócenie czasu zwrócenia wyniku zapytania, jeżeli dana kwerenda była już wcześniej uruchomiona. Buforowanie ma też swoje wady. Pierwszą z nich jest to, że potrzebna jest dodatkowa przestrzeń dyskowa i odpowiednie struktury pozwalające na zapisanie już przetworzonych danych. Dodatkowo, jeżeli dane w bazie danych ulegną zmianie, przez pewien określony czas system bazodanowy może zwracać starsze, nieaktualne informacje.

## 2.2 Strategie i algorytmy buforowania

| algorytmy zmiany strony | buforowanie dokumentów HTTP |
|---|---|
| Last In, First Out (LIFO) | ETag |
| First In, First Out (FIFO) | Expires |
| Least Recently Used (LRU) | Cache-Control |
| Most Recently Used (MRU) | Server Push |
| Least Frequently Used (LFU) | |
| Random Replacement (RR) | |

Tablica 2.1: Zestawienie przykładowych algorytmów buforowania

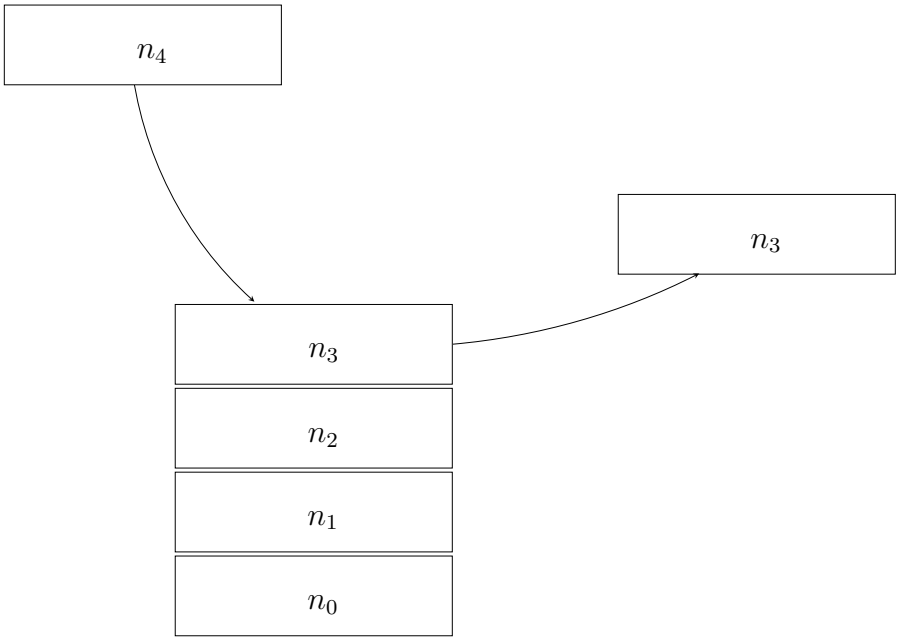
Poniżej zostaną omówione przykładowe algorytmy buforowania.

### 2.2.1 Algorytmy zmiany strony

#### Last In, First Out (LIFO)

LIFO (ang. Last In, First Out co możemy przetłumaczyć jako „ostatni wchodzi, pierwszy wychodzi" lub „ostatni będą pierwszymi" [55]) to porządek, w którym dostęp do danych możliwy jest zaczynając od ostatniego elementu, a kończąc na pierwszym. Przykładem struktury danych zgodnej z takim podejściem jest stos.

**Definicja 2.5** (Stos)**.** Stos *to struktura danych* $S = \{n_0, n_1, n_2, \ldots n_i\}$*, taka, że bezpośredni dostęp do danych możliwy jest wyłącznie poczynając od* $n_i$*, a kończąc na* $n_0$ *elemencie.* □



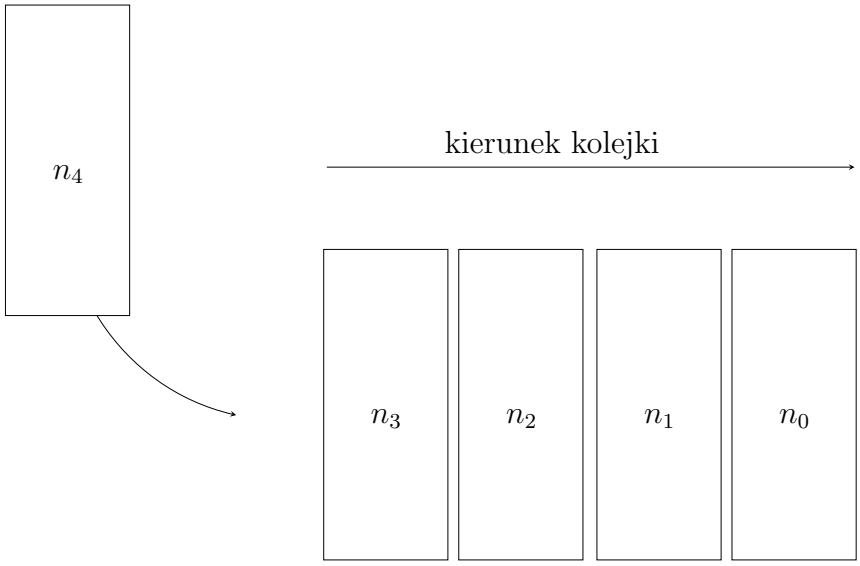

Rysunek 2.1: Stos

Najczęstszym przykładem obrazującym pojęcie stosu, są książki ułożone jedna po drugiej w wieżę pnącą się do góry. Widzimy boki każdej z książek. Bez żadnego problemu mamy też pełen dostęp do pozycji znajdującej się na samej górze. W przypadku książek znajdujących się niżej, aby się z nimi kompletnie zapoznać, musimy najpierw zdjąć te umieszczone nad nimi. Podstawowymi operacjami, jakie możemy wykonywać na stosie są: dodanie nowego elementu (`push`) oraz pobranie (`pop`) konkretnego elementu ze stosu.

### First In, First Out (FIFO)

FIFO (ang. First In, First Out co możemy przetłumaczyć jako „pierwszy wchodzi, pierwszy wychodzi" lub „kto pierwszy, ten lepszy") to porządek, w którym najpierw obsłużone zostają dane, które pojawiły się w buforze jako pierwsze, a na końcu te, które zostały umieszczone tam jako ostatnie. Przykładem struktury danych realizującej to podejście jest kolejka.

**Definicja 2.6** (Kolejka)**.** Kolejka *to struktura danych* $Q = \{n_0, n_1, n_2, \ldots n_i\}$*, gdzie odczytywanie danych rozpoczyna się od elementu* $n_0$*, zaś dodawanie nowych informacji bezpośrednio przed elementem* $n_i$*.* □




Rysunek 2.2: Kolejka

Tę strukturę danych w najprostszym przypadku możemy porównać do kolejki klientów do kasy sklepowej. Pierwsza zostanie obsłużona osoba, która przyszła najpierw, następnie ta będąca za nią i tak dalej. Wchodzący do sklepu klienci ustawiają się za ostatnią stojącą osobą. Specyficznym rodzajem kolejki jest kolejka priorytetowa, gdzie poszczególnym danym mogą być przydzielane priorytety. Kolejność opuszczenia kolejki zależy w tym przypadku nie tylko od umieszczenia danych, ale też od przypisanych im wartości. Podstawowymi operacjami na kolejce są: dodanie nowych danych na ogon oraz usunięcie danych z czoła kolejki.

### Least Recently Used (LRU)

Cechą charakterystyczną algorytmu LRU (Least Recently Used) jest to, że w przypadku kończenia się pamięci buforu, dane najrzadziej używanych pozycji są usuwane.

Implementację tego algorytmu można przeprowadzić na wiele sposobów. Jednym z nich jest dodanie liczników – globalnego oraz unikalnego dla każdego elementu. Przy każdym wywołaniu rekordu, globalny licznik jest zwiększany o 1, a jego wartość jest również kopiowana do unikalnego licznika danego elementu. Do zastąpienia wybierany jest zatem element z najniższą wartością licznika.

Innym sposobem implementacji może być użycie listy dwukierunkowej. Przy odwołaniu do danego rekordu, informacje o tym elemencie usuwa się z miejsca w którym się znajduje (jeżeli został wcześniej dodany do listy) i umieszcza na przodzie. Na początek trafia każdy nowy, ostatnio używany element. Do usunięcia wybierana

jest ostatnia pozycja.
Algorytm w pseudokodzie obrazujący zachowanie się LRU w przypadku skończenia się pamięci buforu, mógłby wyglądać następująco:

```
1 wczytaj pobrany element
2 if element jest już na liście then
3 |   znajdź element na liście
4 |   przenieś element na początek listy
5 else
6 |   usuń element z końca listy
7 |   umieść nowy element na początku listy
```

### Most Recently Used (MRU)

Algorytm ten jest podobny do przytoczonego wcześniej z tą różnicą, że tym razem usuwane są najpierw najczęściej wywoływane elementy.

### Least Frequently Used (LFU)

Algorytm polega na zapisywaniu ilości zapytań do poszczególnych elementów. W przypadku osiągnięcia limitu pamięci bufora, usuwane są te z najmniejszą ilością zapytań. Algorytm ten sprawdza się dobrze w sytuacjach, gdzie odstępy czasowe pomiędzy wywołaniami poszczególnych rekordów są mniej-więcej podobne.

### Random Replacement (RR)

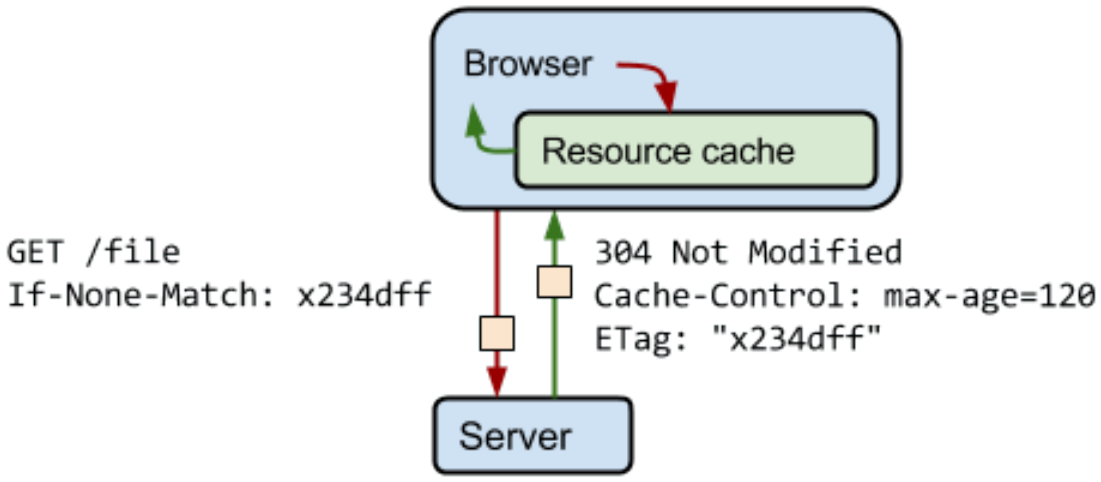
Dla wyżej zaprezentowanych algorytmów, wybór elementów do odrzucenia w przypadku zapełnienia się bufora następuje poprzez spełnienie jakiegoś warunku. Niestety, mimo, że to rozwiązanie wydaje się nam bardziej logiczne, czasem jego praktyczne wykorzystanie wiąże się z pewnymi problemami. Porównywanie czy składowanie informacji o elementach również zajmuje zasoby i czas. Najbardziej trywialnym rozwiązaniem wyboru elementu do odrzucenia staje się więc wybór losowy. Cechuje go prosta implementacja i brak konieczności przechowywania informacji o elementach.

### 2.2.2 Buforowanie dokumentów w protokole HTTP

#### ETag

Użycie mechanizmu `ETag` [20] pozwala na uniknięcie pobierania tych samych danych. Rozwiązanie to działa nawet wtedy, gdy dane w pamięci podręcznej już wygasły. Algorytm zostanie przedstawiony w następującym pseudokodzie:

```
1  if zasób zmieniony then
2      Serwer generuje token danego zasobu
3      Serwer zwraca token danego zasobu
4  Klient wysyła zapytanie o zasób z podaniem dotychczasowego tokena
5  if token wysłany przez klienta != token serwera then
6      Serwer zwraca odpowiedź 304 Not Modified
7      Klient serwuje zasób z pamięci podręcznej
8  else
9      Klient pobiera zmieniony zasób
10     Klient serwuje zmieniony zasób
```




Rysunek 2.3: Ilustracja działania mechanizmu `ETag` (autor grafiki: Ilya Grigorik, licencja: CC BY)

Warto zauważyć, że klient nie musi znać sposobu generacji tokena. Tym zajmuje się serwer.

### Expires

Nagłówek HTTP `Expires` to chyba najbardziej trywialny sposób zarządzania buforowaniem danych. Polecenie to, wraz z dokładną datą określa, kiedy wskazany zasób będzie mógł być uważany za przestarzały.
Składnia jest następująca:

```
Expires = "Expires" ":" HTTP-date
```

**Przykład 2.1.** *Poniższy przykład przedstawia poprawne ustawienie nagłówka Expires dla danych, które mają wygasnąć 7 czerwca 2007 roku o godzinie 19:00:05 GMT (zerowa strefa czasowa).*

```
Expires: Thu, 07 Jun 2007 19:00:05 GMT
```

### Cache-Control

Za pomocą dyrektyw cache-control można wygodnie kontrolować sposób buforowania poszczególnych typów zasobów. Zostaną omówione najpopularniejsze z nich. Dyrektywa `no-cache` pozwala na możliwość przechowywania zwróconej odpowiedzi pod warunkiem, że ta się nie zmieniła. Odpowiedź oznaczona jako `private` oznacza natomiast, że jest ona raczej przeznaczona dla konkretnego użytkownika. Wobec tego można ją buforować, ale nie w pośrednich pamięciach podręcznych, a w przeglądarce użytkownika. Poprzez `public` rozumie się, że daną odpowiedź można buforować w dowolnej pamięci podręcznej, nawet jeżeli inne ustawienia wskazują na co innego. W celu całkowitego zabronienia przeglądarce przechowywania danych, należy użyć `no-store`. Wynikiem tego będzie to, że zasób zawsze będzie pobierany z serwera i nie będą przechowywane jego kopie w pamięci podręcznej. Może to być istotne przy bardziej prywatnych lub często zmieniających się danych. W celu określenia długości czasu przechowywania danych w pamięci podręcznej, możemy użyć `max-age` z odpowiednią wartością wyrażoną w sekundach.

**Przykład 2.2.** *Poniższa dyrektywa określa maksymalny czas przetrzymywania zbuforowanej kopii odpowiedzi. Po przekroczeniu tego czasu, kolejne żądanie danego zasobu, spowoduje pobranie jego nowej wersji.*

```
Cache-Control: max-age=120
```

Nieco innym podejściem cechuje się dyrektywa `min-fresh`, również przyjmująca wartość wyrażoną w sekundach. Możemy tutaj określić czas, w którym dany zasób jest uważany za świeży (ang. fresh). W tym czasie, klientowi serwowana jest jego kopia podręczna. Po upłynięciu podanej liczby sekund, zasób jest pobierany z serwera.

#### Server Push

Technika Server Push [5], która pojawiła się wraz z HTTP/2, pozwala na przesyłanie danych bezpośrednio do pamięci podręcznej klienta. Ogranicza to ilość żądań spływających do serwera, gdyż odbiorca nie musi prosić o konkretne zasoby. Wiąże się też z redukcją czasu renderowania strony, szczególnie na łączach o dużym opóźnieniu. Istnieje możliwość wyłączenia mechanizmu Server Push przez klienta za pomocą parametru `SETTINGS_ENABLE_PUSH` ustawionego na wartość `0`.

## 2.3 Prace powiązane

Prędkość dostępu i przetwarzania danych jest niezwykle ważnym aspektem każdej współczesnej aplikacji. Ilość danych i częstotliwość ich zmian ciągle się zwiększa. Dlatego potrzebne jest opracowanie różnego rodzaju metod mające na celu przyspieszenie przetwarzania danych oraz odciążenie serwerów. Zarówno czas jak i pamięciożerność poszczególnych rozwiązań ma istotne znaczenie.

W artykule [4] zauważono, że systemy buforowania wykorzystane w sieci Internet, mogą prowadzić do zmniejszenie ilości przesyłanych danych, równoważenia obciążenia serwerów, a także m.in wyższej dostępności treści.

W pracy [54] opisano zmiany poczynione w Jena2 względem Jena1, czyli otwartoźródłowym frameworku du budowy Semantycznego Internetu i Danych Połączonych. Autorzy zwracają sporą uwagę na optymalizację zapytań RDF.

W artykule [45] zaproponowano nowe podejście przyspieszające przetwarzanie zapytań SPARQL w dużych magazynach czwórek RDF. Zastosowano strategię „dziel

i zwyciężaj", dzięki której możliwe stało się indeksowanie grup podobnych do siebie grafów RDF, zamiast całego zbioru danych. To, oraz inteligentne zawężanie grupy wyników, które mogą spełniać żądane zapytanie, pozwoliło na sprawne przetwarzanie zapytań na zbiorach liczących ponad miliard czwórek.

W artykule [11] zaproponowano własny format przechowywania czwórek RDF zwany RDFCSA. Jest to integralne połączenie indeksu z danymi. Dzięki temu, tak przygotowane trójki zajmują mniej miejsca niż ich „zwykłe" odpowiedniki, a zapytania wykonują się odpowiednio szybciej.

Nie zawsze skomplikowane rozwiązania są potrzebne. Autor [31] zauważa, że już samo wykorzystanie odpowiednich nagłówków buforowania HTTP, pozwala na spore odciążenie serwerów. Niestety, zarówno w przypadku zwykłych stron internetowych, jak i wyspecjalizowanych rozwiązań Semantycznego Internetu, zdarzają się błędy związanie z niepoprawnym lub nawet brakiem użycia tego typu znaczników. W artykule dokonano przeglądu nagłówków HTTP, a także zalecono zwiększenie ich wykorzystania.

W pracy [51] przeanalizowano rezultaty zapytań z dwóch publicznych baz danych SPARQL z aktualnymi wersjami źródeł. Okazało się, że zwracane wyniki często były nieaktualne lub występowały jakieś braki. Autorzy zaproponowali hybrydowe rozwiązanie, które łączy najlepsze cechy znane zarówno z silników Danych Połączonych z indeksami, jak i tych, które wykonują zapytania „na żywo". Pierwsze z nich pozwalają uzyskać wynik bardzo szybko, ale dane często bywają nieaktualne, natomiast drugie – zapewniają świeżość danych kosztem długiego czasu reakcji. Badacze postanowili podzielić pojedyncze zapytania na podzapytania, gdzie część danych mogła być zwracana od razu. Autorzy zwracają uwagę, że w niektórych przypadkach planowanie podziału na podzapytania może być dość złożone. Mimo to pokazane rozwiązanie pozwala na pewien kompromis pomiędzy agresywnymi mechanizmami buforowania, wiążącymi się z większym prawdopodobieństwem nieaktualności danych, a rozwiązaniami zwracającymi odpowiedzi na zapytania w locie, ale cechującymi się dłuższymi opóźnieniami.

Bardziej tradycyjne algorytmy stronicowania wciąż bardzo dobrze sprawdzają się w wielu zastosowaniach. W pracy [36] możemy zapoznać się z algorytmów stronicowania, takimi jak różne wariacje Least Recently Used (LRU) czy Least Frequently Used (LFU).

Autorzy w [34] zaprezentowali jednak nieco odmienne podejście. Zauważyli oni, że internauci najczęściej przeglądają sieć za pomocą kolejnych odnośników umiesz-

czonych na stronach internetowych, które odwiedzają. Zamiast więc w pierwszej kolejności usuwać z pamięci podręcznej elementy, do których dostęp odbył się najdawniej lub najczęściej, można postąpić inaczej. Stworzony system (SACS), mierzy odległość między obiektami pod względem liczby połączeń niezbędnych do poruszania się z jednego do drugiego. Nietrudno się domyśleć, że pierwszymi do odrzucenia, będą obiekty najbardziej oddalone od najczęściej odwiedzanych stron. Im dany element znajduje się bliżej, tym prawdopodobieństwo jego usunięcia jest mniejsze.

W pracy [53] skupiono się natomiast na rozwiązaniu nieco bardziej sprzętowym. Aby przyspieszyć czas potrzebny na złączenia, wykorzystano układ typu Field-Programmable Gate Array (FPGA) na którym zostały zaimplementowane specjalne algorytmy. Pozwoliło to uzyskać nawet 10-krotne przyspieszenie.

# Rozdział 3

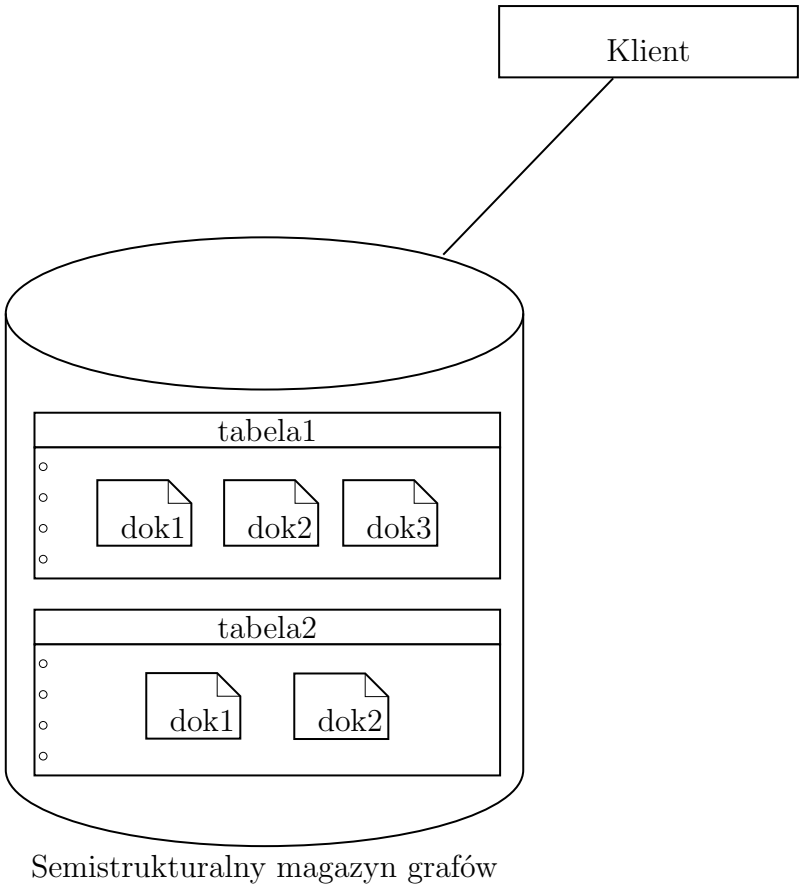
# Algorytmy buforowania w magazynie grafów RDF

## 3.1 Architektura systemu testowego

Architektura środowiska testowego została zaprezentowana na rysunku 3.1. Do budowy semistrukturalnego magazynu grafów RDF została użyta baza danych NoSQL RethinkDB[1]. Klient został napisany w języku Ruby z wykorzystaniem mikroframeworka Sinatra[2] oraz ERB jako systemu szablonów. Eksperymenty zostały wykonanie na komputerze wyposażonym w procesor Intel Core 2 Duo P7350 z dwoma rdzeniami taktowanymi 2.00 GHz każdy, 4GB pracującej w trybie Dual Channel pamięci RAM oraz 1TB dyskiem 5400 RPM.

[1] `http://rethinkdb.com/` [dostęp: 2015-08-11]

[2] `http://www.sinatrarb.com/` [dostęp: 2015-08-15]




Rysunek 3.1: Architektura środowiska testowego

Prezentowane rozwiązanie obsługuje 5 formatów danych wyjściowych:

**Przykład 3.1.** *Przykład prezentuje dane w formacie HyperText Markup Language (HTML)[25]:*

| Subject | Predicate | Object |
|---|---|---|
| 1. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer1125<br>2. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer321<br>3. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer1/Product26<br>4. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer599<br>5. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer1925<br>6. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/ProductType4<br>7. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromRatingSite1/Reviewer50<br>8. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer961<br>9. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromRatingSite1/Review972<br>10. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer599 | 1. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/vendor<br>2. http://www.w3.org/1999/02/22-rdf-syntax-ns#type<br>3. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/product<br>4. http://purl.org/dc/elements/1.1/publisher<br>5. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/offerWebpage<br>6. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/productFeature<br>7. http://www.w3.org/1999/02/22-rdf-syntax-ns#type<br>8. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/vendor<br>9. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/offerWebpage<br>10. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/offerWebpage | 1. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Vendor1<br>2. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/Offe<br>3. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product8<br>4. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromRatingSite1/RatingS<br>5. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer321/<br>6. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/ProductFeature324<br>7. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/ProductType<br>8. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Vendor1<br>9. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer290/<br>10. http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer1466/ |

**Przykład 3.2.** *Przykład prezentuje dane wyjściowe w serializacji RDF/JSON.*


```
[
        {
          "object": {
            "type": "uri",
            "value": "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Vendor1"
          },
          "predicate": {
            "type": "uri",
            "value": "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/vendor"
          },
          "subject": {
            "type": "uri",
            "value": "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer1125"
          }
        },
        {
          "object": {
            "type": "uri",
            "value": "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/Offer"
          },
          "predicate": {
            "type": "uri",
            "value": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type"
          },
          "subject": {
            "type": "uri",
            "value": "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer599"
          }
        },
        {
          "object": {
            "type": "uri",
            "value": "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product85"
          },
          "predicate": {
            "type": "uri",
            "value": "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/product"
          },
          "subject": {
            "type": "uri",
            "value": "http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer1925"
          }
        }
]
```

**Przykład 3.3.** *Przykład prezentuje dane w serializacji N-Triples.*

```
<http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer1125>
<http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/vendor>
<http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Vendor1> .
<http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer599>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/Offer> .
<http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer1925>
<http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/product>
<http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product85> .
```

**Przykład 3.4.** *Przykład prezentuje dane wyjściowe w formacie Comma-Separated Values (CSV) [44].*

```
subject,predicate,object
http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer1125,
http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/vendor,
http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Vendor1
http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer599,
http://www.w3.org/1999/02/22-rdf-syntax-ns#type,
http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/Offer
http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer1925,
http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/product,
http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product85
```

**Przykład 3.5.** *Przykład prezentuje dane wyjściowe w formacie tekstowym.*

```
--- SUBJECTS ---
http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer1125
http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer599
http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Offer1925

--- PREDICATES ---
http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/vendor
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/product

--- OBJECTS ---
http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromVendor1/Vendor1
http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/vocabulary/Offer
http://www4.wiwiss.fu-berlin.de/bizer/bsbm/v01/instances/dataFromProducer2/Product85
```

Warto zauważyć, że obsługa danych wyjściowych w formacie RDF/JSON (przykład 3.2), stwarza możliwości ponownego filtrowania wyników na różne sposoby, a

więc w pewnym sensie umożliwia tworzenie podzapytań. Aby tego dokonać, wystarczy załadować wygenerowany zbiór do bazy, a następnie przeprowadzić potrzebne operacje.

## 3.2 Zbiór danych testowych

BSBM skupia się na przypadkach użycia z e-commerce, gdzie jest oferowany zestaw produktów przez różnych producentów dla różnych konsumentów. Konsumenci mogą też pisać opinie o produktach. Benchmark określa abstrakcyjny model danych wraz z zasadami tworzenia tych danych, które mogą być skalowane do dowolnych rozmiarów, gdzie czynnikiem skali jest liczba produktów. BSBM obsługuje różne serializacje RDF, my wzbogaciliśmy go o serializację JSON-LD.

Model danych testowych zawiera następujące klasy *Product*, *ProductType*, *ProductFeature*, *Producer*, *Vendor*, *Offer*, *Review* oraz *Person*. Najważniejszym czynnikiem jest produkt (*Product*), który jest opisany właściwościami RDFS *label* i *comment*. Produkt zawiera własności tekstowe z przedziału $[3, 5]$. Własności te mają rankingi z przedziału $[1, 2000]$ z rozkładu normalnego.

Produkt ma typ, który jest częścią typu hierarchii (*ProductType*). Głębokość hierarchii zależy od współczynnika skalowania i jest wyliczana z $d = round(log10(n))/2+1$, gdzie $n$ to liczba instancji klasy *Product*. Produkty mają również zmienną liczbę cech (*ProductFeature*). Dwa produkty o tym samym typie dzielą ten sam zestaw możliwych cech produktu. Produkty są wytwarzane przez producentów (*Producer*). Liczba produktów przypadająca na producenta wynika z rozkładu normalnego ze średnią $\mu = 50$ i odchyleniem standardowym $\sigma = 16.6$. Produkty są oferowane przez sprzedawcę (*Vendor*). Oferty (*Offer*) są ważne przez określony czas i mają cenę z przedziału $[5, 10000]$. Liczba dni, w których trzeba dostarczyć produkt jest z przedziału $[1, 21]$. Oferty na dane produkty realizowane są przy użyciu rozkładu normalnego z parametrami $\mu = n/2$ oraz $\sigma = n/4$. Liczba ofert na sprzedawcę jest również realizowana za pomocą rozkładu normalnego z parametrami $\mu = 2000$ i $\sigma = 667$. Opinie *Review* składają się z tytułu i opisu, który składa się ze słów, których liczba jest między 50 a 300. Opinie mają maksymalnie 4 oceny i są to losowe liczby całkowite z przedziału $[1, 10]$.

| zbiór | produkty | cechy produktów | oferty | osoby | recenzje | trójki |
|---|---|---|---|---|---|---|
| 10 | 10 | 289 | 200 | 6 | 100 | 5007 |
| 20 | 20 | 289 | 400 | 10 | 200 | 8498 |
| 30 | 30 | 289 | 600 | 16 | 300 | 12022 |
| 40 | 40 | 569 | 800 | 21 | 400 | 16981 |
| 50 | 50 | 999 | 1000 | 26 | 500 | 22729 |
| 60 | 60 | 999 | 1200 | 30 | 600 | 26263 |
| 70 | 70 | 999 | 1400 | 37 | 700 | 29847 |
| 80 | 80 | 999 | 1600 | 41 | 800 | 33354 |
| 90 | 90 | 999 | 1800 | 46 | 900 | 36879 |
| 100 | 100 | 999 | 2000 | 50 | 1000 | 40377 |

Tablica 3.1: Opis wygenerowanych danych

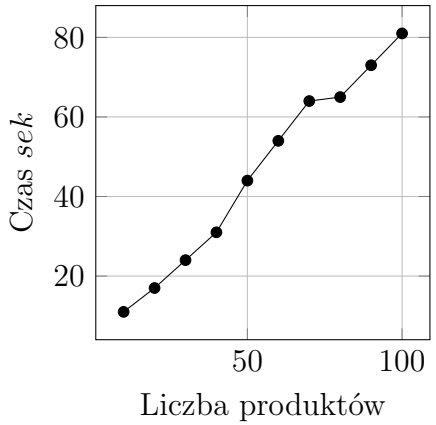
## 3.3 Eksperymenty

W tym podrozdziale zostaną zaprezentowane testy obciążenia, które zostały przeprowadzone przy pomocy Berlin SPARQL Benchmark (BSBM) [8]. Wspomniany generator w swej oryginalnej wersji nie posiada obsługi serializacji RDF/JSON (przykład 1.7). Została ona do niego doimplementowana. Kod źródłowy aplikacji znajduje sie pod adresem: `https://github.com/lszeremeta/bsbmtools-json`.
Testowany graf jest po skolemizacji [17], a więc nie posiada węzłów pustych.

**Definicja 3.1** (Skolemizacja)**.** Skolemizacja *to mechanizm zamiany węzłów pustych (przykład 1.3) na referencje IRI (przykład 1.1).* □

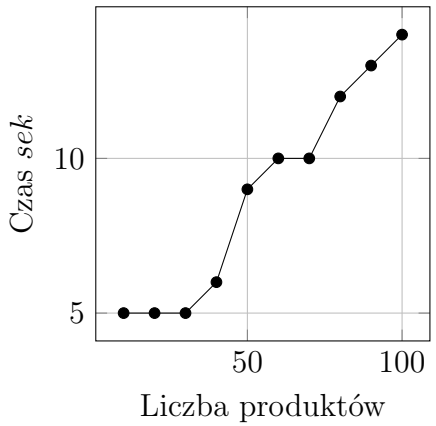
Testy obciążeniowe mierzą czas potrzebny do załadowania wszystkich trójek RDF do środowiska testowego przy pomocy klienta – RTriples. Kod źródłowy aplikacji znajduje sie pod adresem: `https://github.com/lszeremeta/RTriples`.

Na rysunku 3.2 zostały zaprezentowane testy bez proponowanych optymalizacji. Czasy wydają się zbyt długie w zastosowaniach realnych.




Rysunek 3.2: Testy ładowania bez optymalizacji

Rysunek 3.3 prezentuje wyniki ładowania danych z wykorzystaniem buforowania danych. Na wykresie widać, że dzięki redukcji operacji wejścia-wyjścia czasy ładowania znacznie się poprawiają i wydają się być zadowalające nawet w przypadku dużej liczby danych.



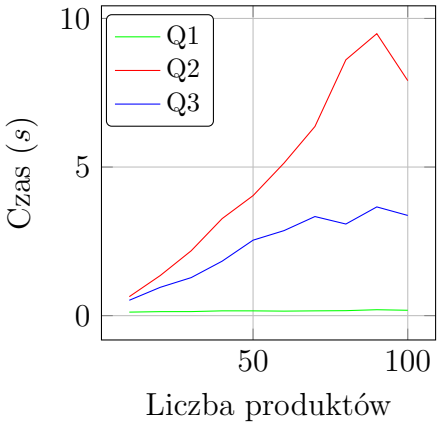

Rysunek 3.3: Testy ładowania z optymalizacją

Kolejny zestaw testów pozwalający ocenić wydajność proponowanego systemu, dotyczył operacji wybierania danych z semistrukturalnego magazynu grafów. Został zmierzony czas dla wszystkich obsługiwanych formatów wyjściowych oraz dla limitu danych na poziomie 10 oraz 100. Testy z większą wartością limitu były przeprowadzone bezpośrednio po eksperymentach z mniejszą wartością tego parametru i bez restartowania bazy danych. Środowisko testowe zostało poddane testom na trzy zapytania:

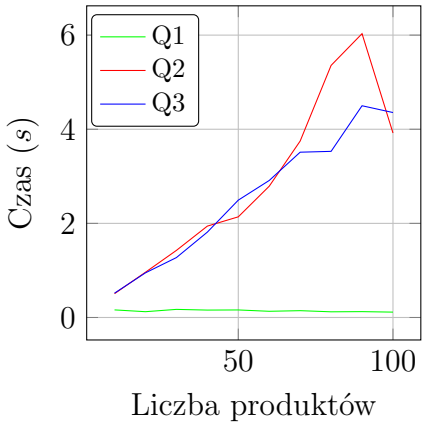
1. Q1: zapytanie wybierające wszystkie wyniki,
2. Q2: zapytanie wybierające predykaty i obiekty pasujące dla zadanego tematu,
3. Q3: zapytanie wybierające tematy i predykaty dla zadanego obiektu typu `typed-literal` z określonym typem.

Na rysunku 3.4 zostały zaprezentowane testy bez proponowanych optymalizacji. Czasy wykonania wydają się być możliwe do zaakceptowania szczególnie, gdy operujemy na mniejszych zbiorach danych.

Rys. 3.4a. Limit ustawiony na 10




Rys. 3.4b. Limit ustawiony na 100




Rysunek 3.4: Testy wybierania bez optymalizacji

Tabele 3.3 oraz 3.3 przedstawiają zestawienie czasów zapytania Q2 dla wszystkich obsługiwanych formatów. Możemy zauważyć, że wartości są do siebie bardzo zbliżone.

| zbiór | HTML | RDF/JSON | N-Triples | CSV | TXT |
|---|---|---|---|---|---|
| 10 | 0,639317 | 0,499001 | 0,506421 | 0,502461 | 0,504450 |
| 20 | 1,349230 | 0,965588 | 0,969615 | 0,969669 | 0,966011 |
| 30 | 2,182343 | 1,423137 | 1,424770 | 1,425798 | 1,423723 |
| 40 | 3,264202 | 1,958241 | 1,958052 | 1,954727 | 1,962659 |
| 50 | 4,037859 | 2,136330 | 2,138006 | 2,147609 | 2,133514 |
| 60 | 5,132782 | 2,794015 | 2,796026 | 2,803888 | 2,799835 |
| 70 | 6,364546 | 3,738322 | 3,751541 | 3,749293 | 3,749151 |
| 80 | 8,608902 | 5,383463 | 5,390001 | 5,374771 | 5,378377 |
| 90 | 9,487836 | 6,024153 | 6,048865 | 6,050515 | 6,028922 |
| 100 | 7,907734 | 3,981267 | 3,978394 | 3,952579 | 3,975572 |

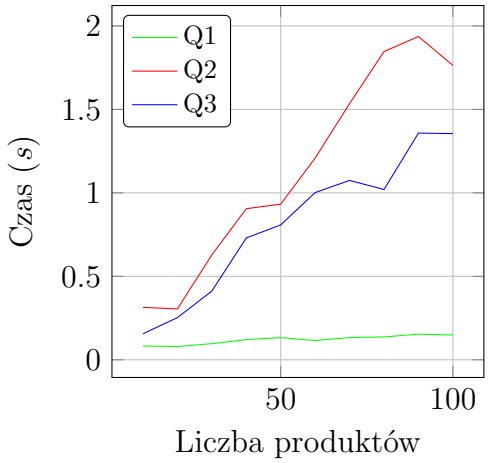
Tablica 3.2: Testy wybierania bez optymalizacji: zapytanie Q2, wszystkie obsługiwane formaty, limit równy 10 (czasy podane w sekundach)

| zbiór | HTML | RDF/JSON | N-Triples | CSV | TXT |
|---|---|---|---|---|---|
| 10 | 0,506047 | 0,506529 | 0,506130 | 0,499553 | 0,504527 |
| 20 | 0,961406 | 0,966248 | 0,962625 | 0,962609 | 0,962797 |
| 30 | 1,425522 | 1,412512 | 1,422709 | 1,424609 | 1,414247 |
| 40 | 1,941558 | 1,933735 | 1,944718 | 1,941091 | 1,934907 |
| 50 | 2,138672 | 2,153551 | 2,144044 | 2,131833 | 2,126102 |
| 60 | 2,789242 | 2,781652 | 2,790488 | 2,792396 | 2,785309 |
| 70 | 3,749019 | 3,732880 | 3,749160 | 3,751582 | 3,752320 |
| 80 | 5,356565 | 5,360915 | 5,383308 | 5,362240 | 5,333586 |
| 90 | 6,030514 | 5,986979 | 6,005629 | 5,990614 | 5,980516 |
| 100 | 3,920396 | 3,949293 | 3,924243 | 3,919427 | 3,906689 |

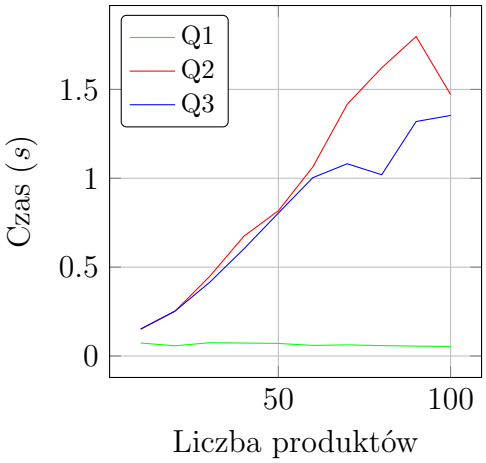
Tablica 3.3: Testy wybierania bez optymalizacji: zapytanie Q2, wszystkie obsługiwane formaty, limit równy 100 (czasy podane w sekundach)

Na rysunkach 3.5a, 3.5b oraz tabelach 3.3 i 3.3 pokazano czasy wybierania trójek RDF do środowiska testowego po zastosowaniu buforowania zbliżonego do modelu LRU (podrozdział 2.2.1). Wyniki pokazują, że czas ładowania są około 5 razy krótsze niż bez optymalizacji (rysunek 3.4).

Rys. 3.5a. Limit równy 10




Rys. 3.5b. Limit równy 100




Rysunek 3.5: Testy wybierania z buforowaniem

| zbiór | HTML | RDF/JSON | N-Triples | CSV | TXT |
|---|---|---|---|---|---|
| 10 | 0,314240 | 0,151175 | 0,149455 | 0,149980 | 0,149815 |
| 20 | 0,304652 | 0,251588 | 0,251627 | 0,259146 | 0,250956 |
| 30 | 0,628254 | 0,591456 | 0,594124 | 0,591899 | 0,594013 |
| 40 | 0,905058 | 0,665024 | 0,669216 | 0,670328 | 0,669601 |
| 50 | 0,932533 | 0,804828 | 0,813917 | 0,816908 | 0,813912 |
| 60 | 1,208271 | 1,055738 | 1,060337 | 1,063326 | 1,060050 |
| 70 | 1,534901 | 1,415004 | 1,415126 | 1,412973 | 1,418806 |
| 80 | 1,846534 | 1,626495 | 1,631112 | 1,630704 | 1,633280 |
| 90 | 1,936703 | 1,767093 | 1,790035 | 1,793109 | 1,793608 |
| 100 | 1,763964 | 1,453812 | 1,466341 | 1,468721 | 1,464906 |

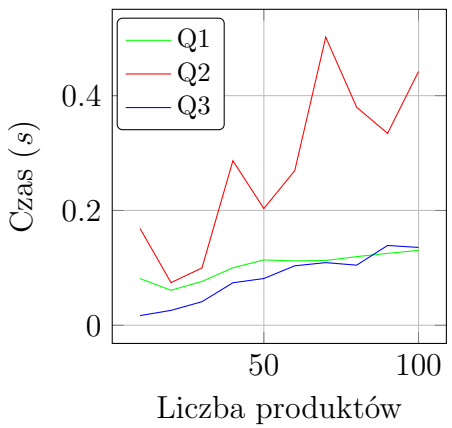
Tablica 3.4: Testy wybierania z buforowaniem: zapytanie Q2, wszystkie obsługiwane formaty, limit równy 10 (czasy podane w sekundach)

| zbiór | HTML | RDF/JSON | N-Triples | CSV | TXT |
|---|---|---|---|---|---|
| 10 | 0,150060 | 0,148382 | 0,149449 | 0,149093 | 0,149321 |
| 20 | 0,251917 | 0,250191 | 0,249987 | 0,249595 | 0,250357 |
| 30 | 0,446849 | 0,441618 | 0,446299 | 0,446370 | 0,448818 |
| 40 | 0,674253 | 0,660852 | 0,668892 | 0,670927 | 0,671093 |
| 50 | 0,817346 | 0,805013 | 0,815293 | 0,812776 | 0,814818 |
| 60 | 1,063417 | 1,055940 | 1,058773 | 1,062927 | 1,060469 |
| 70 | 1,416524 | 1,407278 | 1,410291 | 1,416763 | 1,411715 |
| 80 | 1,621365 | 1,611863 | 1,618209 | 1,616814 | 1,619732 |
| 90 | 1,797628 | 1,771585 | 1,795451 | 1,797791 | 1,792480 |
| 100 | 1,471454 | 1,456074 | 1,464869 | 1,469653 | 1,467658 |

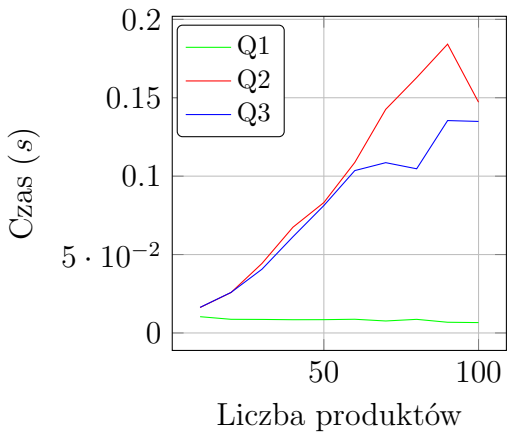
Tablica 3.5: Testy wybierania z buforowaniem: zapytanie Q2, wszystkie obsługiwane formaty, limit równy 100 (czasy podane w sekundach)

Wykresy 3.6a i 3.6b prezentują wyniki ładowania danych zarówno z włączonym buforowaniem, jak i ustawionymi nagłówkami HTTP – `Cache-Control` oraz `Expires` omówionymi w podrozdziale 2.2.2. Wykorzystanie odpowiednich nagłówków HTTP jest szczególnie istotne w realnych zastosowaniach, skutecznie ograniczając ilość i czas transferu danych.

Rys. 3.6a. Limit równy 10




Rys. 3.6b. Limit równy 100




Rysunek 3.6: Testy wybierania z buforowaniem i ustawionymi nagłówkami HTTP

# Podsumowanie

Dotychczasowy model Internetu był projektowany w celu łatwiejszej wymiany dokumentów. Semantyczny Internet i Dane Połączone idą dalej i stwarzają nowe możliwości analizy różnego rodzaju informacji. Dzięki temu podejściu do globalnej pajęczyny, możemy konstruować konkretne zapytania oraz uzyskiwać przydatne informacje bez większego wysiłku.

Celem niniejszej pracy było omówienie zagadnień związanych z Semantycznym Internetem i Danymi Połączonymi, różnych koncepcji buforowania danych oraz implementacja kompletnego magazynu grafów RDF – RTriples. Cel ten, został osiągnięty w całości.

Testy części praktycznej pracy dyplomowej, zostały przeprowadzone na wygenerowanych za pomocą narzędzia BSBM (sekcja 3.2), zbiorów testowych. Do wspomnianego generatora, została doimplementowaną obsługa serializacji RDF/JSON.

Wyniki eksperymentów pokazują duży potencjał zaprezentowanego rozwiązania, które z powodzeniem może być stosowane w środowiskach Semantycznego Internetu i Danych Połączonych. Cechuje się ono dużymi, możliwościami filtrowania danych, obsługą różnych technik buforowania, a także akceptowalnymi czasami przetwarzania danych. Obsługa różnych formatów wyjściowych pozwala na szerokie wykorzystanie przetworzonych danych przez inne istniejące systemy.

Przyszłe prace powinny się koncentrować na dodaniu kompatybilności z Triple Pattern Fragments (TPF) [52] oraz poprawie strategii buforowania danych.

# Bibliografia


[1] Julie D. Allen, Deborah Anderson, Joe Becker, Richard Cook, Mark Davis, Peter Edberg, Michael Everson, Asmus Freytag, Laurentiu Iancu, Richard Ishida, John H. Jenkins, Ken Lunde, Rick McGowan, Lisa Moore, Eric Muller, Addison Phillips, Roozbeh Pournader, Michel Suignard, and Ken Whistler. Unicode 7.0.0. Technical report, Unicode Consortium, October 2014. `http://www.unicode.org/versions/Unicode7.0.0/`.

[2] Andrés Aranda-Andújar, Francesca Bugiotti, Jesús Camacho-Rodríguez, Dario Colazzo, François Goasdoué, Zoi Kaoudi, and Ioana Manolescu. AMADA: web data repositories in the amazon cloud. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2749–2751. ACM, 2012.

[3] 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/`.

[4] Greg Barish and Katia Obraczke. World wide web caching: Trends and techniques. *IEEE Communications magazine*, 38(5):178–184, 2000.

[5] M. Belshe, R. Peon, and M. Thomson. Hypertext Transfer Protocol Version 2 (HTTP/2). RFC 7540, RFC Editor, May 2015. `http://www.rfc-editor.org/rfc/rfc7540.txt`.

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

[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] 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/.

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

[11] Nieves R Brisaboa, Ana Cerdeira-Pena, Antonio Farina, and Gonzalo Navarro. A Compact RDF Store using Suffix Arrays.

[12] Jeen Broekstra, Arjohn Kampman, and Frank Van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *The Semantic Web—ISWC 2002*, pages 54–68. Springer, 2002.

[13] Francesca Bugiotti, François Goasdoué, Zoi Kaoudi, and Ioana Manolescu. RDF data management in the Amazon cloud. In *Proceedings of the 2012 Joint EDBT/ICDT Workshops*, pages 61–72. ACM, 2012.

[14] 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/.

[15] 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/.

[16] Philippe Cudré-Mauroux, Iliya Enchev, Sever Fundatureanu, Paul Groth, Albert Haque, Andreas Harth, Felix Leif Keppmann, Daniel Miranker, Juan F Sequeda, and Marcin Wylot. Nosql databases for rdf: An empirical evaluation. In *The Semantic Web–ISWC 2013*, pages 310–325. Springer, 2013.

[17] 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/.

[18] Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[19] 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.

[20] R. Fielding, M. Nottingham, and J. Reschke. Hypertext Transfer Protocol (HTTP/1.1): Caching. Technical Report 7234, RFC Editor, June 2014. http://www.rfc-editor.org/rfc/rfc7234.txt.

[21] Adam Maria Gadomski. Meta-Ontological Assumptions: Information, Preferences and Knowledge universal interrelations (IPK cognitive architecture). *White paper, page since*, 1999.

[22] Luis Galárraga, Katja Hose, and Ralf Schenkel. Partout: A distributed engine for efficient rdf processing. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion*, pages 267–268. International World Wide Web Conferences Steering Committee, 2014.

[23] 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/.

[24] Tom Heath and Christian Bizer. *Linked data: Evolving the web into a global data space*. Morgan & Claypool Publishers, 2011.

[25] Ian Hickson, Robin Berjon, Steve Faulkner, Travis Leithead, Edward O'Connor, Erika Doyle Navara, and Silvia Pfeiffer. HTML5. W3C recommendation, W3C, October 2014. http://www.w3.org/TR/2014/CR-html5-20140731/.

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

[27] Katja Hose and Ralf Schenkel. WARP: Workload-aware replication and partitioning for RDF. In *Data Engineering Workshops (ICDEW), 2013 IEEE 29th International Conference on*, pages 1–6. IEEE, 2013.

[28] Jiewen Huang, Daniel J Abadi, and Kun Ren. Scalable SPARQL querying of large RDF graphs. *Proceedings of the VLDB Endowment*, 4(11):1123–1134, 2011.

[29] Zoi Kaoudi and Ioana Manolescu. RDF in the Clouds: A Survey. *The VLDB Journal*, pages 1–25, 2014.

[30] Vaibhav Khadilkar, Murat Kantarcioglu, Bhavani Thuraisingham, and Paolo Castagna. Jena-hbase: A distributed, scalable and efficient rdf triple store. In *Proceedings of the 11th International Semantic Web Conference Posters & Demonstrations Track, ISWC-PD*, volume 12, pages 85–88. Citeseer, 2012.

[31] Kjetil Kjernsmo. A survey of HTTP caching implementations on the open Semantic Web. In *The Semantic Web. Latest Advances and New Domains*, pages 286–301. Springer, 2015.

[32] Guenter Ladwig and Andreas Harth. CumulusRDF: Linked Data Management on Nested Key-Value Stores. In *Proceedings of the 7th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2011) at the 10th International Semantic Web Conference (ISWC2011)*, 2011.

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

[34] André Pessoa Negrão, Carlos Roque, Paulo Ferreira, and Luís Veiga. An adaptive semantics-aware replacement algorithm for web caching. *Journal of Internet Services and Applications*, 6(1):1–14, 2015.

[35] 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 companion on World Wide Web*, pages 397–400. ACM, 2012.

[36] Stefan Podlipnig and Laszlo Böszörmenyi. A survey of web cache replacement strategies. *ACM Computing Surveys (CSUR)*, 35(4):374–398, 2003.

[37] 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/`.

[38] Martin Przyjaciel-Zablocki, Alexander Schätzle, Thomas Hornung, Christopher Dorner, and Georg Lausen. Cascading map-side joins over HBase for scalable join processing. *CoRR*, abs/1206.6293, 2012.

[39] Roshan Punnoose, Adina Crainiceanu, and David Rapp. Rya: A Scalable RDF Triple Store for the Clouds. In *Proceedings of the 1st International Workshop on Cloud Intelligence*, Cloud-I '12, pages 4:1–4:8, New York, NY, USA, 2012. ACM.

[40] Shalini Ramanathan, Savita Goel, and Subramanian Alagumalai. Comparison of Cloud database: Amazon's SimpleDB and Google's Bigtable. In *Recent Trends in Information Systems (ReTIS), 2011 International Conference on*, pages 165–168. IEEE, 2011.

[41] Padmashree Ravindra, HyeongSik Kim, and Kemafor Anyanwu. An intermediate algebra for optimizing RDF graph pattern matching on MapReduce. In *The Semanic Web: Research and Applications*, pages 46–61. Springer, 2011.

[42] Kurt Rohloff and Richard E Schantz. Clause-iteration with MapReduce to scalably query datagraphs in the SHARD graph-store. In *Proceedings of the fourth international workshop on Data-intensive distributed computing*, pages 35–44. ACM, 2011.

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

[44] Y. 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`.

[45] Vasil Slavov, Anas Katib, Praveen Rao, Srivenu Paturi, and Dinesh Barenkala. Fast Processing of SPARQL Queries on RDF Quadruples.

[46] Manu Sporny, Ivan Herman, Ben Adida, and Mark Birbeck. RDFa 1.1 Primer - Second Edition. W3C note, W3C, August 2013. `http://www.w3.org/TR/2013/NOTE-rdfa-primer-20130822/`.

[47] Raffael Stein and Valentin Zacharias. Rdf on cloud number nine. In *4th Workshop on New Forms of Reasoning for the Semantic Web: Scalable and Dynamic*, pages 11–23, 2010.

[48] D Tomaszuk. Named graphs in rdf/json serialization. *Zeszyty Naukowe Politechniki Gdańskiej*, pages 273–278, 2011.

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

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

[51] Jürgen Umbrich, Marcel Karnstedt, Aidan Hogan, and Josiane Xavier Parreira. Hybrid SPARQL queries: fresh vs. fast results. In *The Semantic Web–ISWC 2012*, pages 608–624. Springer, 2012.

[52] Ruben Verborgh, Miel Vander Sande, Pieter Colpaert, Sam Coppens, Erik Mannens, and Rik Van de Walle. Web-scale querying through linked data fragments. In *Proceedings of the 7th Workshop on Linked Data on the Web*, 2014.

[53] Stefan Werner, Dennis Heinrich, Marc Stelzner, Volker Linnemann, Thilo Pionteck, and Sven Groppe. Accelerated join evaluation in Semantic Web databases by using FPGAs. *Concurrency and Computation: Practice and Experience*, 2015.

[54] Kevin Wilkinson, Craig Sayers, Harumi A Kuno, Dave Reynolds, et al. Efficient RDF Storage and Retrieval in Jena2. In *SWDB*, volume 3, pages 131–150. Citeseer, 2003.

[55] Piotr Wróblewski. *Algorytmy, struktury danych i techniki programowania. Wydanie IV*. Helion, Gliwice, 2010.

[56] Kai Zeng, Jiacheng Yang, Haixun Wang, Bin Shao, and Zhongyuan Wang. A distributed graph engine for web scale rdf data. *Proceedings of the VLDB Endowment*, 6(4):265–276, 2013.

# Spis rysunków

# Spis tablic