Bridging the Technology Gap Between Industry and Semantic Web:
Generating Databases and Server Code From RDF

Markus Schröder, Michael Schulze, Christian Jilek and Andreas Dengel

German Research Center for Artificial Intelligence

Technische Universität Kaiserslautern

Abstract

Despite great advances in the area of Semantic Web, industry rather seldom adopts Semantic Web technologies and their storage and query approaches. Instead, relational databases (RDB) are often deployed to store business-critical data, which are accessed via REST interfaces. Yet, some enterprises would greatly benefit from Semantic Web related datasets which are usually represented with the Resource Description Framework (RDF). To bridge this technology gap, we propose a fully automatic approach that generates suitable RDB models with REST APIs to access them. In our evaluation, generated databases from different RDF datasets are examined and compared. Our findings show that the databases sufficiently reflect their counterparts while the API is able to reproduce rather simple SPARQL queries. Potentials for improvements are identified, for example, the reduction of data redundancies in generated databases.

Introduction

- observation: distinct solutions between Semantic Web and industry

Approach	Semantic Web	Industry
Storage	Triplestore	SQL Database
Domain Modeling	Terminology in Ontology	Database Schema
Data Modeling	RDF Statement Assertions	Database Records
Identification	URIs	Primary Keys
Query Interface	SPARQL	SQL / REST API
Exchange Format	Result Set / RDF	Result Set / JSON

- goal: bridging the technology gap by generating data and code

Approach

Analysis of RDF

```
(12):malta a :Country ;
      :hasCapitalCity :valletta .

(11):czechRepublic a :Country ;
      :hasCapitalCity :prague .

(2):valletta a :CapitalCity .

(1):prague a :CapitalCity .

(10):The_Kappillan_of_Malta a :Book ;
      :takesPlaceIn :malta ;
      :hasPage :The_Kappillan_of_Malta_Page1 ,
               :The_Kappillan_of_Malta_Page2 ;
      :hasCover <https://example.org/img1.jpg> .

(9):In_Love_With_Malta a :Book ;
      :publishedDate "2016-06-24"^^xsd:date ;
      :takesPlaceIn :malta ;
      :name "In Love With Malta"@en ,
            "Verliebt in Malta"@de ;
      :hasPage [ (3)
            a :Page ;
            :mentions :prague , :valletta
          ] .

(8):Like_Bees_to_Honey a :Book , :Paperback ;
      :hasCover <https://example.org/img2.jpg> ;
      :takesPlaceIn :malta ;
      :hasPage :Like_Bees_to_Honey_Page1 ,
               :Like_Bees_to_Honey_Page2 .

(5):The_Kappillan_of_Malta_Page1 a :Page ;
      :mentions :valletta .
(6):The_Kappillan_of_Malta_Page2 a :Page ;
      :mentions :valletta .

(4):Like_Bees_to_Honey_Page1 a :Page ;
      :mentions :valletta , :prague .
(7):Like_Bees_to_Honey_Page2 a :Page ;
      :mentions :valletta , :prague .
```

blank node skolemization

Conversion to RDB (SQLite)

class to table

country

id	has_capital_city
11	1
12	2

capital_city

id
1
2

one-to-one cardinality

book

id	has_cover	published_date	takes_place_in
8	https://example.org/img2.jpg	*NULL*	12
9	*NULL*	1466726400000	12
10	https://example.org/img1.jpg	*NULL*	12

dangling links

date to seconds (long)

many-to-one cardinality

language strings

nm_book_name

id	value_book_name	lang
9	In Love With Malta	en
9	Verliebt in Malta	de

class to table

paperback

id	has_cover	takes_place_in
8	https://example.org/img2.jpg	12

redundancy

many-to-many cardinality

mn_page_mentions_capital_city

page_id	capital_city_id
3	2
3	1
4	1
4	2
5	2
6	2
7	2
7	1

one-to-many cardinality

page

id	book_has_page	paperback_has_page
3	9	*NULL*
4	8	8
5	10	*NULL*
6	10	*NULL*
7	8	8

Generation of REST API Code

API Package

```java
public class Country {
    private Long id;
    private Long hasCapitalCity;
}
```

```java
public class Book {
    private Long id;
    private String hasCover;
    private Long publishedDate;
    private Long takesPlaceIn;
    private List<LangString> name;
}
```

```java
public class CapitalCity {
    private Long id;
}
```

```java
public class Paperback {
    private Long id;
    private String hasCover;
    private Long takesPlaceIn;
}
```

```java
public class Page {
    private Long id;
    private Long bookHasPage;
    private Long paperbackHasPage;
    private List<Long> mentionsCapitalCity;
}
```

```java
public class DatabaseController {

    public DatabaseController(File databaseFile) {...3 lines }

    public List<Book> selectBook() {...3 lines }

    public List<Book> selectBook(List<Long> ids, Integer offset, Integer limit

    public List<Book> insertBook(List<Book> records) {...67 lines }

    public List<Book> updateBook(List<Book> records, boolean put) {...70 lines

    public void deleteBook() {...3 lines }

    public void deleteBook(List<Long> ids) {...24 lines }
```

Server Package

```java
public class BookResource {

    private DatabaseController dbc;

    public BookResource(DatabaseController dbc) {
        this.dbc = dbc;

        Spark.get("/book", this::getBook);
        Spark.get("/book/:id", this::getBookId);

        Spark.delete("/book", this::deleteBook);
        Spark.delete("/book/:id", this::deleteBookId);

        Spark.post("/book", this::postBook);

        Spark.put("/book/:id", (req, resp) -> putOrPatchBookId(req, resp, t
        Spark.patch("/book/:id", (req, resp) -> putOrPatchBookId(req, resp,
    }

    public Object getBook(Request req, Response resp) {...3 lines }

    public Object getBookId(Request req, Response resp) {...3 lines }

    private Object getBook(Request req, Response resp, List<Long> ids) {...

    public Object postBook(Request req, Response resp) {...8 lines }

    public Object putOrPatchBookId(Request req, Response resp, boolean put)

    public Object deleteBook(Request req, Response resp) {...5 lines }

    public Object deleteBookId(Request req, Response resp) {...5 lines }
```

Usage

```
GET /book
```

```
{
  "type": "Book",
  "list": [
    {
      "id": 8,
      "hasCover": "https://example.org/img2.jpg",
      "takesPlaceIn": 12
    },
    {
      "id": 9,
      "publishedDate": 1466726400000,
      "takesPlaceIn": 12,
      "name": [
        {
          "string": "In Love With Malta",
          "lang": "en"
        },
        {
          "string": "Verliebt in Malta",
          "lang": "de"
        }
      ]
    },
    {
      "id": 10,
      "hasCover": "https://example.org/img1.jpg",
      "takesPlaceIn": 12
    }
  ]
}
```

```
POST /book
{
  "hasCover": "https://example.org/img3.jpg",
  "takesPlaceIn": 12
}
```

```
Location: /book/11
```

Evaluation

Datasets

No.	Name	Stmts	Cls	Multi-Typed instances MT	avgMT	Object Datatype OP	Prop. DP	Cardinalities OO	MO	OM	MM	Entity Tables ET	Many-to-Many Tables MMT	average no. of Columns per entity table avgCol
1	TBL-C	109	5	1	2	26	18	31	19	7	1	5	11	12.2 ± 12.4
2	CTB	10,853	4	0	-	10	5	4	6	1	4	4	7	3.0 ± 1.6
3	EAT	1,674,376	2	0	-	3	3	1	5	0	0	3	1	2.7 ± 2.1
4	Pokedex	26,562	19	0	-	9	29	13	28	5	3	19	40	3.5 ± 6.7
5	BOW	4,041,676	15	349,195	2.9 ± 0.9	7	19	2	9	0	15	15	180	5.3 ± 3.0
6	S-IT	4,477	406	81	14.3 ± 9.0	9	25	18	6	3	7	406	3,056	2.6 ± 1.0
7	IndScn	25,016	16	0	-	24	37	4	34	0	23	16	74	5.7 ± 4.5
8	BSBM	40,177	22	100	2.0 ± 0.0	12	28	16	22	0	2	22	17	13.7 ± 5.5

Comparison with RDF2RDB

- it complies more with the RDF model: more tables

REST Interface Test with BSBM

- utilize provided queries
- check if our API can reproduce them

Results

- sufficiently reflect RDF dataset counterparts but …
  - multi-typed instances: data redundancy
  - trade-off: property becomes a table or a column
- API can reproduce rather simple and common queries

Resources https://github.com/mschroeder-github/rdf-to-rdb-rest-api

Acknowledgement

This work was funded by the BMBF project SensAI (grant no. 01IW20007).

Contact:
M.Sc. Markus Schröder
Doctoral Researcher
German Research Center for Artificial Intelligence (DFKI GmbH)
Smart Data & Knowledge Services

Phone: +49 631 20575-2070
Mail: markus.schroeder@dfki.de
Website: http://www.dfki.uni-kl.de/~mschroeder/