

# MyRDF

A RDF store for PostgreSQL

# What is it?

- A datastore for W3C RDF data built on PostgreSQL RDBMS
- It tries to be small and practical, while conforming to RDF semantics
- It installs as a set of tables and functions in a PostgreSQL database schema. No extra libraries on the client are needed.
- NoSPARQL! (but it might get added later)

# Features

- Functional SQL API for manipulating and querying RDF data
- Support for languages and datatypes, per RDF
- Loading and unloading of models. Each RDF statement belongs to some model, typically the URL of the imported RDF file

# Uses

- Store "metadata" for just about anything that has a URL
- Locally cache data from remote RDF databases for speed
- Use the power of Postgres for managing your RDF data
- Great for web server applications

# Cool features

- Quintuple store  
(subject, predicate, object, context, model)
- Small and powerful. Implemented in PL/PgSQL and PL/Perl. Uses some standard Perl libraries.
- Can use Redland library for robust RDF parsing, so several RDF input formats are supported.

# RDF concepts

RDF database is a set of **statements** about **resources**.



- A Resource is identified by **URL**

# RDF concepts

- **URL** is primary key for everything
- Data as a graph of links

$M = \{R, S\}$

M - model

R - resources

S - statements

- Can store "anything" this way
- Good for Artificial Intelligence ☺

# Resources

- URLs or anonymous nodes



# Models

- Models are resources
- Models hold groups of statements and resources together
- Typically a file or URL where RDF was loaded from
- Can load and remove whole model as a single transaction
- Each database user is also a model

# Models

- Have read/write permissions stored as postgres aclitems

# Properties

# Namespaces

(name,url)

- URL shortening service
- Allows everyone with URL to define a namespace
- Namespace names are local to a database
- Useful for partitioning

# Statements

- With objects or literals
- Literals can have optional datatype (x) or language

# Datatypes

- Can be specified for statements with literals

`"Hello"^^xsd:normalizedString`

- Many already exist in **xsd** namespace

`xsd:string`

`xsd:decimal`

`xsd:dateTime`

`xsd:normalizedString`

Nor very useful right now. Might add constraint checking later.

# Partitioning

- By namespace
- By model
- By subject
- By predicate
- By object

# Turtle notation

Unified textual representation

- URL: `<http://www.w3c.org/>`
- Literal: `"Hello"`
- Typed literal: `"Hello"^^xsd:string`
- Language literal: `"Hello"@en`



# Data manipulation

- Adding statements  
`assert()`
- Replacing statements  
`replace()`
- Deleting statements  
`retract()`

# Adding data

- Loading a whole model

```
SELECT rdf.model_load('http://....');
```

- Individual statements

```
SELECT rdf.assert(  
  rdf.resource('http://www.imdb.com/title/tt0088846/'),  
  rdf.predicate('dc:title'),  
  "Brazil"  
);
```

# Asserting statements

- `assert(turtle,turtle,turtle)`  
specified in turtle notation
- `assert(url,url,url)`  
specified with 3 URLs
- `assert(url,url,string,url)`  
literal with datatype
- `assert(url,url,string,lang)`  
literal with language

# Asserting statements faster

- `assert(nid,nid,nid)`
- `assert(nid,nid,string,url)`
- `assert(nid,nid,string,lang)`

# Other variations

- `assert_with_context`
- `assert_into_model`

# Replacing statements

- `replace(turtle,turtle,turtle)`
- `replace(url,url,url)`
- `replace(url,url,string,url)`
- `replace(url,url,string,lang)`

# Removing statements

- `retract(subject,predicate,object,...)`

# Query scenarios

- Find all properties for one object (URL)
- Find values of a single property for many objects
- Find only certain properties for one object
- Find related objects



# Query scenarios

- Find all child objects (like files in a folder, but for URLs)
- Allow limiting only to certain models

# Query function

```
FUNCTION query(  
    subject,  
    predicate,  
    object,  
    model,  
    context  
)
```

RETURNS SETOF statement

- NULL arguments match all triples
- Smart, finds good query plan

# Missing features

- No SPARQL

This might get added later, possibly in a form of a compiler from SPARQL to PostgreSQL function.

- Type inference

This is needed for really cool OO programming!