Stardog 5

THE MANUAL

Stardog is the world's leading
KNOWLEDGE GRAPH PLATFORM FOR THE ENTERPRISE
Stardog makes it fast and easy to turn enterprise data into knowledge.

What's New in Stardog 5

- ✓ GraphQL Queries (#_graphql_queries)
- ✓ Path Queries (#_path_queries)
- ✓ Stored Functions (#_managing_stored_functions)
- ✓ Inline Rules (#_stardog_rules_syntax)
- ✓ Native Memory Management (#_memory_management)
- ✓ All-new Virtual Graph Engine (#_structured_data)
- ✓ SPARQL Query Hints (#_using_query_hints)

Check out the Quick Start Guide (#_quick_start_guide) to get Stardog installed and running in five easy steps.

INTRODUCTION

Stardog **5.3.6** (**31 Oct 2018**) supports the RDF graph data model (http://www.w3.org/RDF/); SPARQL (http://www.w3.org/TR/sparql11-overview/) query language; property graph model (https://github.com/tinkerpop/blueprints/wiki/Property-Graph-Model) and Gremlin graph traversal language (http://tinkerpop.incubator.apache.org/docs/3.0.2-incubating/#_on_gremlin_language_variants); OWL 2 (http://www.w3.org/TR/owl2-overview/) and user-defined rules for inference and data analytics; virtual graphs; geospatial query answering; and programmatic interaction via several languages and network interfaces.

Stardog is made by hand with skill, taste, and a point of view in DC, Boston, Heidelberg, Madison, Moscow, Hanalei, South Dakota, Porto, Manchester, PA, and Columbia, MO. 🌟 🐶

To learn more about where we've been and where we're headed, consult the RELEASE NOTES (/docs/release-notes/) and milestones (#_milestones).

Downloading & Support

Download (http://stardog.com/#download) Stardog to get started. The Stardog support forum (https://community.stardog.com/) is the place to report bugs, ask questions, etc.

Contributing

There are several open source components of Stardog; feel free to submit pull requests: stardog-docs (https://github.com/complexible/stardog-docs), stardog.js (https://github.com/complexible/stardog.js), stardog-groovy (https://github.com/complexible/stardog-groovy), stardog-spring (https://github.com/complexible/stardog-spring), stardog.rb (https://github.com/antoniogarrote/stardog-rb), and stardog-clj (https://github.com/complexible/stardog-clj). Many thanks to Stardog customers, users, contributors, testers, etc. You know who you are, and so do we. Thank you![1 (#_footnote_1)] You can also help by editing these docs on Github (https://github.com/Complexible/stardog-docs/tree/master/src/doc) if you spot a problem.

ENTERPRISE SUPPORT

Customers with Enterprise Premium Support have access to some extra capabilities and services. Enterprise Premium Support is available as of Stardog 3 release. Email (mailto:sales@stardog.com) for details.

Real-time Support

Get access to the core Stardog development team in real-time via voice or chat. Let us help you get the most from Stardog, 24/7. Our core team has more semantic graph application and tool development experience than any other team on the planet. Other vendors shunt you off to inexperienced level one techs. We've never done that and never will.

Private Maven Repositories

See Using Maven (#_using_maven) for details; this includes a per-customer, private Maven repository, CDN-powered, for 24/7 builds, updates, and feature releases.

We're also tying Maven and Docker together, providing private Docker repositories for customers, which allows us to build out clusters, custom configurations, best practices, and devops tips-and-tricks into custom Docker images…so that you don't have to.

Private Docker Repositories

Docker-based deliverables not only shortens your development and devops cycles but keeps you up-to-date with the latest-greatest versions of Stardog, including security fixes, performance hot fixes, and deployment best practices and configurations.

Priority Bug Fixes

With Maven and Docker in place, we've got a software delivery mechanism ready to push priority bug fixes into your enterprise as soon as they're ready. We've averaged one Stardog release every two weeks since 2012. Enterprise Premium Support customers can now take advantage of our development pace in a controlled fashion.

Priority Feature Releases

We hate holding new features in a feature branch, especially for mundane business reasons; we want to release new stuff as soon as possible to our customers. With Enterprise Premium Support, we can maintain a disruptive pace of innovation without disrupting **you**.

QUICK START GUIDE

Requirements

It just doesn't get any easier than this: **Stardog runs on Java 8**. Stardog runs best on, but does not require, a 64-bit JVM that supports `sun.misc.Unsafe`.

Insecurity

We optimize Stardog out-of-the-box for ease and simplicity. You should take additional steps to secure it before production deployment. It's easy and it's smart, so just do it. In case that's not blunt enough:

NOTE Stardog ships with an **insecure** but usable default setting: the super user is `admin` and the `admin` password is "admin". This is fine until it isn't, at which point you should read the Security section.

Upgrading to Stardog 5

If you are upgrading to Stardog 5 from any previous version, please see Migrating to Stardog 5 (#_migrating_to_stardog_5) for details about auto-migrating on-disk indexes.

Package Managers

As of version 5.0.6 Stardog is available in rpm and deb packages and apt and yum respositories have been setup to make installation easier.

Debian Based Systems

To install Stardog using apt-get run the following commands:

```
curl http://packages.stardog.com/stardog.gpg.pub | apt-key add
echo "deb http://packages.stardog.com/deb/ stable main" >> /etc/apt/sources.list
apt-get update
apt-get install -y stardog[=<version>]
```

This will first add the Stardog gpg key to the systems and then fetch and install the latest Stardog deb package.

RPM Based Systems

To install Stardog using yum run the following commands:

```
curl http://packages.stardog.com/rpms/stardog.repo > /etc/yum.repos.d/stardog.repo
yum install -y stardog[-<version>]
```

Amazon EC2

Certain Amazon EC2 instances do not let you redirect output into /etc/yum.repos.d as specified above. On such instances you can install Stardog like so:

```
sudo yum-config-manager --add-repo http://packages.stardog.com/rpms/stardog.repo
sudo yum-config-manager --enable stardog
yum install -y stardog[-<version>]
```

Package Layout

The packages require that OpenJDK 8 and all of its dependecies are installed on the system. The package managers will install them if they are not already there. Stardog is then configured to start on boot via systemd and thus it can be controlled by the systemctl tool as shown below:

```
systemctl start stardog
systemctl restart stardog
systemctl stop stardog
```

To customize the environment in which stardog is run the file /etc/stardog.env.sh can be altered with key value pairs, for example:

```
export STARDOG_HOME=/var/opt/stardog
export STARDOG_SERVER_JAVA_ARGS="-Xmx8g -Xms8g -XX:MaxDirectMemorySize=2g"
```

Linux and OSX

First, tell Stardog where its home directory (where databases and other files will be stored) is:

```
$ export STARDOG_HOME=/data/stardog
```

Note: If using a package manager this line is added to the /etc/stardog.env.sh file.

If you're using some weird Unix shell that doesn't create environment variables in this way, adjust accordingly. **If `STARDOG_HOME` isn't defined, Stardog will use the Java `user.dir` property value.**

NOTE You should consider the upgrade process when setting `STARDOG_HOME` for production or other serious usage. In particular, you probably don't want to set the directory where you install Stardog as `STARDOG_HOME` as that makes upgrading less easy. Set `STARDOG_HOME` to some other location.

Second, copy the `stardog-license-key.bin` into the right place:

```
$ cp stardog-license-key.bin $STARDOG_HOME
```

Of course `stardog-license-key.bin` has to be readable by the Stardog process.

Stardog won't run without a valid `stardog-license-key.bin` in `STARDOG_HOME`.

Third, optionally, place the `bin` folder of the Stardog install on your PATH so the `stardog` and `stardog-admin` scripts can be used regardless of current working directory

```
$ export PATH="$PATH:/opt/my-stardog-install/bin"
```

Fourth, start the Stardog server. By default the server will HTTP on port 5820.

```
$ stardog-admin server start
```

Fifth, create a database with an input file:

```
$ stardog-admin db create -n myDB /path/to/some/data.ttl
```

Sixth, query the database:

```
$ stardog query myDB "SELECT DISTINCT ?s WHERE { ?s ?p ?o } LIMIT 10"
```

You can use Stardog Studio to search or query the new database you created by connecting to the http://localhost:5820 (http://localhost:5820) endpoint.

Windows

Windows…really? Okay, but don't blame us if this hurts…The following steps are carried out using the Windows command prompt which you can find under **Start** › **Programs** › **Accessories** › **Command Prompts** or **Start** › **Run** › `cmd`.

First, tell Stardog where its home directory (where databases and other files will be stored) is:

```
> SET STARDOG_HOME=C:\data\stardog
```

Second, copy the `stardog-license-key.bin` into the right place:

```
> COPY /B stardog-license-key.bin %STARDOG_HOME%
```

The `/B` is required to perform a binary copy or the license file may get corrupted. Of course `stardog-license-key.bin` has to be readable by the Stardog process. Finally, Stardog won't run without a valid `stardog-license-key.bin` in `STARDOG_HOME`.

Third, optionally, place the `bin` folder of the Stardog install on your PATH so the `stardog.bat` and `stardog-admin.bat` scripts can be used regardless of current working directory

```
> SET PATH=%PATH%;C:\stardog-5.2.2\bin
```

Fourth, start the Stardog server. By default the server will expose HTTP on port 5820.

```
> stardog-admin.bat server start
```

This will start the server in the current command prompt, you should leave this window open and open a new command prompt window to continue.

Fifth, create a database with some input file:

```
> stardog-admin.bat db create -n myDB C:\path\to\some\data.ttl
```

Sixth, query the database:

```
> stardog.bat query myDB "SELECT DISTINCT ?s WHERE { ?s ?p ?o } LIMIT 10"
```

You can use Stardog Studio to search or query the new database you created by connecting to the http://localhost:5820 (http://localhost:5820) endpoint.

USING STARDOG

Stardog supports SPARQL (http://www.cambridgesemantics.com/semantic-university/sparql-by-example), the W3C standard for querying RDF graphs, as well as a range of other capabilities, including updating, versioning exporting, searching, obfuscating, and browsing graph data.

Querying

Stardog supports SPARQL 1.1 and also the (https://www.w3.org/TR/sparql11-entailment/ (https://www.w3.org/TR/sparql11-entailment/)) [OWL 2 Direct Semantics entailment regime].

To execute a SPARQL query against a Stardog database, use the `query` subcommand with a query string, a query file, or the name of a stored query:

```
$ stardog query myDb "select * where { ?s ?p ?o }"
```

Any SPARQL query type (`SELECT`, `CONSTRUCT`, `DESCRIBE`, `PATHS`, `ASK` or any update query type) can be executed using this command.

By default, all Stardog CLI commands assume the server is running on the same machine as the client using port 5820. But you can interact with a server running on another machine using a full connection string (#_how_to_make_a_connection_string):

```
$ stardog query http://myHost:9090/myDb "select * where { ?s ?p ?o }"
```

Detailed information on using the query command in Stardog can be found on its `man page` (/docs/5.3.6/man/query-execute). See Managing Stored Queries (#_managing_stored_queries) section for configuration, usage, and details of stored queries.

Path Queries

Stardog extends SPARQL for path queries which can be used to find paths between two nodes in a graph. Path queries are similar to SPARQL property paths (https://www.w3.org/TR/sparql11-query/#propertypaths) that recursively traverse a graph and find two nodes connected via a complex path of edges. But SPARQL property paths only return the start and end nodes of a path. **Stardog path queries return all the intermediate nodes on the path and allow arbitrary SPARQL patterns to be used in the query.**

Here's a simple path query to find how `Alice` and `Charlie` are connected to each other:

```
$ stardog query exampleDB "PATHS START ?x = :Alice END ?y = :Charlie VIA ?p"

+----------+------------+----------+
|    x     |     p      |    y     |
+----------+------------+----------+
| :Alice   | :knows     | :Bob     |
| :Bob     | :worksWith | :Charlie |
|          |            |          |
| :Alice   | :worksWith | :Carol   |
| :Carol   | :knows     | :Charlie |
+----------+------------+----------+

Query returned 2 paths in 00:00:00.056
```

Each row of the result table shows one edge. Adjacent edges are printed on subsequent rows of the table. Multiple paths in the results are separated by an empty row.

Path queries by default return only the shortest paths. See the Path Queries (#_path_queries_2) chapter for details about finding different kinds of paths, e.g. all paths (not just shortest ones), paths between all nodes, and cyclic paths.

DESCRIBE

SPARQL provides a `DESCRIBE` query type that returns a subgraph containing information about a resource:

```
DESCRIBE <theResource>
```

SPARQL's `DESCRIBE` keyword is deliberately underspecified. In Stardog, by default, a `DESCRIBE` query retrieves all the triples for which `<theResource>` is the subject. There are, of course, about seventeen thousand other ways to implement `DESCRIBE`. Starting with Stardog 5.3, we are providing two additional describe strategies out of the box. The desired describe strategy can be selected by using a special query hint (#_using_query_hints). For example, the following query will return all the triples where `theResource` is either the subject or the object:

```
#pragma describe.strategy bidirectional

DESCRIBE <theResource>
```

The other built-in describe strategy returns the CBD - Concise Bounded Description (https://www.w3.org/Submission/CBD/) of the given resource:

```
#pragma describe.strategy cbd

DESCRIBE <theResource>
```

The default describe strategy can be changed by setting the `query.describe.strategy` database configuration option (#_configuration_options). Finally, it is also possible to implement a custom describe strategy by implementing a simple Java interface. An example can be found in the stardog examples repo (https://github.com/stardog-union/stardog-examples/tree/develop/examples/describe).

Query Functions

Stardog supports all of the functions from the SPARQL spec (https://www.w3.org/TR/sparql11-query/#SparqlOps), as well as some others from XPath and SWRL. See SPARQL Query Functions (#_sparql_query_functions) for a complete list of built-in functions supported.

Any of the supported functions can be used in queries or rules. Note that, some functions appear in multiple namespaces, but using any of the namespaces will work. Namespaces can be omitted when calling functions too.[2 (#_footnote_2)] XPath comparison (https://www.w3.org/TR/xpath-functions-30/#comp.datetime) and arithmetic (https://www.w3.org/TR/xpath-functions-30/#dateTime-arithmetic) operators on duration, date and time values are supported by overloading the corresponding SPARQL operators such as `=`, `>`, `+`, `-`, etc.

In addition to the built-in functions, new functions can be defined by assigning a new name to a SPARQL expression. These function definitions can either be defined inline in a query or stored in the system (#_managing_stored_functions) and used in any query or rule. Finally, custom function implementations can be implemented in a JVM-compatible language and registered in the system. See the query functions (#_query_functions_2) section for more details.

Federated Queries

In Stardog 3.0 we added support for the SERVICE (http://www.w3.org/TR/sparql11-federated-query/) keyword[3 (#_footnote_3)] which allows users to query distributed RDF via SPARQL-compliant data sources. You can use this to federate queries between several Stardog databases or Stardog and other public endpoints.

As of 5.2 Stardog supports service variables (https://www.w3.org/TR/sparql11-federated-query/#variableService). This allows dynamic selection of endpoints for federated queries, for example:

```
{
  ?service a :MyService .

  SERVICE ?service { ... }
}
```

Stardog ships with a default `Service` implementation which uses SPARQL Protocol to send the service fragment to the remote endpoint and retrieve the results. Any endpoint that conforms to the SPARQL protocol can be used.

The Stardog SPARQL endpoint is `http://<server>:<port>/{db}/query (http://<server>:<port>/{db}/query)`.

HTTP Authentication

Stardog requires authentication. If the endpoint you're referencing with the `SERVICE` keyword requires HTTP authentication, credentials are stored in a password file (#_using_a_password_file) called `services.sdpass` located in `STARDOG_HOME` directory. The default `Service` implementation assumes HTTP BASIC authentication; for services that use DIGEST auth, or a different authentication mechanism altogether, you'll need to implement a custom `Service` implementation.

Querying RDBMS via Virtual Graphs

Sometimes enterprise information belongs to or in a Stardog graph but for various reasons the upstream or canonical data source should not be disturbed, consolidated, or decommissioned.

In those (and other) cases it makes sense to use Stardog's R2RML[4 (#_footnote_4)] mapping feature to create virtual graphs, that is, mappings between a Stardog graph and external data sources such that Stardog will query the external data sources—by rewriting SPARQL to SQL automatically—as if they were actually materialized in Stardog.

See the Enterprise Data Unification (#_enterprise_data_unification) chapter for configuration and usage.

Querying Geospatial Data

This works more or less like any SPARQL query, as long as the right geospatial vocabulary is used in the data.

See the Geospatial Query (#_geospatial_query) chapter for configuration and usage.

Traversing

Stardog supports graph traversals via Gremlin Console and TinkerPop 3 APIs. You can write Gremlin traversals against TP3 APIs in many different languages. But the easiest way to use Gremlin with Stardog is via the Gremin Console (#_stardog_gremlin_console).

Updating

There are many ways to update the data in a Stardog database; two of the most commonly used methods are the CLI and SPARQL Update queries, each of which are discussed below.

SPARQL Update

SPARQL 1.1 Update can be used to insert RDF into or delete RDF from a Stardog database using SPARQL query forms `INSERT` and `DELETE`, respectively.

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
INSERT DATA
{
  GRAPH <http://example/bookStore> { <http://example/book1>  ns:price  42 }
}
```

An example of deleting RDF:

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>

DELETE DATA
{
  <http://example/book2> dc:title "David Copperfield" ;
                         dc:creator "Edmund Wells" .
}
```

Or they can be combined with `WHERE` clauses:

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>

WITH <http://example/addresses>
DELETE { ?person foaf:givenName 'Bill' }
INSERT { ?person foaf:givenName 'William' }
WHERE
  { ?person foaf:givenName 'Bill' }
```

NOTE Per the SPARQL Update spec, Stardog treats Update queries as implicitly transactional and atomic. Since Stardog does not support nested transactions, it will not (currently) support an Update query in an open transaction.[5]

Adding Data with the CLI

As of Stardog 5.3.6, the most efficient way to load data into Stardog is at database creation time. See the Creating a Database (#_creating_a_database) section for bulk loading data at database creation time. To add data to an existing Stardog database, use the `add` (/docs/5.3.6/man/data-add) command:

```
$ stardog data add myDatabase 1.rdf 2.rdf 3.rdf
```

The optional arguments are `-f` (or `--format`) to specify the RDF serialization type of the files to be loaded; if you specify the wrong type, `add` will fail. If you don't specify a type, Stardog will try to determine the type on its own based on the file extension. For example, the files that have names ending with '.ttl' will be parsed with Turtle syntax. If you specify a type, then all the files being loaded must of that same type.

If you want to add data to a named graph, specify it via the `--named-graph` or `-g` options.

Removing Data with the CLI

To remove data, use `remove` (/man/data-remove.html) with the following input(s):

1. one Named Graph, *or*
2. one or more files containing RDF (in some recognized serialization format, i.e., RDF/XML, Turtle, Trig), *or*
3. one Named Graph and one or more RDF files.

For example,

```
$ stardog data remove -g http://foo myDatabase
```

will remove the named graph `http://foo` and all its triples from `myDatabase`.

```
$ stardog data remove myDatabase 1.rdf
```

will remove the triples in `1.rdf` from (the default graph of) `myDatabase`.

```
$ stardog data remove -g http://foo -f TURTLE myDatabase 2.rdf 3.rdf
```

will remove the triples in the Turtle files `2.rdf` and `3.rdf` from the named graph `http://foo` of `myDatabase`.

How Stardog Handles RDF Parsing

RDF parsing in Stardog is strict: it requires typed RDF literals to match their explicit datatypes, URIs to be well-formed, etc. In some cases, strict parsing isn't ideal—it may be disabled using the `strict.parsing` configuration option (#_configuration_options).

However, even with strict parsing disabled, Stardog's RDF parser may encounter parse errors from which it cannot recover. And loading data in lax mode may lead to unexpected SPARQL query results. For example, malformed literals (`"2.5"^^xsd:int`) used in filter evaluation may lead to undesired results.

Versioning

Stardog supports graph change management capability that lets users track changes between revisions of a Stardog database, add comments and other metadata to the revisions, extract diffs between those revisions, tag revisions with labels, and query over the revision history of the database using SPARQL.

For a more detailed tutorial on using Stardog versioning, see the stardog-examples repo (https://github.com/Complexible/stardog-examples/blob/master/examples/cli/versioning/README.md).

Versioning support for a database is disabled by default but can be enabled at any time by setting the configuration option `versioning.enabled` to true. For example, you can create a database with versioning support as follows:

```
$ stardog-admin db create -o versioning.enabled=true -n myDb
```

This option can also be set after database creation using the `stardog-admin metadata set` command.

The following examples give a **very brief overview** of this capability; see the Man Pages (#_man_pages) for all the details.

Committing Changes

Commit a new version by adding and removing triples specified in files. Different from the `data add/remove` commands, `commit` allows one to add and remove triples in one commit and to associate a commit message.

To commit changes:

```
$ stardog vcs commit --add add_file1.ttl add_file2.ttl --remove remove_file.ttl -m "This is an example
commit" myDb
```

It is also possible to run an update query with a commit message:

NOTE Removals are performed before additions and queries are executed last.

```
$ stardog vcs commit --query update.sparql -m "This is another commit" myDb
```

If the database is updated through the regular `data add`, `data remove`, or `query` commands when versioning is enabled, a corresponding version will be created but the commit message will be empty.

Viewing Revisions

To see all revisions (commits) in a database:

```
$ stardog vcs list myDb
$ stardog vcs list --committer userName myDb
```

The output can be tweaked using `--after`, `--before`, and `--committer`.

Reverting Revisions

You can revert specific revisions, ranges, etc.

```
$ stardog vcs revert myDb
$ stardog vcs revert myDb de44369d-cc7b-4244-a3fb-3f6e271420b0
```

Viewing Diffs

You can also see the differences between revisions; by default, between the head version and its previous versions or the changes in a specific commit, respectively:

```
$ stardog vcs diff myDb
$ stardog vcs diff myDb de44369d-cc7b-4244-a3fb-3f6e271420b0
```

NOTE Diffs are represented as SPARQL Update queries so that they may be used as a kind of graph patch.

Using Tags

You can also create, drop, list tags, i.e., named revisions:

```
$ stardog vcs tag --list myDb
```

Querying the Revision History

The revision history of the database is represented as RDF using an extended version of the W3C PROV vocabulary and can be queried using SPARQL.[6 (#_footnote_6)] The following query retrieves all the versions:

```
SELECT ?author ?msg ?date {
  ?v a vcs:Version ;
     rdfs:comment ?msg ;
     prov:wasAttributedTo / rdfs:label ?author ;
     prov:generatedAtTime ?date
}
```

You can execute versioning queries using the `vcs query` command:

```
$ stardog vcs query myDb versioning-query.sparql
```

Queries executed by `vcs query` only query the versioning history and automatically recognizes the `vcs` and `prov` namespaces. If you would like to query the current state of the database along with the version history then you can run a regular query using a special versioning `SERVICE`. The following query finds the instances of `foaf:Person` class that has been modified in the last 24 hours:

```
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX vcs: <tag:stardog:api:versioning:>

SELECT DISTINCT ?x ?date {
  ?x a foaf:Person

  SERVICE vcs:service {
    ?v a vcs:Version;
       prov:generatedAtTime ?date;
       vcs:updates/(vcs:additions|vcs:removals) ?update
    GRAPH ?update {
      ?x ?p ?o
    }
    FILTER (NOW() - ?date < 'PT24H'^^xsd:dayTimeDuration)
  }
}
ORDER BY DESC(?date)
```

You can run this query using the regular query command:

```
$ stardog query myDb versioning-mixed-query.sparql
```

Note that, in this query the namespaces `vcs` and `prov` have been declared explicitly in the query. If you register these namespaces (#_namespacing) to your database then you can omit these declarations.

For more details about the Stardog versioning representation, see the stardog-examples repo (https://github.com/Complexible/stardog-examples/blob/master/examples/cli/versioning/README.md).

Exporting

To export data from a Stardog database back to RDF, `export` (/man/data-export.html) is used by specifying

1. the name of the database to export
2. optionally, the URIs of the named graphs to export if you wish to export specific named graphs only. Keywords `DEFAULT` and `ALL` can be used as values of the `--named-graph` parameter to export the default graph and all graphs, respectively
3. the export format: `N-TRIPLES, RDF/XML, TURTLE, TRIG`. The default is `N-TRIPLES`. `TRIG` must be used when exporting the entire database if the database contains triples inside named graphs
4. a file to export to

For example,

```
$ stardog data export --format TURTLE myDatabase myDatabase_output.ttl

$ stardog data export --named-graph http://example.org/context myDatabase myDatabase_output.nt
```

Searching

Stardog includes an RDF-aware semantic search capability: it will index RDF literals and supports information retrieval-style queries over indexed data. See [Managing Search] (#Managing Search) for more details.

Searching with the Command Line

First, check out the `search` man page (/man/query-search.html):

```
$ stardog help query search
```

Okay, now let's do a search over the O'Reilly book catalog in RDF for everything mentioning "html":

```
$ stardog query search -q "html" -l 10 catalog
```

The results?

```
+-------+-------+---------------------------------------------------+
| Index | Score |                        Hit                        |
+-------+-------+---------------------------------------------------+
| 1     | 6.422 | urn:x-domain:oreilly.com:product:9780596002251.IP |
| 2     | 6.422 | urn:x-domain:oreilly.com:product:9780596002961.IP |
| 3     | 6.422 | urn:x-domain:oreilly.com:product:9780596003166.IP |
| 4     | 6.422 | urn:x-domain:oreilly.com:product:9780596101978.IP |
| 5     | 6.422 | urn:x-domain:oreilly.com:product:9780596154066.IP |
| 6     | 6.422 | urn:x-domain:oreilly.com:product:9780596157616.IP |
| 7     | 6.422 | urn:x-domain:oreilly.com:product:9780596527273.IP |
| 8     | 6.422 | urn:x-domain:oreilly.com:product:9780596527402.IP |
| 9     | 6.422 | urn:x-domain:oreilly.com:product:9780596805876.IP |
| 10    | 6.422 | urn:x-domain:oreilly.com:product:9781565924949.IP |
+-------+-------+---------------------------------------------------+
```

See Search (#_search) for more details.

Obfuscating

When sharing sensitive RDF data with others, you might want to (selectively) obfuscate it so that sensitive bits are not present, but non-sensitive bits remain. For example, this feature can be used to submit Stardog bug reports using sensitive data.

Data obfuscation works much the same way as the `export` command and supports the same set of arguments:

```
$ stardog data obfuscate myDatabase obfDatabase.ttl
```

By default, all URIs, bnodes, and string literals in the database will be obfuscated using the SHA256 message digest algorithm. Non-string typed literals (numbers, dates, etc.) are left unchanged as well as URIs from built-in namespaces (`RDF`, `RDFS`, and `OWL`). It's possible to customize obfuscation by providing a configuration file.

```
$ stardog data obfuscate --config obfConfig.ttl myDatabase  obfDatabase.ttl
```

The configuration specifies which URIs and strings will be obfuscated by defining inclusion and exclusion filters. See the example configuration file in the stardog-examples Github repo. (https://github.com/Complexible/stardog-examples/blob/master/config/obfuscation.ttl)

Once the data is obfuscated, queries written against the original data will no longer work. Stardog provides query obfuscation capability, too, so that queries can be executed against the obfuscated data. If a custom configuration file is used to obfuscate the data, then the same configuration should be used for obfuscating the queries as well:

```
$ stardog query obfuscate --config obfConfig.ttl myDatabase myQuery.sparql > obfQuery.ttl
```

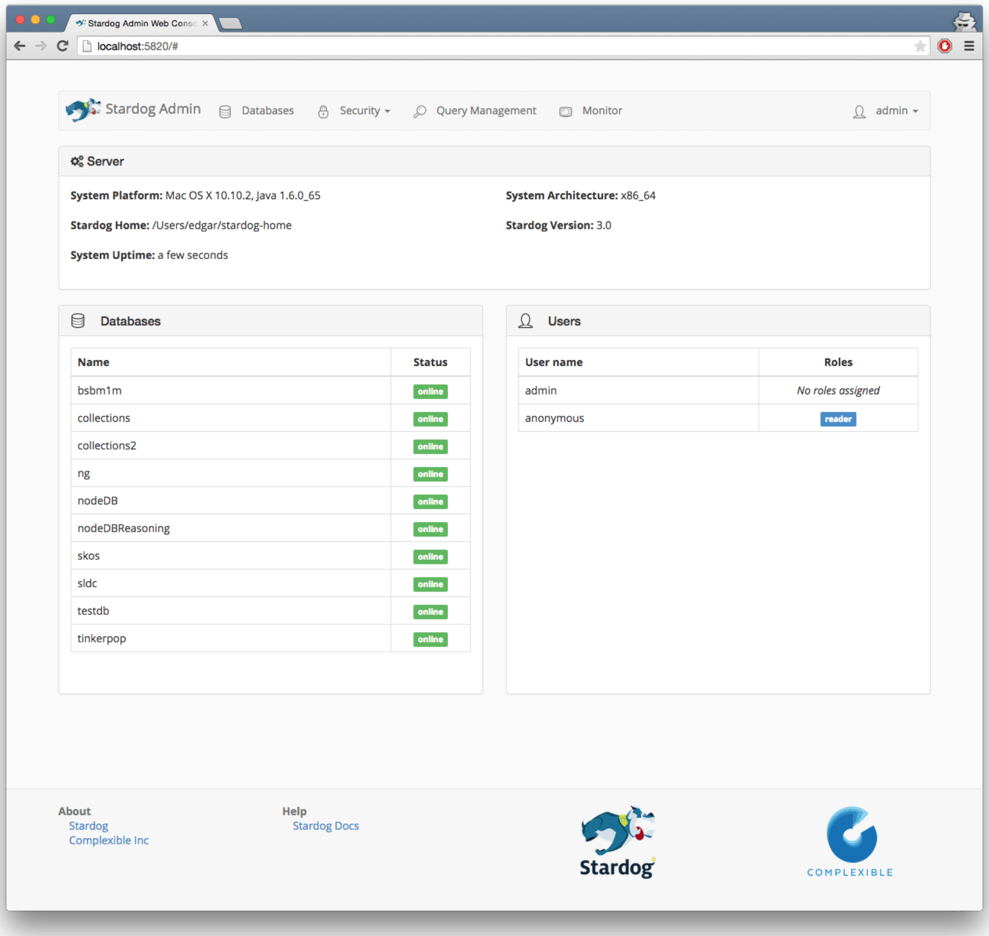
Browsing

The Stardog Web Console is a responsive web app for the Stardog Server and for every Stardog database that makes administration and interaction with data quick and easy; you can access it at `http://foo:5820 (http://foo:5820)` where `foo` is the name of the machine where Stardog is running.

A Screenshot Tour…

Seriously, this is a lot more fun if you just download (http://stardog.com/) the damn thing and hit it with a browser!

Web Console



Server

System Platform: Mac OS X 10.10.2, Java 1.6.0_65

System Architecture: x86_64

Stardog Home: /Users/edgar/stardog-home

Stardog Version: 3.0

System Uptime: a few seconds

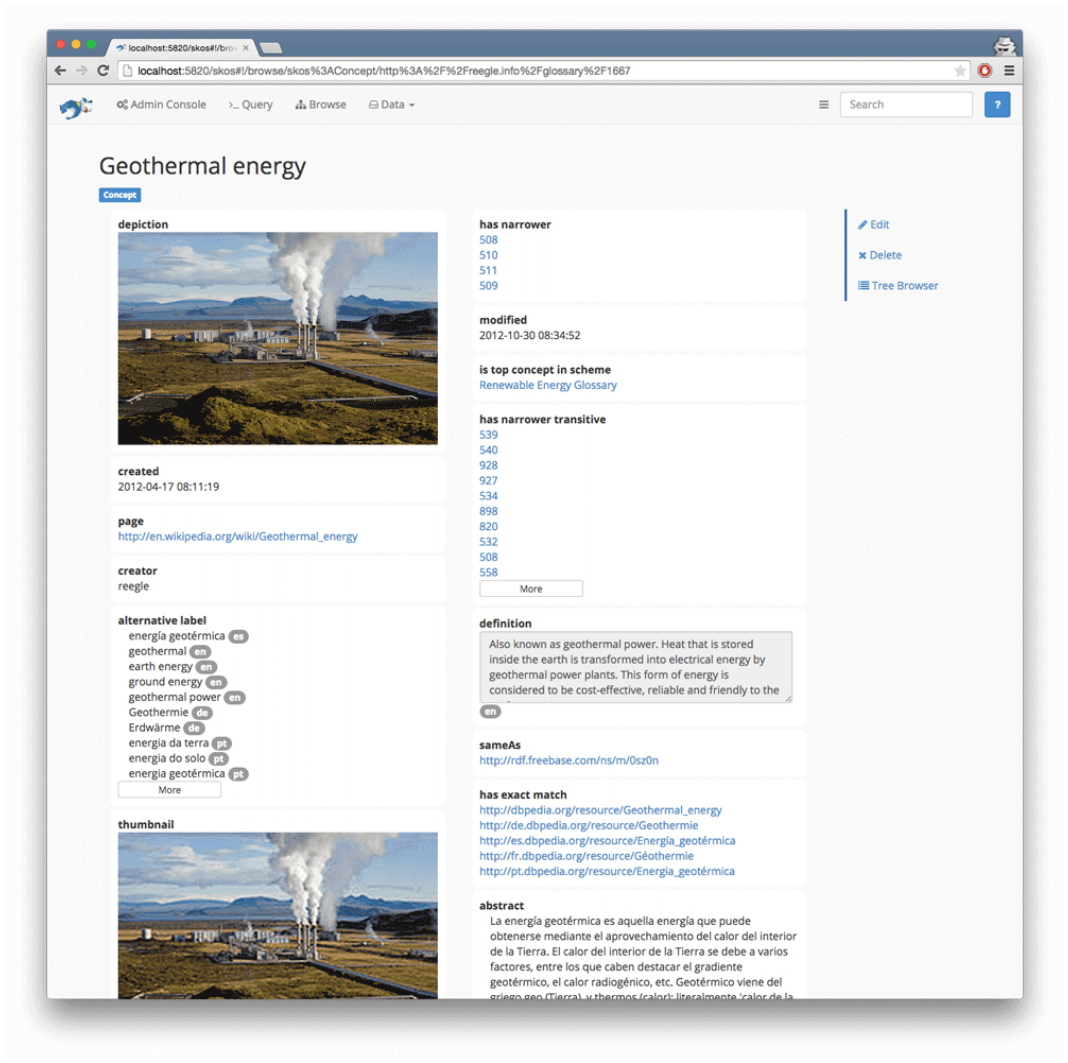
Databases

Name	Status
bsbm1m	online
collections	online
collections2	online
ng	online
nodeDB	online
nodeDBReasoning	online
skos	online
sldc	online
testdb	online
tinkerpop	online

Users

User name	Roles
admin	*No roles assigned*
anonymous	reader

Browsing the Graph


localhost:5820/skos#!/bro
localhost:5820/skos#!/browse/skos%3AConcept/http%3A%2F%2Freegle.info%2Fglossary%2F1667
Admin Console
>_ Query
Browse
Data
Search
?
Edit
Delete
Tree Browser
More

Geothermal energy

Concept

depiction

created
2012-04-17 08:11:19

page
http://en.wikipedia.org/wiki/Geothermal_energy

creator
reegle

alternative label
energía geotérmica es
geothermal en
earth energy en
ground energy en
geothermal power en
Geothermie de
Erdwärme de
energia da terra pt
energia do solo pt
energia geotérmica pt

thumbnail

has narrower
508
510
511
509

modified
2012-10-30 08:34:52

is top concept in scheme
Renewable Energy Glossary

has narrower transitive
539
540
928
927
534
898
820
532
508
558

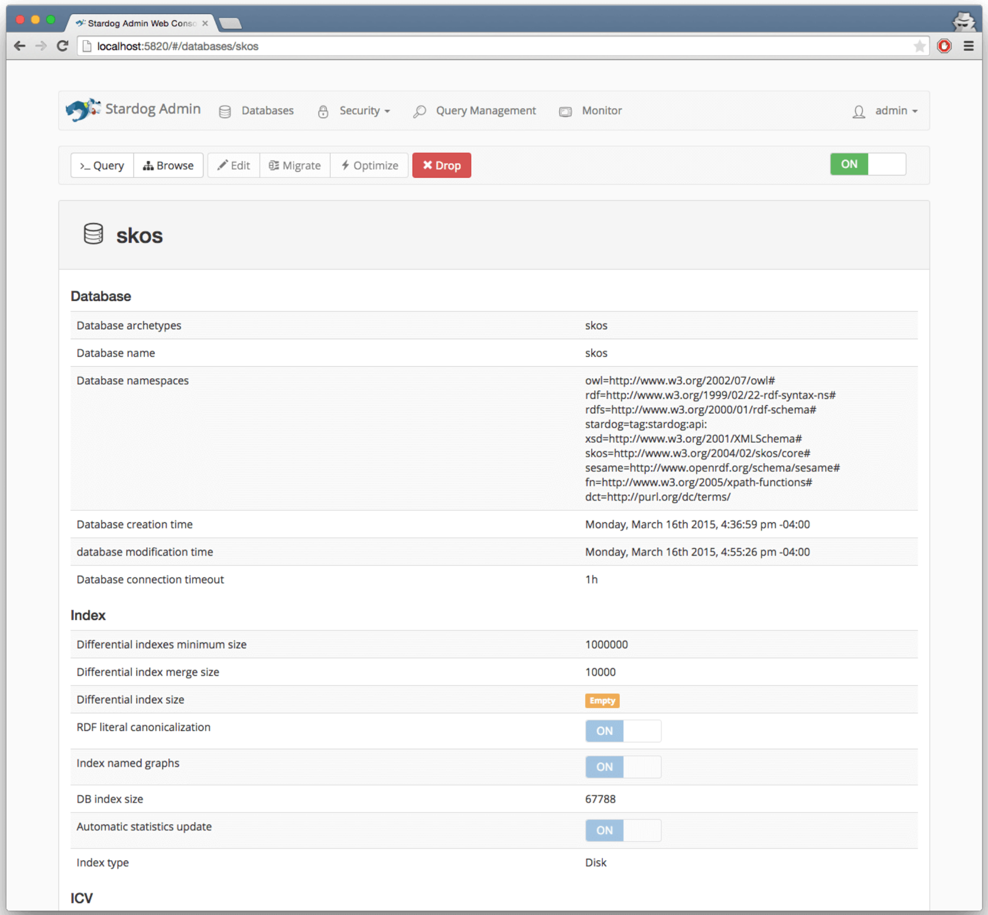
definition
Also known as geothermal power. Heat that is stored inside the earth is transformed into electrical energy by geothermal power plants. This form of energy is considered to be cost-effective, reliable and friendly to the
en

sameAs
http://rdf.freebase.com/ns/m/0sz0n

has exact match
http://dbpedia.org/resource/Geothermal_energy
http://de.dbpedia.org/resource/Geothermie
http://es.dbpedia.org/resource/Energía_geotérmica
http://fr.dbpedia.org/resource/Géothermie
http://pt.dbpedia.org/resource/Energia_geotérmica

abstract
La energía geotérmica es aquella energía que puede obtenerse mediante el aprovechamiento del calor del interior de la Tierra. El calor del interior de la Tierra se debe a varios factores, entre los que caben destacar el gradiente geotérmico, el calor radiogénico, etc. Geotérmico viene del griego geo (Tierra), y thermos (calor); literalmente 'calor de la

Managing a Database



skos

Database

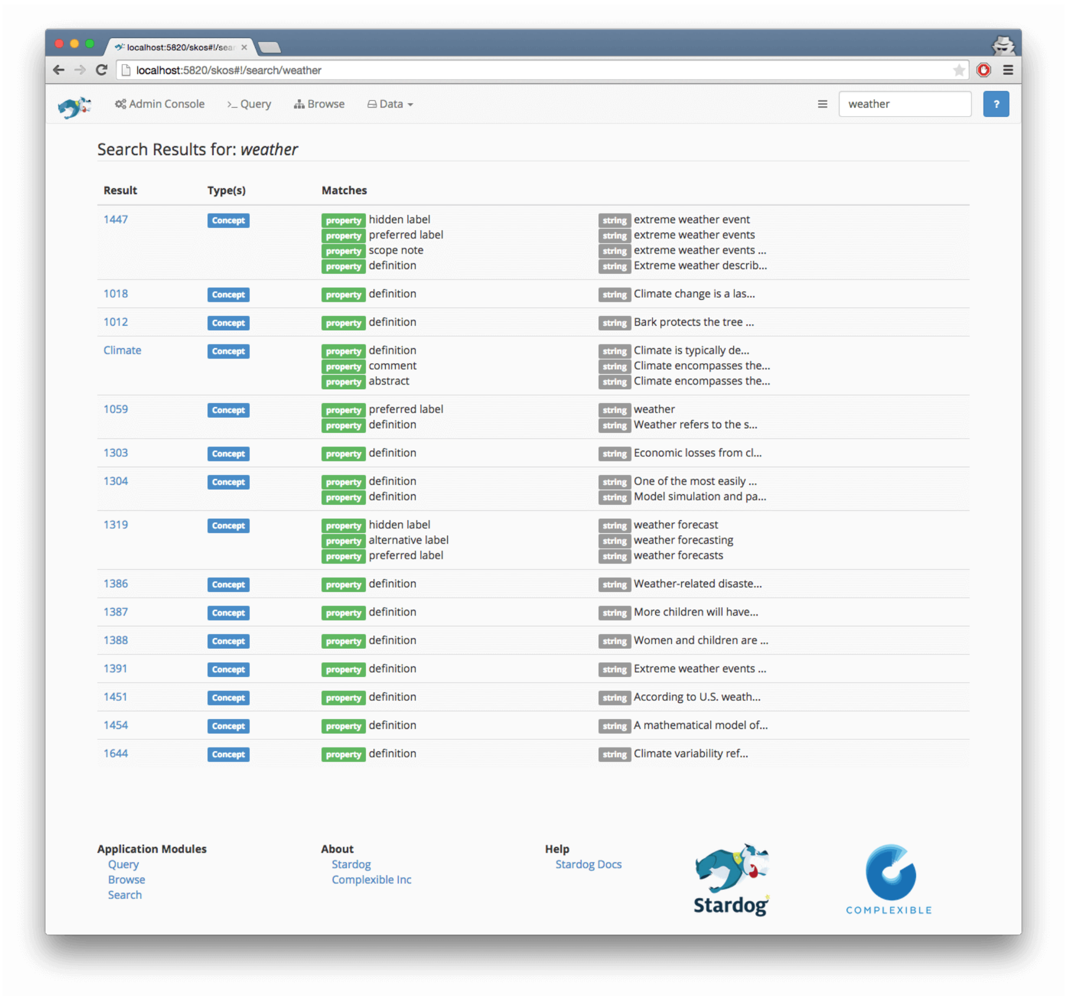
Database archetypes	skos
Database name	skos
Database namespaces	owl=http://www.w3.org/2002/07/owl# rdf=http://www.w3.org/1999/02/22-rdf-syntax-ns# rdfs=http://www.w3.org/2000/01/rdf-schema# stardog=tag:stardog:api: xsd=http://www.w3.org/2001/XMLSchema# skos=http://www.w3.org/2004/02/skos/core# sesame=http://www.openrdf.org/schema/sesame# fn=http://www.w3.org/2005/xpath-functions# dct=http://purl.org/dc/terms/
Database creation time	Monday, March 16th 2015, 4:36:59 pm -04:00
database modification time	Monday, March 16th 2015, 4:55:26 pm -04:00
Database connection timeout	1h

Index

Differential indexes minimum size	1000000
Differential index merge size	10000
Differential index size	Empty
RDF literal canonicalization	ON
Index named graphs	ON
DB index size	67788
Automatic statistics update	ON
Index type	Disk

ICV

Searching the Graph


localhost:5820/skos#!/sea
localhost:5820/skos#!/search/weather
Admin Console
Query
Browse
Data
weather

Search Results for: *weather*

Result	Type(s)	Matches	
1447	Concept	property hidden label property preferred label property scope note property definition	string extreme weather event string extreme weather events string extreme weather events ... string Extreme weather describ...
1018	Concept	property definition	string Climate change is a las...
1012	Concept	property definition	string Bark protects the tree ...
Climate	Concept	property definition property comment property abstract	string Climate is typically de... string Climate encompasses the... string Climate encompasses the...
1059	Concept	property preferred label property definition	string weather string Weather refers to the s...
1303	Concept	property definition	string Economic losses from cl...
1304	Concept	property definition property definition	string One of the most easily ... string Model simulation and pa...
1319	Concept	property hidden label property alternative label property preferred label	string weather forecast string weather forecasting string weather forecasts
1386	Concept	property definition	string Weather-related disaste...
1387	Concept	property definition	string More children will have...
1388	Concept	property definition	string Women and children are ...
1391	Concept	property definition	string Extreme weather events ...
1451	Concept	property definition	string According to U.S. weath...
1454	Concept	property definition	string A mathematical model of...
1644	Concept	property definition	string Climate variability ref...

Namespacing

Stardog allows users to store and manage custom namespace prefix bindings for each database. These stored namespaces allow users to omit prefix declarations in Turtle files and SPARQL queries. Namespace Prefix Bindings (#_namespace_prefix_bindings) section describes how to manage these namespace prefixes in detail.

Stored namespaces allow one to use Stardog without declaring a single namespace prefix. Stardog will use its default namespace (`http://api.stardog.com/ (http://api.stardog.com/)`) behind the scenes so that everything will still be valid RDF, but users won't need to deal with namespaces manually. Stardog will act **as if** there are no namespaces, which in some cases is exactly what you want!

For example, let's assume we have some data that does not contain any namespace declarations:

```
:Alice a :Person ;
       :knows :Bob .
```

We can create a database using this file directly:

```
$ stardog-admin db create -n mydb data.ttl
```

We can also add this file to the database after it is created. After the data is loaded, we can then execute SPARQL queries without prefix declarations:

```
$ stardog query mydb "SELECT * { ?person a :Person }"
+--------+
| person |
+--------+
| :Alice |
+--------+

Query returned 1 results in 00:00:00.111
```

NOTE Once we export the data from this database, the default (i.e., in-built) prefix declarations will be printed, but otherwise we will get the same serialization as in the original data file:

```
$ stardog data export mydb
@prefix : <http://api.stardog.com/> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix stardog: <tag:stardog:api:> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:Alice a :Person ;
       :knows :Bob .
```

Naming

Stardog includes aliases for several commonly used sets of named graphs. These non-standard extensions are provided for convenience and can be used wherever named graph IRIs are expected. This includes SPARQL queries & updates, property graph operations and configuration values. Following is a list of special named graph IRIs.

Named Graph IRI	Refers to
`tag:stardog:api:context:default`	the default (no) context graph
`tag:stardog:api:context:all`	all contexts, including the default (no) context graph
`tag:stardog:api:context:named`	all named graphs, excluding the default graph

Deploying

As of Stardog 5, we support both AWS and Pivotal Cloud Foundry as first-class environments.

Stardog Graviton

Configuring and managing highly available cluster applications can be a complex black art. Graviton is a tool that leverages the power of Amazon Web Services (http://aws.amazon.com) to make launching the Stardog cluster easy.

The source code (http://github.com/stardog-union/stardog-graviton) is available as Apache 2.0 licensed code.

Download

- Linux
- OSX

Requirements

- A Stardog release zip file (5.0 or later).
- A Stardog license.
- An AWS (http://aws.amazon.com/) account.
- terraform (http://terraform.io/) 0.8.8.
- packer (https://www.packer.io/) 0.12.3.

Setup Your Environment

In order to use `stardog-graviton` in its current form the following environment variables must be set.

```
AWS_ACCESS_KEY_ID=<a valid aws access key>
AWS_SECRET_ACCESS_KEY=<a valid aws secret key>
```

The account associated with the access tokens must have the ability to create IAM credentials and full EC2 access.

Both `terraform` and `packer` must be in your system path.

The easiest way to launch a cluster is to run `stardog-graviton` in interactive mode. This will cause the program to ask a series of questions in order to get the needed values to launch a cluster. Here is a sample session:

```
$ stardog-graviton --log-level=DEBUG launch mystardog423
What version of stardog are you launching?: 4.2.3
What is the path to the Stardog release?:
A value must be provided.
What is the path to the Stardog release?: /Users/bresnaha/stardog-4.2.3.zip
There is no base image for version 4.2.3.
- Running packer to build the image...
done
AMI Successfully built: ami-c06246a0
Creating the new deployment mystardog423
Would you like to create an SSH key pair? (yes/no): no
EC2 keyname (default): <aws key name>
Private key path: /path/to/private/key
What is the path to your Stardog license?: /path/to/stardog/license
\ Calling out to terraform to create the volumes...
- Calling out to terraform to stop builder instances...
Successfully created the volumes.
\ Creating the instance VMs......
Successfully created the instance.
Waiting for stardog to come up...
The instance is healthy
Changing the default password...
Password changed successfully for user admin.
\ Opening the firewall......
Successfully opened up the instance.
The instance is healthy
The instance is healthy
Stardog is available here: http://mystardog423sdelb-1763823291.us-west-1.elb.amazonaws.com:5821
ssh is available here: mystardog423belb-124202215.us-west-1.elb.amazonaws.com
Using 3 stardog nodes
        10.0.101.189:5821
        10.0.100.107:5821
        10.0.100.140:5821
Success.
```

To avoid being asked questions a file named `~/.graviton/default.json` can be created. An example can be found in the defaults.json.example (https://github.com/stardog-union/stardog-graviton/blob/master/defaults.json.example) file.

All of the components needed to run a Stardog cluster are considered part of a deployment. Every deployment must be given a name that is unique to each cloud account. In the above example the deployment name is `mystardog2`.

Status

Once the image has been successfully launched its health can be monitored with the `status` command:

```
$ stardog-graviton --log-level=DEBUG status mystardog423
The instance is healthy
Stardog is available here: http://mystardog423sdelb-1763823291.us-west-1.elb.amazonaws.com:5821
ssh is available here: mystardog423belb-124202215.us-west-1.elb.amazonaws.com
Using 3 stardog nodes
        10.0.101.189:5821
        10.0.100.107:5821
        10.0.100.140:5821
Success.
```

Cleanup

AWS EC2 charges by the hour for the VMs that Graviton runs thus when the cluster is no longer in use it is important to clean it up with the `destroy` command.

```
stardog-graviton --log-level=DEBUG destroy mystardog423
This will destroy all volumes and instances associated with this deployment.
Do you really want to destroy? (yes/no): yes
/ Deleting the instance VMs...
Successfully destroyed the instance.
\ Calling out to terraform to delete the images...
Successfully destroyed the volumes.
Success.
```

More information

For more information about Graviton check out the README (https://github.com/stardog-union/stardog-graviton/blob/master/README.md) and the blog post (https://blog.stardog.com/stardog-graviton-aws-made-easy/).

Pivotal Cloud Foundry

As of Stardog 5, we've added support for Pivotal Cloud Foundry. More docs (http://docs.pivotal.io/partners/stardog-service-broker/) are available at PCF.

Our open source service broker (https://github.com/stardog-union/service-broker) adheres to the Open Service Broker API (https://www.openservicebrokerapi.org/) and thus can be used with Cloud Foundry (https://cloudfoundry.org/), Open Shift (https://www.openshift.org/), and Kubernetes (http://kubernetes.io/).

Visualizing

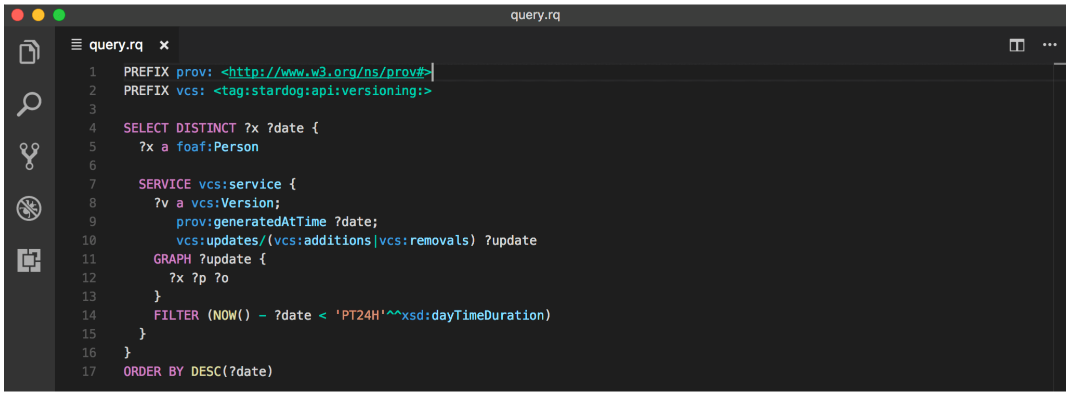
As of Stardog 5 we support Tableau (http://www.tableau.com/) so that you can use its visualization powers to understand and visualize the knowledge graph.

You can get the Stardog Data Connector for Tableau by opening the Tableau Web Connector endpoint for Stardog (http://tableau.stardog.com/) in Tableau.

Developing

Writing anything other than the most basic SPARQL queries is challenging; writing that same query in plain text is punishment. We don't want to punish our beloved users, so we published an IDE extension to make your (and, let's face it, our) lives easier.

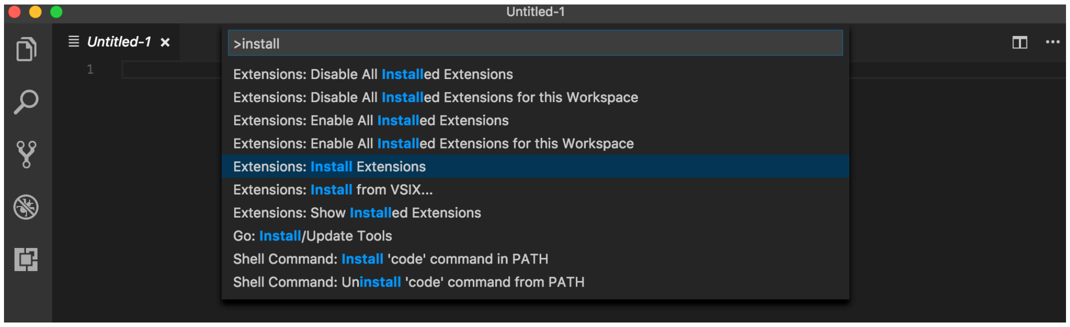
To get beautifully highlighted SPARQL, along with Turtle, Stardog Mapping Syntax, and Stardog Rules Syntax, you'll first need Visual Studio Code. Visual Studio Code (https://code.visualstudio.com/) is an extremely lightweight and extensible code editor, and happens to be the editor of choice among our front-end engineers. It's free, platform-independent, and open source.



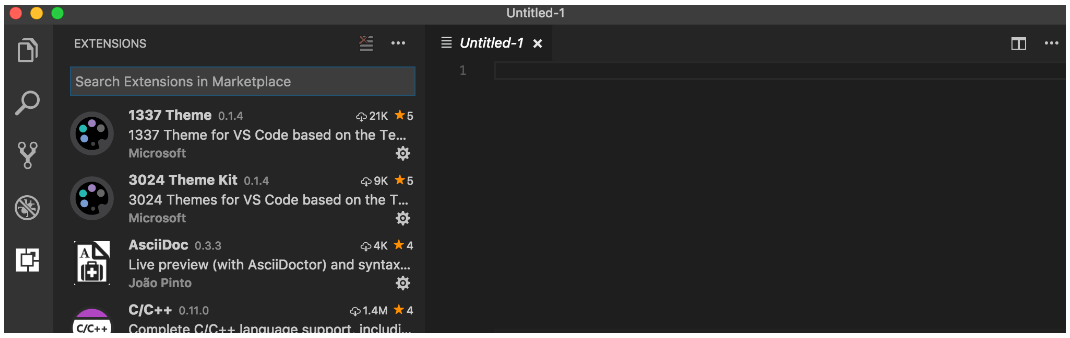
```
PREFIX prov: <http://www.w3.org/ns/prov#>
PREFIX vcs: <tag:stardog:api:versioning:>

SELECT DISTINCT ?x ?date {
  ?x a foaf:Person

  SERVICE vcs:service {
    ?v a vcs:Version;
       prov:generatedAtTime ?date;
       vcs:updates/(vcs:additions|vcs:removals) ?update
    GRAPH ?update {
      ?x ?p ?o
    }
    FILTER (NOW() - ?date < 'PT24H'^^xsd:dayTimeDuration)
  }
}
ORDER BY DESC(?date)
```

Getting Started

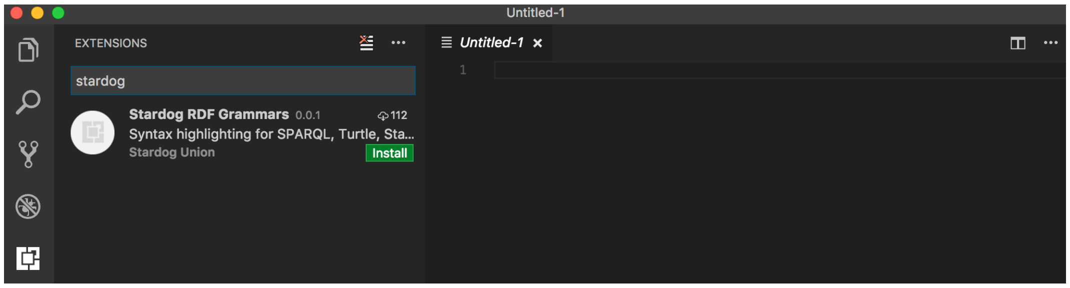
To get started, download (https://code.visualstudio.com/Download) the editor and install it on your OS of choice. After firing up Visual Studio Code, open the command palette with `Cmd` + `Shift` + `P` (`Ctrl` + `Shift` + `P` on Windows) and type "install"; then click [**Extensions: Install Extensions**] .


Untitled-1
Untitled-1
>install
Extensions: Disable All Installed Extensions
Extensions: Disable All Installed Extensions for this Workspace
Extensions: Enable All Installed Extensions
Extensions: Enable All Installed Extensions for this Workspace
Extensions: Install Extensions
Extensions: Install from VSIX...
Extensions: Show Installed Extensions
Go: Install/Update Tools
Shell Command: Install 'code' command in PATH
Shell Command: Uninstall 'code' command from PATH

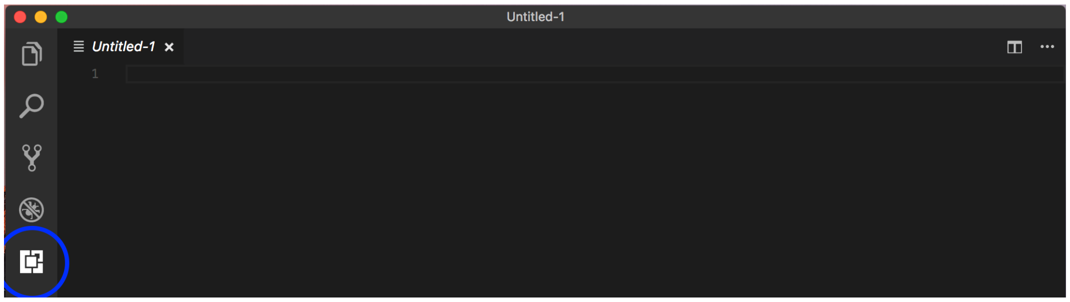
This will take you to the Marketplace tab, where you can search for your favorite enterprise knowledge graph by name: `stardog`.


Untitled-1
EXTENSIONS
Search Extensions in Marketplace
1337 Theme 0.1.4 21K 5
1337 Theme for VS Code based on the Te...
Microsoft
3024 Theme Kit 0.1.4 9K 5
3024 Themes for VS Code based on the T...
Microsoft
AsciiDoc 0.3.3 4K 4
Live preview (with AsciiDoctor) and syntax...
João Pinto
C/C++ 0.11.0 1.4M 4
Complete C/C++ language support, includi...
Untitled-1

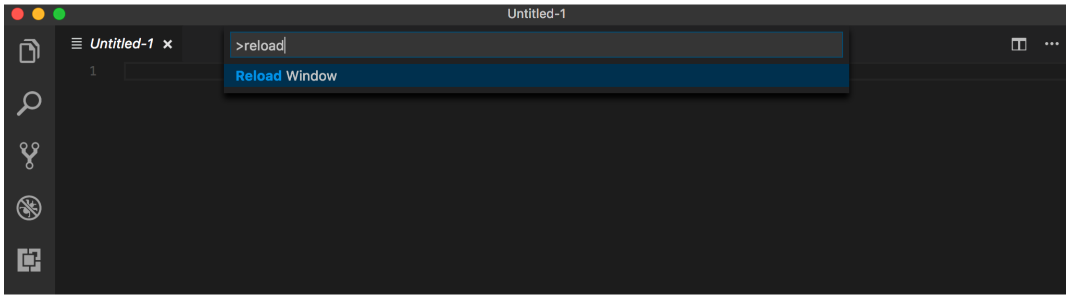
Our SPARQL highlighting extension is called `Stardog RDF Grammars`. Click the [**Install**] button in the search result, and the extension will be downloaded and installed automatically.


Untitled-1
EXTENSIONS
stardog
Stardog RDF Grammars 0.0.1 112
Syntax highlighting for SPARQL, Turtle, Sta...
Stardog Union
Install
Untitled-1

Alternatively, you can simply click the Marketplace icon and search in the same way.



You'll have to reload the window to initialize the extension, which you can do by quitting and restarting Visual Studio Code, or simply by opening the command palette again, typing "reload", and clicking [**Reload Window**] .



Once you've reloaded the window, the extension should be initialized and ready to highlight all the SPARQL and Turtle you can throw at it!

Features

The extension will automatically detect and provide highlighting for the following file types:

Language	File extensions
SPARQL	`.sparql`, `.rq`
Turtle	`.turtle`, `.ttl`
Stardog Mapping Syntax	`.sms`
Stardog Rules Syntax	`.srs`

Man Pages

Stardog CLI

- `data add` (/docs/man/data-add), `data export` (/docs/man/data-export), `data obfuscate` (/docs/man/data-obfuscate), `data remove` (/docs/man/data-remove), `data size` (/docs/man/data-size)
- `doc count` (/docs/man/doc-count), `doc delete` (/docs/man/doc-delete), `doc get` (/docs/man/doc-get), `doc put` (/docs/man/doc-put), `doc reindex` (/docs/man/doc-reindex)
- `file cat` (/docs/man/file-cat), `file obfuscate` (/docs/man/file-obfuscate), `file split` (/docs/man/file-split)
- `graphql execute` (/docs/man/graphql-execute), `graphql explain` (/docs/man/graphql-explain), `graphql schema` (/docs/man/graphql-schema)
- `icv convert` (/docs/man/icv-convert), `icv explain` (/docs/man/icv-explain), `icv export` (/docs/man/icv-export), `icv fix` (/docs/man/icv-fix), `icv validate` (/docs/man/icv-validate)
- `namespace add` (/docs/man/namespace-add), `namespace export` (/docs/man/namespace-export), `namespace import` (/docs/man/namespace-import), `namespace list` (/docs/man/namespace-list), `namespace remove` (/docs/man/namespace-remove)
- `query execute` (/docs/man/query-execute), `query explain` (/docs/man/query-explain), `query obfuscate` (/docs/man/query-obfuscate), `query search` (/docs/man/query-search)
- `reasoning consistency` (/docs/man/reasoning-consistency), `reasoning explain` (/docs/man/reasoning-explain), `reasoning schema` (/docs/man/reasoning-schema), `reasoning undo` (/docs/man/reasoning-undo)
- `tx begin` (/docs/man/tx-begin), `tx commit` (/docs/man/tx-commit), `tx list` (/docs/man/tx-list), `tx rollback` (/docs/man/tx-rollback)
- `vcs commit` (/docs/man/vcs-commit), `vcs diff` (/docs/man/vcs-diff), `vcs list` (/docs/man/vcs-list), `vcs query` (/docs/man/vcs-query), `vcs revert` (/docs/man/vcs-revert), `vcs tag` (/docs/man/vcs-tag)

Stardog Admin CLI

- `cluster generate` (/docs/man/cluster-generate), `cluster info` (/docs/man/cluster-info), `cluster status` (/docs/man/cluster-status), `cluster stop` (/docs/man/cluster-stop), `cluster zkstart` (/docs/man/cluster-zkstart), `cluster zkstop` (/docs/man/cluster-zkstop)

- `db backup` (/docs/man/db-backup), `db copy` (/docs/man/db-copy), `db create` (/docs/man/db-create), `db drop` (/docs/man/db-drop), `db list` (/docs/man/db-list), `db offline` (/docs/man/db-offline), `db online` (/docs/man/db-online), `db optimize` (/docs/man/db-optimize), `db repair` (/docs/man/db-repair), `db restore` (/docs/man/db-restore), `db status` (/docs/man/db-status)
- `function add` (/docs/man/function-add), `function list` (/docs/man/function-list), `function remove` (/docs/man/function-remove)
- `icv add` (/docs/man/icv-add), `icv drop` (/docs/man/icv-drop), `icv remove` (/docs/man/icv-remove)
- `license info` (/docs/man/license-info), `license register` (/docs/man/license-register)
- `log print` (/docs/man/log-print)
- `metadata get` (/docs/man/metadata-get), `metadata set` (/docs/man/metadata-set)
- `query kill` (/docs/man/query-kill), `query list` (/docs/man/query-list), `query status` (/docs/man/query-status)
- `role add` (/docs/man/role-add), `role grant` (/docs/man/role-grant), `role list` (/docs/man/role-list), `role permission` (/docs/man/role-permission), `role remove` (/docs/man/role-remove), `role revoke` (/docs/man/role-revoke)
- `server metrics` (/docs/man/server-metrics), `server profile` (/docs/man/server-profile), `server start` (/docs/man/server-start), `server status` (/docs/man/server-status), `server stop` (/docs/man/server-stop)
- `stored add` (/docs/man/stored-add), `stored list` (/docs/man/stored-list), `stored remove` (/docs/man/stored-remove)
- `user add` (/docs/man/user-add), `user addrole` (/docs/man/user-addrole), `user disable` (/docs/man/user-disable), `user enable` (/docs/man/user-enable), `user grant` (/docs/man/user-grant), `user list` (/docs/man/user-list), `user passwd` (/docs/man/user-passwd), `user permission` (/docs/man/user-permission), `user remove` (/docs/man/user-remove), `user removerole` (/docs/man/user-removerole), `user revoke` (/docs/man/user-revoke)
- `virtual add` (/docs/man/virtual-add), `virtual import` (/docs/man/virtual-import), `virtual list` (/docs/man/virtual-list), `virtual mappings` (/docs/man/virtual-mappings), `virtual options` (/docs/man/virtual-options), `virtual remove` (/docs/man/virtual-remove)
- `zk clear` (/docs/man/zk-clear), `zk info` (/docs/man/zk-info), `zk start` (/docs/man/zk-start), `zk stop` (/docs/man/zk-stop)

Installing Man Pages Locally

To install the man pages locally in your Unix-like environment:

```
$ cp docs/man1/* /usr/local/share/man1
$ cp docs/man8/* /usr/local/share/man8
$ mandb
$ man stardog-admin-server-start
```

ADMINISTERING STARDOG

In this chapter we describe the administration of Stardog Server and Stardog databases, including command-line programs, configuration options, etc.

Security is an important part of Stardog administration; it's discussed separately (Security (#_security)).

Command Line Interface

Stardog's command-line interface (CLI) comes in two parts:

1. `stardog-admin`: administrative client
2. `stardog`: a user's client

The admin and user's tools operate on local or remote databases using HTTP protocol. These CLI tools are Unix-only, are self-documenting, and the help output of these tools is their canonical documentation.[7 (#_footnote_7)]

Help

To use the Stardog CLI tools, you can start by asking them to display help:

```
stardog help
```

Or:

```
$ stardog-admin help
```

These work too:

```
$ stardog
$ stardog-admin
```

Security Considerations

We divide administrative functionality into two CLI programs for reasons of security: `stardog-admin` will need, in production environments, to have considerably tighter access restrictions than `stardog`.

CAUTION For usability, Stardog provides a default user "admin" and password "admin" in `stardog-admin` commands if no user or password are given. This is **insecure**; before any serious use of Stardog is contemplated, read the Security section at least twice, and then—minimally—change the administrative password to something we haven't published on the interwebs!

Command Groups

The CLI tools use "command groups" to make CLI subcommands easier to find. To print help for a particular command group, just ask for help:

```
$ stardog help [command_group_name]
```

The command groups and their subcommands:

- **data:** add, remove, export;
- **query:** search, execute, explain, status;
- **reasoning:** explain, consistency;
- **namespace:** add, list, remove;
- **server:** start, stop;
- **metadata:** get, set;
- **user:** add, drop, edit, grant, list, permission, revoke, passwd;
- **role:** add, drop, grant, list, permission, revoke;
- **db:** backup, copy, create, drop, migrate, optimize, list, online, offline, repair, restore, status;
- **virtual:** add, import, list, mappings, options, remove.

NOTE See the man pages for the canonical list of commands.

The main help command for either CLI tool will print a listing of the command groups:

```
usage: stardog <command> [ <args> ]

The most commonly used stardog commands are:
    data        Commands which can modify or dump the contents of a database
    help        Display help information
    icv         Commands for working with Stardog Integrity Constraint support
    namespace   Commands which work with the namespaces defined for a database
    query       Commands which query a Stardog database
    reasoning   Commands which use the reasoning capabilities of a Stardog database
    version     Prints information about this version of Stardog

See 'stardog help' for more information on a specific command.
```

To get more information about a particular command, simply issue the help command for it including its command group:

```
$ stardog help query execute
```

Finally, everything here about command groups, commands, and online help works for `stardog-admin`, too:

```
$ stardog reasoning consistency -u myUsername -p myPassword -r myDB

$ stardog-admin db migrate -u myUsername -p myPassword myDb
```

Autocomplete

Stardog also supports CLI autocomplete via `bash` autocompletion. To install autocomplete for bash shell, you'll first want to make sure bash completion is installed:

Homebrew

To install:

```
$ brew install bash-completion
```

To enable, edit `.bash\_profile`:

```
if [ -f `brew --prefix`/etc/bash_completion ]; then
  . `brew --prefix`/etc/bash_completion
fi
```

MacPorts

First, you really should be using Homebrew…ya heard?

If not, then:

```
$ sudo port install bash-completion
```

Then, edit `.bash\_profile`:

```
if [ -f /opt/local/etc/bash_completion ]; then
   . /opt/local/etc/bash_completion
fi
```

Ubuntu

And for our Linux friends:

```
$ sudo apt-get install bash-completion
```

Fedora

```
$ sudo yum install bash-completion
```

All Platforms

Now put the Stardog autocomplete script—`stardog-completion.sh`—into your `bash\_completion.d` directory, typically one of `/etc/bash_completion.d, /usr/local/etc/bash_completion.d or ~/bash_completion.d.`

Alternately you can put it anywhere you want, but tell `.bash_profile` about it:

```
source ~/.stardog-completion.sh
```

How to Make a Connection String

You need to know how to make a connection string to talk to a Stardog database. A connection string may consist solely of the **database name** in cases where

1. Stardog is listening on the standard port **5820**; and
2. the command is invoked on the same machine where the server is running.

In other cases, a "fully qualified" connection string, as described below, is required.

Further, the connection string is now assumed to be the first argument of any command that requires a connection string. Some CLI subcommands require a Stardog connection string as an argument to identify the server and database upon which operations are to be performed.

Connection strings are URLs and may either be local to the machine where the CLI is run or they may be on some other remote machine.

Stardog connection strings use the `http://` protocol scheme.

Example Connection Strings

To make a connection string, you need to know the machine name and the port Stardog Server is running on and the name of the database:

```
{scheme}{machineName}:{port}/{databaseName};{connectionOptions}
```

Here are some example connection strings:

```
http://server/billion-triples-punk
http://localhost:5000/myDatabase
http://169.175.100.5:1111/myOtherDatabase;reasoning=true
```

Using the default port for Stardog's use of HTTP protocol simplifies connection strings. `connectionOptions` are a series of `;` delimited key-value pairs which themselves are `=` delimited. Key names must be **lowercase** and their values are case-sensitive.

Server Admin

Stardog Server supports all the administrative functions over the HTTP protocol.

Upgrading Stardog Server

The process of installation is pretty simple; see the Quick Start Guide (#_quick_start_guide) for details.

But how do we easily upgrade between versions? The key is judicious use of `STARDOG_HOME`. Best practice is to keep installation directories for different versions separate and use a `STARDOG_HOME` in another location for storing databases.[8 (#_footnote_8)] Once you set your `STARDOG_HOME` environment variable to point to this directory, you can simply stop the old version and start the new version without copying or moving any files. You can also specify the home directory using the `--home` argument when starting the server.

Server Security

See the Security (#_security) section for information about Stardog's security system, secure deployment patterns, and more.

Configuring Stardog Server

NOTE The properties described in this section control the behavior of the Stardog Server; to set properties or other metadata on individual Stardog **databases**, see Database Admin.

Stardog Server's behavior can be configured via the JVM arg `stardog.home`, which sets Stardog Home, overriding the value of `STARDOG_HOME` set as an environment variable. Stardog Server's behavior can also be configured via a `stardog.properties`—which is a Java Properties file—file in `STARDOG_HOME`. To change the behavior of a running Stardog Server, it is necessary to restart it.

Configuring Temporary ("Scratch") Space

Stardog uses the value of the JVM argument `java.io.tmpdir` to write temporary files for many different operations. If you want to configure temp space to use a particular disk volume or partition, use the `java.io.tmpdir` JVM argument on Stardog startup.

Bad (or, at least, weird) things are guaranteed to happen if this part of the filesystem runs out of (or even low on) free disk space. Stardog will delete temporary files when they're no longer needed. But Stardog admins should configure their monitoring systems to make sure that free disk space is always available, both on `java.io.tmpdir` and on the disk volume that hosts `STARDOG_HOME`.[9 (#_footnote_9)]

Stardog Configuration

The following twiddly knobs for Stardog Server are available in `stardog.properties`:[10 (#_footnote_10)]

1. `query.all.graphs`: Controls what data Stardog Server evaluates queries against; if `true`, it will query over the default graph and the union of all named graphs; if `false` (the default), it will query only over the default graph.
2. `query.pp.contexts`: Controls how property paths interact with named graphs in the data. When set to `true` and the property path pattern is in the default scope (i.e. not inside a `graph` keyword), Stardog will check that paths do not span multiple named graphs (per 18.1.7 (https://www.w3.org/TR/sparql11-query/#sparqlPropertyPaths)). For this to affect query results either there should be multiple `FROM` clauses or `query.all.graphs` must be also set to true.
3. `query.timeout`: Sets the upper bound for query execution time that's inherited by all databases unless explicitly overriden. See Managing Query Performance (#_managing_query_performance) section below for details.
4. `logging.[access,audit].[enabled,type,file]`: Controls whether and how Stardog logs server events; described in detail below.
5. `logging.slow_query.enabled`, `logging.slow_query.time`, `logging.slow_query.type`: The three slow query logging options are used in the following way. To enable logging of slow queries, set `enabled` to `true`. To define what counts as a "slow" query, set `time` to a time duration value (positive integer plus "h", "m", "s", or "ms" for hours, minutes, seconds, or milliseconds respectively). To set the type of logging, set `type` to `text` (the default) or `binary`. A `logging.slow_query.time` that exceeds the value of `query.timeout` will result in empty log entries.**
6. `http.max.request.parameters`: Default is 1024; any value smaller than `Integer.MAX_VALUE` may be provided. Useful if you have lots of named graphs and are at risk of blowing out the value of `http.max.request.parameters`.
7. `database.connection.timeout`: The amount of time a connection to the database can be open, but inactive, before being automatically closed to reclaim the resources. The timeout values specified in the property file should be a positive integer followed by either letter `h` (for hours), letter `m` (for minutes), letter `s` (for seconds), or letters `ms` (for milliseconds). Example intervals: `1h` for 1 hour, `5m` for 5 minutes, `90s` for 90 seconds, `500ms` for 500 milliseconds. Default value is `1h`. **NOTE: setting a short timeout can have adverse results, especially if updates are being performed without commit changes to the server, closing the connection prematurely while using it.**

8. `password.length.min`: Sets the password policy for the minimum length of user passwords, the value can't be lower than `password.length.min` or greater than `password.length.max`. Default: `4`.
9. `password.length.max`: Sets the password policy for the maximum length of user passwords. Default: `1024`.
10. `password.regex`: Sets the password policy of accepted chars in user passwords, via a Java regular expression. Default: `[\w@#$%!&]+`
11. `security.named.graphs`: Sets named graph security on globally. Default: `false`.
12. `spatial.use.jts`: Enabled support for JTS in the geospatial module. Default: `false`

Starting & Stopping the Server

NOTE Unlike the other `stardog-admin` subcommands, starting the server may only be run locally, i.e., on the same machine the Stardog Server is will run on.

The simplest way to start the server—running on the default port, detaching to run as a daemon, and writing `stardog.log` to the current working directory— is

```
$ stardog-admin server start
```

To specify parameters:

```
$ stardog-admin server start --require-ssl --port=8080
```

The port can be specified using the property `--port`.

To shut the server down:

```
$ stardog-admin server stop
```

If you started Stardog on a port other than the default, or want to shut down a remote server, you can simply use the `--server` option to specify the location of the server to shutdown.

By default Stardog will bind it's server to `0.0.0.0`. You can specify a different network interface for Stardog to bind to using the `--bind` property of `server start`.

Server Monitoring

Stardog provides server monitoring via the Metrics library (http://metrics.dropwizard.io/). In addition to providing some basic JVM information, Stardog also exports information about the Stardog DBMS configuration as well as stats for all databases within the system, such as the total number of open connections, size, and average query time.

Accessing Monitoring Information

Monitoring information is available via the Java API, the HTTP API, the CLI or (if configured) the JMX interface. Performing a `GET` on `/admin/status` which will return a JSON object containing the information available the server and all the databases. The endpoint `DB/status` will return the monitoring information about the database status. The `stardog-admin server status` command will print a subset of this information on the console.

Configuring JMX Monitoring

By default, JMX monitoring is not enabled. You can enable it by setting `metrics.reporter=jmx` in the `stardog.properties` file. Then, you can simply use a tool like VisualVM or JConsole to attach to the process running the JVM, or connect directly to the JMX server.

If you want to connect to the JMX server remotely you need to set `metrics.jmx.remote.access=true` in `stardog.properties`. Stardog will bind an RMI server for remote access on port `5833`. If you want to change this port Stardog binds the remote server to, you can set the property `metrics.jmx.port` in `stardog.properties`.

Finally, if you wish to disable monitoring completely, set `metrics.enabled` to `false` in `stardog.properties`.

Locking Stardog Home

Stardog Server will lock `STARDOG_HOME` when it starts to prevent synchronization errors and other nasties if you start more than one Stardog Server with the same `STARDOG_HOME`. If you need to run more than one Stardog Server instance, choose a different `STARDOG_HOME` or pass a different value to `--home`.

Access & Audit Logging

See the exemplar `stardog.properties` (https://github.com/Complexible/stardog-examples/blob/master/config/stardog.properties) file for a complete discussion of how access and audit logging work in Stardog Server. Audit logging is a superset of the events in access logging. Access logging covers the most often required logging events; you should consider enabling audit logging if you really need to log **every** server event. Logging generally doesn't have much impact on performance; but the safest way to insure that impact is negligible is to log to a separate disk (or to a centralized logging server, etc.).

The important configuration choices are whether logs should be binary or plain text (both based on ProtocolBuffer message formats); the type of logging (audit or access); the logging location (which may be "off disk" or even "off machine") Logging to a centralized logging facility requires a Java plugin that implements the Stardog Server logging interface; see Java Programming (#_java_programming) for more information; and the log rotation policy (file size or time).

Slow query logging is also available. See the Managing Running Queries (#_managing_running_queries) section below.

Database Admin

Stardog is a multi-tenancy system and will happily give access to many, physically distinct databases.

Configuring a Database

To administer a Stardog database, some config options must be set at creation time; others may be changed subsequently and some may never be changed. All config options have sensible defaults (except for the database name), so you don't have to twiddle any of the knobs till you really need to.

To configure a database, use the `metadata-get` and `metadata-set` CLI commands. See Man Pages (#_man_pages) for the details.

Configuration Options

1. Table of Configuration Options

Option	Mutable	Default	API
`database.archetypes`	Yes		DatabaseOptions.ARCHETYPES (/docs/java/snarl/com/complexible/stardog/db/databaseoptions#ARCHETYPES)
The name of one or more database archetypes.			
`database.connection.timeout`	Yes	`1h`	DatabaseOptions.CONNECTION_TIMEOUT (/docs/java/snarl/com/complexible/stardog/db/databaseoptions#CONNECTION_TIME
Same as `database.connection.timeout` described in Stardog Configuration (#_stardog_configuration) but applies only to one database.			
`database.name`	No		DatabaseOptions.NAME (/docs/java/snarl/com/complexible/stardog/db/databaseoptio
A database name, the legal value of which is given by the regular expression `[A-Za-z]{1}[A-Za-z0-9_-]`.			
`database.namespaces`	Yes	`rdf, rdfs, xsd, owl, stardog`	DatabaseOptions.NAMESPACES (/docs/java/snarl/com/complexible/stardog/db/databaseoptions#NAMESPACES)
Sets the default namespaces for new databases.			
`database.online`	No	`true`	DatabaseOptions.ONLINE (/docs/java/snarl/com/complexible/stardog/db/databaseopt

Option	Mutable	Default	API
The status of the database: online or offline. It may be set so that the database is created initially in online or offline status; subsequently, it can’t be set directly but o relevant admin commands.			
`docs.default.rdf.extractors`	Yes	`tika`	BitesOptions.DOCS_DEFAULT_RDF_EXTRACTORS (/docs/java/snarl/com/complexible/stardog/docs/bitesoptions#DOCS_DEFAULT_RDF_E
Comma-separated list of names of RDF extractors to use when processing documents when no RDF extractor names are given.			
`docs.default.text.extractors`	Yes	`tika`	BitesOptions.DOCS_DEFAULT_TEXT_EXTRACTORS (/docs/java/snarl/com/complexible/stardog/docs/bitesoptions#DOCS_DEFAULT_TEXT_
Comma-separated list of names of text extractors to use when processing documents when no text extractor names are given.			
`docs.filesystem.uri`	Yes	`file:/// (file:///)`	BitesOptions.DOCS_FILESYSTEM_URI (/docs/java/snarl/com/complexible/stardog/docs/bitesoptions#DOCS_FILESYSTEM_UR
A URI indicating which `FileSystem` provider to use for document storage. In addition to local storage (`file:/// (file:///)`), documents can be stored on Amazo document storage can be disabled altogether (`none`).			
`docs.opennlp.models.path`	Yes		BitesOptions.DOCS_OPENNLP_MODELS_PATH (/docs/java/snarl/com/complexible/stardog/docs/bitesoptions#DOCS_OPENNLP_MOD
The directory where OpenNLP models are located. See Entity Extraction and Linking (#_entity_extraction_and_linking) for details.			
`docs.path`	Yes	`docs/`	BitesOptions.DOCS_PATH (/docs/java/snarl/com/complexible/stardog/docs/bitesoptio
The path under which documents will be stored. A relative path is relative to the database directory. S3 storage should specify an absolute path with the bucket name the path.			
`docs.s3.protocol`	Yes	`https`	BitesOptions.DOCS_S3_PROTOCOL (/docs/java/snarl/com/complexible/stardog/docs/bitesoptions#DOCS_S3_PROTOCOL)
Protocol used when storing on S3 (and compatible) stores. Can be set to `http` to disable TLS/SSL.			
`icv.active.graphs`	No	`default`	ICVOptions.ICV_ACTIVE_GRAPHS (/docs/java/snarl/com/complexible/stardog/icv/icvoptions#ICV_ACTIVE_GRAPHS)
Specifies which part of the database, in terms of named graphs, is checked with IC validation. Set to `tag:stardog:api:context:all` to validate all the named grap otherwise, the legal value of `icv.active.graphs` is a comma-separated list of named graph identifiers.			
`icv.consistency.automatic`	Yes	`false`	ICVOptions.ICV_CONSISTENCY_AUTOMATIC (/docs/java/snarl/com/complexible/stardog/icv/icvoptions#ICV_CONSISTENCY_AUTOM
Enables automatic ICV consistency check as part of transactions.			
`icv.enabled`	Yes	`false`	ICVOptions.ICV_ENABLED (/docs/java/snarl/com/complexible/stardog/icv/icvoptions#I
Determines whether ICV is active for the database; if true, all database mutations are subject to IC validation (i.e., "guard mode").			
`icv.reasoning.enabled`	Yes	`false`	ICVOptions.ICV_REASONING_ENABLED (/docs/java/snarl/com/complexible/stardog/icv/icvoptions#ICV_REASONING_ENABLED
Determines if reasoning is used during IC validation.			
`index.differential.enable.limit`	Yes	`500,000`	IndexOptions.DIFF_INDEX_MIN_LIMIT (/docs/java/snarl/com/complexible/stardog/index/indexoptions#DIFF_INDEX_MIN_LIM
Sets the minimum size of the Stardog database before differential indexes are used. The legal value is an integer.			
`index.differential.merge.limit`	Yes	`20,000`	IndexOptions.DIFF_INDEX_MAX_LIMIT (/docs/java/snarl/com/complexible/stardog/index/indexoptions#DIFF_INDEX_MAX_LIM
Sets the size in number of RDF triples before the differential indexes are merged to the main indexes. The legal value is an integer.			

Option	Mutable	Default	API
`index.literals.canonical`	No	`true`	IndexOptions.CANONICAL_LITERALS (/docs/java/snarl/com/complexible/stardog/index/indexoptions#CANONICAL_LITERAL
Enables RDF literal canonicalization.			
`index.named.graphs`	No	`true`	IndexOptions.INDEX_NAMED_GRAPHS (/docs/java/snarl/com/complexible/stardog/index/indexoptions#INDEX_NAMED_GRAP
Enables optimized index support for named graphs; speeds SPARQL query evaluation with named graphs at the cost of some overhead for database loading and inde			
`index.statistics.update.automatic`	Yes	`true`	IndexOptions.AUTO_STATS_UPDATE (/docs/java/snarl/com/complexible/stardog/index/indexoptions#AUTO_STATS_UPDAT
Determines whether statistics are maintained automatically.			
`index.statistics.update.min.size`	Yes	10000	IndexOptions.STATS_UPDATE_DB_MIN_SIZE (/docs/java/snarl/com/complexible/stardog/index/indexoptions#STATS_UPDATE_DB_
Minimum number of triples that should be in the database for statistics to be updated automatically.			
`index.statistics.update.ratio`	Yes	0.1	IndexOptions.STATS_UPDATE_RATIO (/docs/java/snarl/com/complexible/stardog/index/indexoptions#STATS_UPDATE_RATI
Ratio of updated triples to the number of triples in the database that triggers the automatic statistics computation in a background thread. This option has no effect i `index.statistics.update.automatic` is off or the index size is less than `index.statistics.update.min.size` .			
`index.statistics.update.blocking.ratio`	Yes	0.0	IndexOptions.STATS_UPDATE_BLOCKING_RAITO (/docs/java/snarl/com/complexible/stardog/index/indexoptions#STATS_UPDATE_BLO
Similar to `index.statistics.update.ratio` but once the updates go over this limit statistics computation will be performed synchronously within the transaction background thread. Setting this option to a non-positive number ({@code ⇐ 0}) will disable blocking updates.			
`preserve.bnode.ids`	No	`true`	DatabaseOptions.PRESERVE_BNODE_IDS (/docs/java/snarl/com/complexible/stardog/db/databaseoptions#PRESERVE_BNODE_
Determines how the Stardog parser handles bnode identifiers that may be present in RDF input. If this property is enabled (i.e., `TRUE`), parsing and data loading perf improved; but the other effect is that if distinct input files use (randomly or intentionally) the same bnode identifier, that bnode will point to one and the same node i have input files that use explicit bnode identifiers, and more than one of those files may use the same bnode identifiers, and you don't want those bnodes to be smu node in the database, then this configuration option should be disabled (set to `FALSE`).			
`query.all.graphs`	Yes	`false`	DatabaseOptions.QUERY_ALL_GRAPHS (/docs/java/snarl/com/complexible/stardog/db/databaseoptions#QUERY_ALL_GRAPHS
Determines what data the database evaluates queries against; if `true` , it will query over the default graph and the union of all named graphs; if `false` (the default), the default graph. This database option overrides any global server settings.			
`query.describe.strategy`	Yes	default	DatabaseOptions.QUERY_DESCRIBE_STRATEGY (/docs/java/snarl/com/complexible/stardog/db/databaseoptions#QUERY_DESCRIBE_S
Option to set the default DESCRIBE query strategy for the database			
`query.pp.contexts`	Yes	`false`	DatabaseOptions.PROPERTY_PATH_CONTEXTS (/docs/java/snarl/com/complexible/stardog/db/databaseoptions#PROPERTY_PATH_CO
Determines how property paths interact with named graphs in the data. When set to `true` and the property path pattern is in the default scope (i.e. not inside a `gra` will check that paths do not span multiple named graphs (per 18.1.7 (https://www.w3.org/TR/sparql11-query/#sparqlPropertyPaths)). For this to affect query results multiple `FROM` clauses or `query.all.graphs` must be also set to true. This database option overrides any global server settings.			
`query.plan.reuse`	Yes	Always	DatabaseOptions.QUERY_PLAN_REUSE (/docs/java/snarl/com/complexible/stardog/db/databaseoptions#QUERY_PLAN_REUS

Option	Mutable	Default	API
Option for configuring how Stardog will reuse query plans. Stardog answers queries by first generating an execution plan. Generating an optimal query plan is hard a these plans are cached and reused for structurally equivalent queries; i.e. queries such that one can be transformed into another by replacing constants. This option conditions under which a cached plan will be reused. See QueryPlanReuse (/docs/java/snarl/com/complexible/stardog/queryplanreuse) for the available values.			
`query.timeout`	Yes		DatabaseOptions.QUERY_TIMEOUT (/docs/java/snarl/com/complexible/stardog/db/databaseoptions#QUERY_TIMEOUT)
Determines max execution time for query evaluation.			
`reasoning.consistency.automatic`	Yes	`false`	ReasoningOptions.CONSISTENCY_AUTOMATIC (/docs/java/snarl/com/complexible/stardog/reasoning/reasoningoptions#CONSISTENC
Enables automatic consistency checking with respect to a transaction.			
`reasoning.punning.enabled`	Yes	`false`	ReasoningOptions.PUNNING_ENABLED (/docs/java/snarl/com/complexible/stardog/reasoning/reasoningoptions#PUNNING_E
Enables punning.			
`reasoning.schema.graphs`	Yes	`*`	ReasoningOptions.SCHEMA_GRAPHS (/docs/java/snarl/com/complexible/stardog/reasoning/reasoningoptions#SCHEMA_GR
Determines which, if any, named graph or graphs contains the "TBox", i.e., the schema part of the data. The legal value is a comma-separated list of named graph ide (optionally) the special names, `tag:stardog:api:context:default` and `tag:stardog:api:context:all` , which represent the default graph and the union of al the default graph, respectively. In the context of database configurations only, Stardog will recognize `default` and `*` as short forms of those URIs, respectively.			
`reasoning.type`	Yes	`SL`	
Specifies the reasoning type associated with the database; legal values are `SL` , `RL` , `QL` , `EL` , `DL` , `RDFS` , and `NONE` .			
`reasoning.approximate`	Yes	`false`	ReasoningOptions.APPROXIMATE (/docs/java/snarl/com/complexible/stardog/reasoning/reasoningoptions#SCHEMA_GR
Enables approximate reasoning. With this flag enabled Stardog will approximate an axiom that is outside the profile Stardog supports and normally ignored. For exa class axiom might be split into two subclass axioms and only one subclass axiom is used.			
`reasoning.sameas`	Yes	OFF	ReasoningOptions.EQUALITY_REASONING (/docs/java/snarl/com/complexible/stardog/reasoning/reasoningoptions#EQUALITY_R
Option to enable `owl:sameAs` reasoning. When this option is set to `ON` reflexive, symmetric, and transitive closure of the `owl:sameAs` triples in the database is com to `FULL` , `owl:sameAs` inferences are computed based on the schema axioms such as functional properties.			
`search.enabled`	Yes	`false`	SearchOptions.SEARCHABLE (/docs/java/snarl/com/complexible/stardog/search/searchoptions#SEARCHABLE)
Enables semantic search for the database.			
`search.wildcard.search.enabled`	Yes	`false`	SearchOptions.LEADING_WILDCARD_SEARCH_ENABLED (/docs/java/snarl/com/complexible/stardog/search/searchoptions#LEADING_WILDCAR
Enable support in Lucene for searches with leading wildcards.			
`search.default.limit`	Yes	`-1`	SearchOptions.SEARCH_DEFAULT_LIMIT (/docs/java/snarl/com/complexible/stardog/search/searchoptions#SEARCH_DEFAULT_
Specify the default limit on the number of results returned from a full-text search (`-1` returns all results)			
`search.reindex.tx`	Yes	`true`	SearchOptions.SEARCH_REINDEX_IN_TX (/docs/java/snarl/com/complexible/stardog/search/searchoptions#SEARCH_REINDEX_
If `false` , literals added during a transaction are not automatically indexed; users need to `optimize` the database in order to make them available for search.			
`spatial.enabled`	Yes	`false`	GeospatialOptions.SPATIAL_ENABLED (/docs/java/snarl/com/complexible/stardog/spatial/geospatialoptions#SPATIAL_ENABI

Option	Mutable	Default	API
Enables geospatial search for the database.			
`spatial.result.limit`	Yes	10000	GeospatialOptions.SPATIAL_RESULT_LIMIT (/docs/java/snarl/com/complexible/stardog/spatial/geospatialoptions#SPATIAL_RESUI
Specify the default limit on the number of results returned from a geospatial query (`-1` returns all results)			
`spatial.precision`	No	`11`	GeospatialOptions.SPATIAL_PRECISION (/docs/java/snarl/com/complexible/stardog/spatial/geospatialoptions#SPATIAL_PRECI
Specifies the precision used for the indexing of geospatial data. The smaller the value, the less precision.			
`strict.parsing`	No	`true`	DatabaseOptions.STRICT_PARSING (/docs/java/snarl/com/complexible/stardog/db/databaseoptions#STRICT_PARSING)
Controls whether Stardog parses RDF strictly (`true` , the default) or loosely (`false`)			
`transaction.isolation`	Yes	`SNAPSHOT`	DatabaseOptions.TRANSACTION_ISOLATION (/docs/java/snarl/com/complexible/stardog/db/databaseoptions#TRANSACTION_ISOL
Configures isolation level for transactions; legal values are SNAPSHOT and SERIALIZABLE.			
`transaction.logging`	Yes	`false`	DatabaseOptions.TRANSACTION_LOGGING (/docs/java/snarl/com/complexible/stardog/db/databaseoptions#TRANSACTION_LOG
Enables logged transactions. Logged transactions are activated by default in Cluster mode.			
`transaction.logging.rotation.size`	Yes	`524288000`	DatabaseOptions.TRANSACTION_LOGGING_ROTATION_SIZE (/docs/java/snarl/com/complexible/stardog/db/databaseoptions#TRANSACTION_LOG
When `transaction.logging` is `true` , it determines the size (in bytes) at which the transaction log will be rotated. Default is 500 MB.			
`transaction.logging.rotation.remove`	Yes	`true`	DatabaseOptions.TRANSACTION_LOGGING_ROTATION_REMOVE (/docs/java/snarl/com/complexible/stardog/db/databaseoptions#TRANSACTION_LOG
When `transaction.logging` is `true` , it determines that old log files will be deleted after rotation. Default is true.			

A Note About Database Status

A database must be set to `offline` status before most configuration parameters may be changed. Hence, the normal routine is to set the database offline, change the parameters, and then set the database to online. All of these operations may be done programmatically from CLI tools, such that they can be scripted in advance to minimize downtime. In a future version, we will allow some properties to be set while the database remains online.

Managing Database Status

Databases are either online or offline; this allows database maintenance to be decoupled from server maintenance.

Online and Offline

Databases are put online or offline synchronously: these operations block until other database activity is completed or terminated. See `stardog-admin help db` for details.

Examples

To set a database from offline to online:

```
$ stardog-admin db offline myDatabase
```

To set the database online:

```
$ stardog-admin db online myDatabase
```

If Stardog Server is shutdown while a database is offline, the database will be offline when the server restarts.

Creating a Database

Stardog databases may be created locally or remotely; but performance is better if data files don't have to be transferred over a network during creation and initial loading. See the section below about loading compressed data. All data files, indexes, and server metadata for the new database will be stored in Stardog Home. Stardog won't create a database with the same name as an existing database. Stardog database names must conform to the regular expression, `[A-Za-z]{1}[A-Za-z0-9_-]`.

NOTE There are four reserved words that may not be used for the names of Stardog databases: `system`, `admin`, and `docs`.

Minimally, the only thing you must know to create a Stardog database is a database **name**; alternately, you may customize some other database parameters and options depending on anticipated workloads, data modeling, and other factors.

See `stardog-admin help db create` for all the details including examples.

Database Archetypes

Stardog database archetypes are a new feature in 2.0. A database archetype is a named, vendor-defined or user-defined bundle of data and functionality to be applied at database-creation time. Archetypes are primarily for supporting data standards or tool chain configurations in a simple way.

For example, the SKOS standard from W3C defines an OWL vocabulary for building taxonomies, thesauruses, etc. SKOS is made up by a vocabulary, some constraints, some kinds of reasoning, and (typically) some SPARQL queries. If you are developing an app that uses SKOS, without Stardog's SKOS archetype, you are responsible for assembling all that SKOS stuff yourself. Which is tedious, error-prone, and unrewarding—even when it's done right the first time.

Rather than putting that burden on Stardog users, we've created database archetypes as a mechanism to collect these "bundles of stuff" which, as a developer, you can then simply attach to a particular database.

The last point to make is that archetypes are composable: you can mix-and-match them at database creation time as needed.

Stardog supports two database archetypes out-of-the-box: PROV (http://www.w3.org/TR/prov-overview/) and SKOS (http://www.w3.org/2004/02/skos/).

SKOS Archetype

The SKOS archetype is for databases that will contain SKOS data, and includes the SKOS schema, SKOS constraints using Stardog's Integrity Constraint Validation, and some namespace-prefix bindings.

PROV Archetype

The PROV archetype is for databases that will contain PROV data, and includes the SKOS schema, SKOS constraints using Stardog's Integrity Constraint Validation, and some namespace-prefix bindings.

Archetypes are composable, so you can use more of them and they are intended to be used alongside **your domain data**, which may include as many other schemas, ontologies, etc. as are required.

User-defined Archetypes

Please see the Stardog Examples repository (https://github.com/Complexible/stardog-examples/blob/master/examples/foaf/readme.md) on Github for an example that shows how to create your own Stardog archetype.

Database Creation Templates

As a boon to the overworked admin or devops peeps, Stardog Server supports database creation templates: you can pass a Java Properties file with config values set and with the values (typically just the database name) that are unique to a specific database passed in CLI parameters.

Examples

To create a new database with the default options by simply providing a name and a set of initial datasets to load:

```
$ stardog-admin db create -n myDb input.ttl another_file.rdf moredata.rdf.gz
```

Datasets can be loaded later as well. To create (in this case, an empty) database from a template file:

```
$ stardog-admin db create -c database.properties
```

At a minimum, the configuration file must have a value for `database.name` option.

If you only want to change only a few configuration options you can directly give the values for these options in the CLI args as follows:

```
$ stardog-admin db create -n db -o icv.enabled=true icv.reasoning.enabled=true -- input.ttl
```

"--" is used in this case when "-o" is the last option to delimit the value for "-o" from the files to be bulk loaded.

Please refer to the CLI help for more details of the `db create` command.

Database Create Options

2. Table of Options for Stardog's `create` command

Name	Description	Arg values	Default
`--name`, `-n`	Required, the name of the database to create		
`--copy-server-side`,	Flag to specify whether bulk loaded files should be first copied to the server		`false`
`--type`, `-t`	Specifies the kind of database indexes: memory or disk	`M`, `D`	disk
`--index-triples-only`, `-i`	Specifies that the database's indexes should be optimized for RDF triples only		`false`

Repairing a Database

If an I/O error or an index exception occurs while querying a DB, the DB may be corrupted and repaired with the repair command. If the errors occur during executing admin commands, then the system DB may have been corrupted. System database corruptions can also cause other problems including authorization errors.

This command needs exclusive access to your Stardog home directory and therefore requires the Stardog Server not to be running. This also means that the command can only be run on the machine where the Stardog home directory is located, and you will not be able to start the Stardog Server while this command is running.

NOTE The repair process can take considerable time for large databases.

If the built-in Stardog system database is corrupted, then you can use the database name `system` as the repair argument. To repair the database myDB:

```
$ stardog-admin db repair myDB
```

To repair the system database:

```
$ stardog-admin db repair system
```

Backing Up and Restoring

Stardog includes both physical and logical backup utilities; logical backups are accomplished using the `export` CLI command. Physical backups and restores are accomplished using `stardog-admin db backup` and `stardog-admin db restore` commands, respectively.

These tools perform physical backups, including database metadata, rather than logical backups via some RDF serialization. They are **native** Stardog backups and can only be restored with Stardog tools. Backup may be accomplished while a database is online; backup is performed in a read transaction: reads and writes may continue, but writes performed during the backup are not reflected in the backup.

See the man pages for `backup` (/man/db-backup.html) and `restore` (/man/db-restore.html) for details.

Backup

`stardog-admin db backup` assumes a default location for its output, namely, `$STARDOG_HOME/.backup`; that default may be overriden by passing a `-t` or `--to` argument. Backup sets are stored in the backup directory by database name and then in date-versioned subdirectories for each backup volume. You can use a variety of OS-specific options to do remote backups over some network or data protocol; those options are left as an exercise for the admin.

To backup a Stardog database called `foobar`:

```
$ stardog-admin db backup foobar
```

To perform a remote backup, for example, pass in a specific directory that may be mounted in the current OS namespace via some network protocol, thus:

```
$ stardog-admin db backup --to /my/network/share/stardog-backups foobar
```

Backups can also be performed directly to S3. To do so use an S3 URL in the following format:

```
s3://[<endpoint hostname>:<endpoint port>]/<bucket name>/<path prefix>?region=<AWS
Region>&AWS_ACCESS_KEY_ID=<access key>&AWS_SECRET_ACCESS_KEY=<verySecretKey1>
```

The `endpoint hostname` and `endpoint port` values are only used for on-premises S3 clones. To use Amazon S3 those values can be left blank and the URL will have three `/` before the bucket, eg `s3:///mybucket/backup/prefix?region=us-east-1&AWS_ACCESS_KEY_ID=accessKey&AWS_SECRET_ACCESS_KEY=secret`

A default S3 backup location can also be specified in the `stardog.properties` file with the key `backup.location`.

Restore

To restore a Stardog database from a Stardog backup volume, simply pass a fully-qualified path to the volume in question. The location of the backup should be the full path to the backup, not the location of the backup directory as specified in your Stardog configuration. There is no need to specify the name of the database to restore.

To restore a database from its backup:

```
$ stardog-admin db restore $STARDOG_HOME/.backups/myDb/2012-06-21
```

Backups can also be restored directly from S3 by using an S3 URL in the following format:

```
s3://[<endpoint hostname>:<endpoint port>]/<bucket name>/<path prefix>/<database name>?region=<A
WS Region>&AWS_ACCESS_KEY_ID=<access key>&AWS_SECRET_ACCESS_KEY=<verySecretKey1>
```

Note: Unlike the backup URL the database name must be specified as the last entry of the `path` field in the URL.

Restore On Startup

Stardog can be configured to automatically restore databases from a backup location on startup. For example, when a Stardog cluster node first starts it could pull all of the database data down from an S3 backup before joining the cluster.

There are two options that control this behavior.

3. Table of Auto-restore options

Option	Description
`backup.autorestore.dbnames`	A regular expression that matches the names of the databases to automatically restore on startup, eg: `.*` for every database.
`backup.autorestore.onfailure`	A boolean value that determines if all databases which failed to load should be automatically restored from a backup location.

One-time Database Migrations for Backup

The backup system cannot directly backup databases created in versions before 2.1. These databases must be explicitly migrated to use the new backup system; this is a one-time operation per database and is accomplished by running

```
$ stardog-admin db migrate foobar
```

to migrate a database called `foobar`. Again, this is a one-time operation only and all databases created with 2.1 (or later) do not require it.

Namespace Prefix Bindings

Stardog allows database administrators to persist and manage custom namespace prefix bindings:

1. At database creation time, if data is loaded to the database that has namespace prefixes, then those are persisted for the life of the database. This includes setting the default namespace to the default that appears in the file. Any subsequent queries to the database may simply omit the `PREFIX` declarations:

   ```
   $ stardog query myDB "select * {?s rdf:type owl:Class}"
   ```

2. To add new bindings, use the `namespace` subcommand in the CLI:

   ```
   $ stardog namespace add myDb --prefix ex --uri 'http://example.org/test#'
   ```

3. To change the default binding, use a quote prefix when adding a new one:

   ```
   $ stardog namespace add myDb --prefix "" --uri http://new.default
   ```

4. To change an existing binding, delete the existing one and then add a new one:

   ```
   $ stardog namespace remove myDb --prefix ex
   ```

5. Finally, to see all the existing namespace prefix bindings:

   ```
   $ stardog namespace list myDB
   ```

If no files are used during database creation, or if the files do not define any prefixes (e.g. NTriples), then the "Big Four" default prefixes are stored: `RDF, RDFS, XSD`, and `OWL`.

When executing queries in the CLI, the default table format for SPARQL `SELECT` results will use the bindings as qnames. SPARQL `CONSTRUCT` query output (including export) will also use the stored prefixes. To reiterate, namespace prefix bindings are **per database**, not global.

Index Strategies

By default Stardog builds extra indexes for named graphs. These indexes are used when SPARQL queries specify datasets using `FROM` and `FROM NAMED`.

Stardog may also be configured to create and to use fewer indexes, if the database is only going to be used to store RDF triples—that is to say, if the database will not be used to store named graph information. In this mode, Stardog will keep fewer indexes, which will result in faster database creation and faster updates without compromising query answering performance. In such databases, quads (that is: triples with named graphs or contexts specified) may still be added to these database at any time, but query performance may degrade in such cases.

To create a database which indexes only RDF triples, set the option `index.named.graphs` to `false` at database creation time. The CLI provides a shorthand option, `-i` or `--index-triples-only`, which is equivalent.

NOTE This option can only be set at database creation time and cannot be changed later without rebuilding the database; use this option with care.

Differential Indexes

While Stardog is generally biased in favor of read performance, write performance is also important in many applications. To increase write performance, Stardog may be used, optionally, with a **differential index**.

Stardog's differential index is used to persist additions and removals separately from the main indexes, such that updates to the database can be performed faster. Query answering takes into consideration all the data stored in the main indexes and the differential index; hence, query answers are computed as if all the data is stored in the main indexes.

There is a slight overhead for query answering with differential indexes if the differential index size gets too large. For this reason, the differential index is merged into the main indexes when its size reaches `index.differential.merge.limit`. There is also no benefit of differential indexes if the main index itself is small. For this reason, the differential index is not used until the main index size reaches `index.differential.enable.limit`.

Usually the default values of the differential index parameters will work fine and don't need to be changed. The corollary is that you shouldn't change this value in a production system till you've tested the effects of a change in a non-production system.

NOTE Increasing `index.differential.merge.limit` will improve the throughput for updates, but there will be longer pauses when the differential index is being merged. Reducing the value of `index.differential.merge.limit` will make sure there are no long pauses but overall throughput will suffer. The optimal value depends on the average size of a transaction for the application and the frequency of updates.

Loading Compressed Data

Stardog supports loading data from compressed files directly: there's no need to uncompress files before loading. Loading compressed data is the recommended way to load large input files. Stardog supports GZIP, BZIP2 and ZIP compressions natively.

GZIP and BZIP2

A file passed to `create` will be treated as compressed if the file name ends with `.gz` or `.bz2`. The RDF format of the file is determined by the penultimate extension. For example, if a file named `test.ttl.gz` is used as input, Stardog will perform GZIP decompression during loading and parse the file with Turtle parser. All the formats supported by Stardog (RDF/XML, Turtle, Trig, etc.) can be used with compression.

ZIP

The ZIP support works differently since zipped files can contain many files. When an input file name ends with `.zip`, Stardog performs ZIP decompression and tries to load all the files inside the ZIP file. The RDF format of the files inside the zip is determined by their file names as usual. If there is an unrecognized file extension (e.g. '.txt'), then that file will be skipped.

Dropping a Database

This command removes a database and all associated files and metadata. This means all files on disk related to the database will be deleted, so only use `drop` when you're certain! Databases must be offline in order to be dropped.

It takes as its only argument a valid database name. For example,

```
$ stardog-admin db drop my_db
```

Using Integrity Constraint Validation

Stardog supports integrity constraint validation as a data quality mechanism via closed world reasoning. Constraints can be specified in OWL, SWRL, and SPARQL. Please see the Validating Constraints (#_validating_constraints) section for more about using ICV in Stardog.

The CLI `icv` subcommand can be used to add, delete, or drop all constraints from an existing database. It may also be used to validate an existing database with constraints that are passed into the `icv` subcommand; that is, using different constraints than the ones already associated with the database.

For details of ICV usage, see `stardog help icv` and `stardog-admin help icv`. For ICV in transacted mutations of Stardog databases, see the database creation section above.

Migrating a Database

The `migrate` subcommand migrates an older Stardog database to the latest version of Stardog. Its only argument is the name of the database to migrate. `migrate` won't necessarily work between arbitrary Stardog version, so before upgrading check the release notes for a new version carefully to see whether migration is required or possible.

```
$ stardog-admin db migrate myDatabase
```

will update `myDatabase` to the latest database format.

Getting Database Information

You can get some information about a database by running the following command:

```
$ stardog-admin metadata get my_db_name
```

This will return all the metadata stored about the database, including the values of configuration options used for this database instance. If you want to get the value for a specific option then you can run the following command:

```
$ stardog-admin metadata get -o index.named.graphs my_db_name
```

Managing Stored Functions

Stored functions, available since Stardog 5.1, provide the ability to reuse expressions. This avoids duplication and ensures consistency across instances of the same logic. Stored functions are treated similarly to built-in and user-defined functions in that they can be used in `FILTER` constraints and `BIND` assignments in SPARQL queries, path queries and rules.

Creating and Using Functions

Functions are useful to encapsulate computational or business logic for reuse. We can create a new function to compute the permutation using the `function add` command to `stardog-admin` on the command line:

```
stardog-admin function add "function permutation(?n, ?r) { factorial(?n) / factorial(?n - ?r) }"
```

We can use this function in a SPARQL query and see that the function is expanded in the query plan:

```
Explaining Query:

select * where { ?x :p :q. filter(permutation(?x, 3) > 1) }

The Query Plan:

Projection(?x) [#1]
`─ Filter((factorial(?x) / factorial((?x - "3"^^xsd:integer))) > "1"^^xsd:integer) [#1]
   `─ Scan[POS](?x, :p, :q) [#1]
```

Stored Function Syntax

Function definitions provided to the `add` command must adhere to the following grammar:

```
FUNCTIONS ::= Prolog FUNCTION+

FUNCTION ::= 'function' FUNC_NAME '(' ARGS ')' '{' Expression '}'

FUNC_NAME ::= IRI | PNAME | LOCAL_NAME

ARGS ::= [Var [',' Var]* ]?

Prolog ::= // BASE and PREFIX declarations as defined by SPARQL 1.1
Expression ::= // as defined by SPARQL 1.1
Var ::= // as defined by SPARQL 1.1
```

We can use IRIs or prefixed names as function names and include several functions in one `add` call:

```
$ stardog-admin function add "prefix ex: <http://example/> \
     function ex:permutation(?n, ?r) { factorial(?n) / factorial(?n - ?r) } \
     function <http://example/combination>(?n, ?r) { permutation(?n, ?r) / factorial(?r) }"

Stored 2 functions successfully
```

Additional Function Management

The admin commands cover adding, listing and removing functions. Examples of these commands are shown below:

```
$ stardog-admin function list
FUNCTION combination(?n,?r) {
   ((factorial(?n) / factorial((?n - ?r))) / factorial(?r))
}

FUNCTION permutation(?n,?r) {
   (factorial(?n) / factorial((?n - ?r)))
}
```

```
$ stardog-admin function remove permutation
Removed stored function successfully
```

HTTP APIs are also provided to add, list and remove stored functions:

- `GET /admin/functions/stored[/?name={functionName}]`
- `DELETE /admin/functions/stored[/?name={functionName}]`
- `POST /admin/functions/stored`

The contents of the `POST` request should be a document containing one or more function definitions using the syntax describes above. The `GET` request by default returns the definitions for all the functions. If the `name` parameter is specified a definition for the function with that name is returned. Similarly, the `DELETE` request deletes all the functions by default or deletes a single function if the `name` parameter is specified.

Stored functions are persisted in the system database. The system database should be backed up properly to avoid loss of functions.

Dependencies Across Stored Functions

Stored functions are compiled at creation time in a way that guarantees they will work indefinitely, even if other functions are removed or changed in ways that would affect them. For this reason, dependent functions need to be reloaded when their dependencies are changed.

Managing Stored Queries

Stardog 4.2 added the capability to name and store SPARQL queries for future evaluation by referring to the query's name.

Queries of any type can be stored in Stardog and executed directly by using the name of the stored query. Stored queries can be shared with other users, which gives those users the ability to run those queries provided that they have appropriate permissions for a database.

Stored queries can be managed via CLI, Java API, and HTTP API. The CLI command group is `stardog-admin stored`. The HTTP API will be detailed below.

Storing Queries

Queries can be stored using the `stored add` admin command and specifying a unique name for the stored query:

```
$ stardog-admin stored add -n types "select distinct ?type {?s a ?type}"
```

If a file is used to specify the query string without an explicit `-n/--name` option then the name of the query file is used for the stored query:

```
$ stardog-admin stored add listProperties.sparql
```

Queries can also be stored via HTTP:

```
POST /admin/queries/stored → application/json
```

Input JSON example:

```
{
  "name": "types",
  "creator": "admin",
  "database": "*",
  "query": "select distinct ?type {?s a ?type}"
}
```

By default, stored queries can be executed over any database. But they can be scoped by providing a specific database name with the `-d/--database` option. Also, by default, only the user who stored the query can access that stored query. Using the `--shared` flag will allow other users to execute the stored query.

The following example stores a shared query with a custom name that can be executed over only the database `myDb`:

```
$ stardog-admin stored add --shared -d myDb -n listProperties "select distinct ?p {?s ?p ?o}"
```

The JSON attributes which correspond to `--shared` and `-d` are `shared` and `database`.

Stored query names must be unique for a Stardog instance. Existing stored queries can be replaced using the `--overwrite` option in the command.

Running Stored Queries

Stored queries can be executed using the regular query execution CLI command by passing the name of the stored query:

```
$ stardog query myDb listProperties
```

Other commands like `query explain` also accept stored query names. They can also be passed instead of query string into HTTP API calls.

Listing Stored Queries

To see all the stored queries, use the `stored list` subcommand:

```
$ stardog-admin stored list
```

The results are formatted tabularly:

```
+--------+-----------------------------------------+
|  Name  |              Query String               |
+--------+-----------------------------------------+
| graphs | SELECT ?graph (count(*) as ?size)       |
|        | FROM NAMED stardog:context:all          |
|        | WHERE { GRAPH ?graph {?s ?p ?o}}        |
|        | GROUP BY ?graph                         |
|        | ORDER BY desc(?size)                    |
| people | CONSTRUCT WHERE {                       |
|        |    ?person a foaf:Person ;              |
|        |            ?p ?o                        |
|        | }                                       |
| types  | SELECT DISTINCT ?type ?label            |
|        | WHERE {                                 |
|        |    ?s a ?type .                         |
|        |    OPTIONAL { ?type rdfs:label ?label } |
|        | }                                       |
+--------+-----------------------------------------+

3 stored queries
```

Users can only see the queries they've stored and the queries stored by other users that have been `--shared`. The `--verbose` option will show more details about the stored queries.

Stored queries can be obtained via HTTP:

```
GET /admin/queries/stored
```

The results will be returned in JSON, for example:

```
{
  "queries": [
    {
      "name": "types",
      "creator": "admin",
      "database": "*",
      "query": "select distinct ?type {?s a ?type}",
      "shared": false
    }
  ]
}
```

Removing Stored Queries

Stored queries can be removed using the `stored remove` command:

```
$ stardog-admin stored remove storedQueryName
```

If you would like to clear all the stored queries then use the `-a/--all` option:

```
$ stardog-admin stored remove -a
```

Stored queries can also be removed via HTTP:

```
DELETE /admin/queries/stored/{name}
```

Managing Running Queries

Stardog includes the capability to manage running queries according to configurable policies set at run-time; this capability includes support for **listing** running queries; **deleting** running queries; **reading** the status of a running query; **killing** running queries that exceed a time threshold automatically; and **logging** slow queries for analysis.

Stardog is pre-configured with sensible **server-wide** defaults for query management parameters; these defaults may be overridden or disabled per database, or even per query.

Configuring Query Management

For many uses cases the default configuration will be sufficient. But you may need to tweak the timeout parameter to be longer or shorter, depending on the hardware, data load, queries, throughput, etc. The default configuration has a server-wide query timeout value of `query.timeout`, which is inherited by all the databases in the server. You can customize the server-wide timeout value and then set per-database custom values, too. Any database without a custom value inherits the server-wide value. To disable query timeout, set `query.timeout` to `0`. If individual queries need to set their own timeout, this can be done (by passing a `timeout` parameter over HTTP or using the `--timeout` flag on the CLI), but only if the `query.timeout.override.enabled` property is set to true for the database (true is the default).

Listing Queries

To see all running queries, use the `query list` subcommand:

```
$ stardog-admin query list
```

The results are formatted tabularly:

```
+----+----------+-------+--------------+
| ID | Database | User  | Elapsed time |
+----+----------+-------+--------------+
| 2  | test     | admin | 00:00:20.165 |
| 3  | test     | admin | 00:00:16.223 |
| 4  | test     | admin | 00:00:08.769 |
+----+----------+-------+--------------+

3 queries running
```

You can see which user owns the query (superuser's can see all running queries), as well as the elapsed time and the database against which the query is running. The ID column is the key to deleting queries.

Deleting Queries

To delete a running query, simply pass its ID to the `query kill` subcommand:

```
$ stardog-admin query kill 3
```

The output confirms the query kill completing successfully:

```
Query 3 killed successfully
```

Automatically Killing Queries

For production use, especially when a Stardog database is exposed to arbitrary query input, some of which may not execute in an acceptable time, the automatic query killing feature is useful. It will protect a Stardog Server from queries that consume too many resources.

Once the execution time of a query exceeds the value of `query.timeout`, the query will be killed automatically.[11 (#_footnote_11)] The client that submitted the query will receive an error message. The value of `query.timeout` may be overriden by setting a different value (smaller or longer) in database options. To disable, set to `query.timeout` to `0`.

The value of `query.timeout` is a positive integer concatenated with a letter, interpreted as a time duration: 'h' (for hours), 'm' (for minutes), 's' (for seconds), or 'ms' (for milliseconds). For example, '1h' for 1 hour, '5m' for 5 minutes, '90s' for 90 seconds, and '500ms' for 500 milliseconds.

The default value of `query.timeout` is five minutes.

Query Status

To see more detail about query in-flight, use the `query status` subcommand:

```
$ stardog-admin query status 1
```

The resulting output includes query metadata, including the query itself:

```
Username: admin
Database: test
Started : 2013-02-06 09:10:45 AM
Elapsed : 00:01:19.187
Query   :
select ?x ?p ?o1 ?y ?o2
   where {
     ?x ?p ?o1.
     ?y ?p ?o2.
     filter (?o1 > ?o2).
    }
order by ?o1
limit 5
```

Slow Query Logging

Stardog does not log slow queries in the default configuration because there isn't a single value for what counts as a "slow query", which is entirely relative to queries, access patterns, dataset sizes, etc. While slow query logging has minimal overhead, what counts as a slow query in some context may be acceptable in another. See Configuring Stardog Server (#_configuring_stardog_server) above for the details.

Protocols and Java API

For HTTP protocol support, see Stardog's Apiary (http://docs.stardog.apiary.io/) docs.

For Java, see the Javadocs (http://stardog.com/docs/java/snarl/).

Security and Query Management

The security model for query management is simple: any user can kill any running query submitted by that user, and a superuser can kill any running query. The same general restriction is applied to query status; you cannot see status for a query that you do not own, and a superuser can see the status of every query.

Managing Query Performance

Stardog answers queries in two major phases: determining the query plan and executing that plan to obtain answers from the data. The former is called *query planning* (or *query optimization*) and includes all steps required to select the most efficient way to execute the query. How Stardog evaluates a query can only be understood by analyzing the query plan. Query plan analysis is also the main tool for investigating performance issues as well as addressing them, in particular, by re-formulating the query to make it more amenable to optimization.

TIP

- 7 Steps to Fast SPARQL Queries (https://www.stardog.com/blog/7-steps-to-fast-sparql-queries/)
- Avoidably Slow Queries (https://www.stardog.com/blog/7-steps-to-fast-sparql-queries/2/)
- How to Read Stardog Query Plans (https://www.stardog.com/blog/how-to-read-stardog-query-plans/)

Query Plan Syntax

We will use the following running example to explain query plans in Stardog.

```
SELECT DISTINCT ?person ?name
WHERE {
  ?article rdf:type bench:Article .
  ?article dc:creator ?person .
  ?inproc rdf:type bench:Inproceedings .
  ?inproc dc:creator ?person .
  ?person foaf:name ?name
}
```

This query returns the names of all people who have authored both a journal article and a paper in a conference proceedings. The query plan used by Stardog (in this example, 4.2.2) to evaluate this query is:

```
Distinct [#812K]
`─ Projection(?person, ?name) [#812K]
   `─ MergeJoin(?person) [#812K]
      +─ MergeJoin(?person) [#391K]
      │  +─ Sort(?person) [#391K]
      │  │  `─ MergeJoin(?article) [#391K]
      │  │     +─ Scan[POSC](?article, rdf:type, bench:Article) [#208K]
      │  │     `─ Scan[PSOC](?article, dc:creator, ?person) [#898K]
      │  `─ Scan[PSOC](?person, foaf:name, ?name) [#433K]
      `─ Sort(?person) [#503K]
         `─ MergeJoin(?inproc) [#503K]
            +─ Scan[POSC](?inproc, rdf:type, bench:Inproceedings) [#255K]
            `─ Scan[PSOC](?inproc, dc:creator, ?person) [#898K]
```

The plan is arranged in an hierarchical, tree-like structure. The nodes, called *operators*, represent units of data processing during evaluation. They correspond to evaluations of graphs patterns or solution modifiers as defined in SPARQL 1.1 specification (https://www.w3.org/TR/sparql11-query/#sparqlDefinition). All operators can be regarded as functions which may take some data as input and produce some data as output. All input and output data is represented as streams of solutions (https://www.w3.org/TR/sparql11-query/#sparqlSolutions), that is, sets of bindings of the form `x → value` where `x` is a variable used in the query and `value` is some RDF term (IRI, blank node, or literal). Examples of operators include scans, joins, filters, unions, etc.

Numbers in square brackets after each node refer to the *estimated* cardinality of the node, i.e. how many solutions Stardog expects this operator to produce when the query is evaluated. Statistics-based cardinality estimation in Stardog merits a separate blog post, but here are the key points for the purpose of reading query plans:

1. all estimations are approximate and their accuracy can vary greatly (generally: more precise for bottom nodes, less precise for upper nodes)
2. estimations are only used for selecting the best plan but have no bearing on the actual results of the query
3. in most cases a sub-optimal plan can be explained by inaccurate estimations

Stardog Evaluation Model

Stardog generally evaluates query plans according to the bottom-up SPARQL semantics (https://www.w3.org/TR/sparql11-query/#sparqlAlgebraEval). Leaf nodes are evaluated first and without input, and their results are then sent to their parent nodes up the plan. Typical examples of leaf nodes include scans, i.e. evaluations of triple patterns, evaluations of full-text search predicates, and `VALUES` (https://www.w3.org/TR/sparql11-query/#inline-data) operators. They contain all information required to produce output, for example, a triple pattern can be directly evaluated against Stardog indexes. Parent nodes, such as joins, unions, or filters, take solutions as inputs and send their results further towards the root of the tree. The root node in the plan, which is typically one of the solution modifiers (https://www.w3.org/TR/sparql11-query/#solutionModifiers), produces the final results of the query which are then encoded and sent to the client.

Pipelining And Pipeline Breakers

Stardog implements the Volcano model (http://dbms-arch.wikia.com/wiki/Volcano_Model), in which evaluation is as *lazy* as possible. Each operator does just enough work to produce the next solution. This is important for performance, especially for queries with a `LIMIT` clause (of which `ASK` queries are a special case) and also enables Stardog's query engine to send the first result(s) as soon as they are available (as opposed to waiting till all results have been computed).

Not all operators can produce output solutions as soon as they get first input solutions from their children nodes. Some need to accumulate intermediate results before sending output. Such operators are called *pipeline breakers*, and they are often the culprits for performance problems, typically resulting from memory pressure. It is important to be able to spot them in the plan since they can suggest either a way to re-formulate the query to help the planner or a way to make the query more precise by specifying extra constants where they matter.

Here are some important pipeline breakers in the example plan:

- `HashJoin` (https://en.wikipedia.org/wiki/Hash_join) algorithms build a hash table for solutions produced by the right operand. Typically all such solutions need to be hashed, either in memory or spilled to disk, before the first output solution is produced by the `HashJoin` operator.
- `Sort`: the sort operator builds an intermediate *sorted* collection of solutions produced by its child node. The main use case for sorting solutions is to prepare data for an operator which can benefit from sorted inputs, such as `MergeJoin`, `Distinct`, or `GroupBy`. All solutions have to be fetched from the child node before the smallest (w.r.t. the sort key) solution can be emitted.
- `GroupBy`: group-by operators are used for aggregation, e.g. counting or summing results. When evaluating a query like `select ?x (count(?y) as ?count) where { … } group by ?x` Stardog has to scroll through all solutions to compute the count for every `?x` key before returning the first result.

Other operators can produce output as soon as they get input:

- `MergeJoin`: merge join algorithms do a single zig-zag pass over sorted streams of solutions produced by children nodes and output a solution as soon as the join condition is satisfied.
- `DirectHashJoin`: contrary to the classical hash join algorithm, this operator does not build a hash table. It utilizes Stardog indexes for look-ups which doesn't require extra data structures. This is only possible when the right operand is sorted by the join key, but the left isn't, otherwise Stardog would use a merge join.
- `Filter`: a solution modifier which evaluates the filter condition on each input solution.
- `Union`: combines streams of children solutions without any extra work, e.g. joining, so there's no need for intermediate results.

Now, returning to the above query, one can see `Sort` pipeline breakers in the plan:

```
Sort(?person) [#391K]
`─ MergeJoin(?article) [#391K]
   +─ Scan[POSC](?article, rdf:type, bench:Article) [#208K]
   `─ Scan[PSOC](?article, dc:creator, ?person) [#898K]
```

This means that all solutions representing the join of `?article rdf:type bench:Article` and `?article dc:creator ?person` will be put in a sequence ordered by the values of `?person`. Stardog expects to sort `391K` solutions before they can be further merge-joined with the results of the `?person foaf:name ?name` pattern. Alternately the engine may build a hash table instead of sorting solutions; such decisions are made by the optimizer based on a number of factors.

Skipping Intermediate Results

One tricky part of understanding Stardog query plans is that evaluation of each operator in the plan is context-sensitive, i.e. it depends on what other nodes are in the same plan, maybe in a different sub-tree. In particular, the cardinality estimations, even if assumed accurate, only specify how many solutions the operator is expected to produce when evaluated as the root node of a plan.

However, as it is joined with other parts of the plan, the results can be different. This is because Stardog employs optimizations to reduce the number of solutions produced by a node by pruning those which are incompatible with other solutions with which they will later be joined.

Consider the following basic graph pattern and the corresponding plan:

```
?erdoes rdf:type foaf:Person .
?erdoes foaf:name "Paul Erdoes"^^xsd:string .
?document dc:creator ?erdoes .

MergeJoin(?erdoes) [#10]
+— MergeJoin(?erdoes) [#1]
│  +— Scan[POSC](?erdoes, rdf:type, foaf:Person) [#433K]
│  `— Scan[POSC](?erdoes, foaf:name, "Paul Erdoes") [#1]
`— Scan[POSC](?document, dc:creator, ?erdoes) [#898K]
```

The pattern matches all documents created by a person named Paul Erdoes. Here the second pattern is selective (only one entity is expected to have the name "Paul Erdoes"). This information is propagated to the other two scans in the plan via merge joins, which allows them to skip scanning large parts of data indexes.

In other words, the node `Scan[POSC](?erdoes, rdf:type, foaf:Person) [#433K]` will not produce all `433K` solutions corresponding to all people in the database and, similarly, `Scan[POSC](?document, dc:creator, ?erdoes) [#898K]` will not go through all `898K` document creators.

Diagnosing Performance Problems

Performance problems may arise because of two reasons:

1. complexity of the query itself, especially the amount of returned data
2. failure to select a good plan for the query.

It is important to distinguish the two. In the former case the best way forward is to make the patterns in `WHERE` more selective. In the latter case, i.e. when the query returns some modest number of results but takes an unacceptably long time to do so, one needs to look at the plan, identify the bottlenecks (most often, pipeline breakers), and reformulate the query or report it to us for further analysis.

Here's an example of a un-selective query:

```
SELECT DISTINCT ?name1 ?name2
WHERE {
  ?article1 rdf:type bench:Article .
  ?article2 rdf:type bench:Article .
  ?article1 dc:creator ?author1 .
  ?author1 foaf:name ?name1 .
  ?article2 dc:creator ?author2 .
  ?author2 foaf:name ?name2 .
  ?article1 swrc:journal ?journal .
  ?article2 swrc:journal ?journal
  FILTER (?name1<?name2)
  }
```

The query returns all distinct pairs of authors who published (possibly different) articles in the same journal. It returns more than 18M results from a database of 5M triples. Here's the plan:

```
Distinct [#17.7M]
`— Projection(?name1, ?name2) [#17.7M]
   `— Filter(?name1 < ?name2) [#17.7M]
      `— HashJoin(?journal) [#35.4M]
         +— MergeJoin(?author2) [#391K]
         │  +— Sort(?author2) [#391K]
         │  │  `— NaryJoin(?article2) [#391K]
         │  │     +— Scan[POSC](?article2, rdf:type, bench:Article) [#208K]
         │  │     +— Scan[PSOC](?article2, swrc:journal, ?journal) [#208K]
         │  │     `— Scan[PSOC](?article2, dc:creator, ?author2) [#898K]
         │  `— Scan[PSOC](?author2, foaf:name, ?name2) [#433K]
         `— MergeJoin(?author1) [#391K]
            +— Sort(?author1) [#391K]
            │  `— NaryJoin(?article1) [#391K]
            │     +— Scan[POSC](?article1, rdf:type, bench:Article) [#208K]
            │     +— Scan[PSOC](?article1, swrc:journal, ?journal) [#208K]
            │     `— Scan[PSOC](?article1, dc:creator, ?author1) [#898K]
            `— Scan[PSOC](?author1, foaf:name, ?name1) [#433K]
```

This query requires an expensive join on `?journal` which is evident from the plan (it's a hash join in this case). It produces more than 18M results (Stardog expects 17.7M which is pretty accurate here) that need to be filtered and examined for duplicates. Given all this information from the plan, the only reasonable way to address the problem would be to restrict the criteria, e.g. to particular journals, people, time periods, etc.

If a query is well-formulated and selective, but performance is unsatisfactory, one may look closer at the pipeline breakers, e.g. this part of the query plan:

```
MergeJoin(?person) [#391K]
+─ Sort(?person) [#391K]
|  `─ MergeJoin(?article) [#391K]
|     +─ Scan[POSC](?article, rdf:type, bench:Article) [#208K]
|     `─ Scan[PSOC](?article, dc:creator, ?person) [#898K]
`─ Scan[PSOC](?person, foaf:name, ?name) [#433K]
```

A reasonable thing to do would be to evaluate the join of `?article rdf:type bench:Article` and `?article dc:creator ?person` separately, i.e. as a separate queries, to see if the estimation of `391K` is reasonably accurate and to get an idea about memory pressure. This is a valuable piece of information for a performance problem report, especially when the data cannot be shared with us. Similar analysis can be done for hash joins.

In addition to pipeline breakers, there could be other clear indicators of performance problems. One of them is the presence of `LoopJoin` nodes in the plan. Stardog implements the nested loop join (https://en.wikipedia.org/wiki/Nested_loop_join) algorithm which evaluates the join by going through the Cartesian product of its inputs. This is the slowest join algorithm and it is used only as a last resort. It sometimes, but not always, indicates a problem with the query.

Here's an example:

```
SELECT DISTINCT ?person ?name
WHERE {
  ?article rdf:type bench:Article .
  ?article dc:creator ?person .
  ?inproc rdf:type bench:Inproceedings .
  ?inproc dc:creator ?person2 .
  ?person foaf:name ?name .
  ?person2 foaf:name ?name2
  FILTER (?name=?name2)
}
```

The query is similar to an earlier query plan we saw but runs much slower. The plan shows why:

```
Distinct [#98456.0M]
`─ Projection(?person, ?name) [#98456.0M]
   `─ Filter(?name = ?name2) [#98456.0M]
      `─ LoopJoin(_) [#196912.1M]
         +─ MergeJoin(?person) [#391K]
         │  +─ Sort(?person) [#391K]
         │  │  `─ MergeJoin(?article) [#391K]
         │  │     +─ Scan[POSC](?article, rdf:type, bench:Article) [#208K]
         │  │     `─ Scan[PSOC](?article, dc:creator, ?person) [#898K]
         │  `─ Scan[PSOC](?person, foaf:name, ?name) [#433K]
         `─ MergeJoin(?person2) [#503K]
            +─ Sort(?person2) [#503K]
            │  `─ MergeJoin(?inproc) [#503K]
            │     +─ Scan[POSC](?inproc, rdf:type, bench:Inproceedings) [#255K]
            │     `─ Scan[PSOC](?inproc, dc:creator, ?person2) [#898K]
            `─ Scan[PSOC](?person2, foaf:name, ?name2) [#433K]
```

The loop join near the top of the plan computes the Cartesian product of the arguments which produces almost 200B solutions. This is because there is no shared variable between the parts of the query which correspond to authors of articles and conference proceedings papers, respectively. The filter condition `?name = ?name2` cannot be transformed into an equi-join because the semantics of term equality (https://www.w3.org/TR/sparql11-query/#OperatorMapping) used in filters is different from the solution compatibility (https://www.w3.org/TR/sparql11-query/#BasicGraphPattern) semantics used for checking join conditions.

The difference manifests itself in the presence of numerical literals, e.g. `"1"^^xsd:integer` = `"1.0"^^xsd:float`, where they are different RDF terms. However, as long as all names in the data are strings, one can re-formulate this query by renaming `?name2` to `?name` which would enable Stardog to use a more efficient join algorithm.

Query Plan Operators

The following operators are used in Stardog query plans:

- `Scan[Index]`: evaluates a triple/quad pattern against Stardog indexes. Indicates the index used, e.g. `CSPO` or `POSC`, where `S,P,O,C` stand for the kind of lexicographic ordering of quads that the index provides. `SPOC` means that the index is sorted first by *S*ubject, then *P*redicate, *O*bject, and *C*ontext (named graph IRI).
- `HashJoin(join key)`: hash join algorithm, hashes the right operand. **Pipeline breaker**.
- `DirectHashJoin(join key)`: a hash join algorithm which directly uses indexes for lookups instead of building a hash table. Not a pipeline breaker.
- `MergeJoin(join key)`: merge join algorithm, the fastest option for joining two streams of solutions. Requires both operands be sorted on the join key. Not a pipeline breaker.
- `LoopJoin`: the nested loops join algorithm, the slowest join option. Not a pipeline breaker.
- `Sort(sort key)`: sorts the argument solutions by the sort key, typically used as a part of a merge join. **Pipeline breaker**.
- `Filter(condition)`: filters argument solutions according to the condition. Not a pipeline breaker.
- `Union`: combines streams of argument solutions. If both streams are sorted by the same variable, the result is also sorted by that variable. Not a pipeline breaker.
- `PropertyPath`: evaluates a property path (https://www.w3.org/TR/sparql11-query/#propertypaths) pattern against Stardog indexes. Not a pipeline breaker.
- `GroupBy`: groups results of the child operator by values of the group-by expressions (i.e. keys) and aggregates solutions for each key. **Pipeline breaker** (unless the input is sorted by first key).
- `Distinct`: removes duplicate solutions from the input. Not a pipeline breaker but accumulates solutions in memory as it runs so the memory pressure increases as the number of unique solutions increases.
- `VALUES`: produces the inlined results (https://www.w3.org/TR/sparql11-query/#inline-data) specified in the query. Not a pipeline breaker.
- `Search`: evaluates a full-text search predicates against the Lucene index within a Stardog database.
- `Projection`: projects variables as results of a query or a sub-query. Not a pipeline breaker.
- `Bind`: evaluates expressions on each argument solution and binds their values to (new) variables. Not a pipeline breaker.
- `Empty` and `Singleton`: correspond to the empty solution set and a single empty solution, respectively.
- `Type`: reasoning operator for evaluating patterns of the form `?x rdf:type ?type` or `:instance rdf:type ?type`. Not a pipeline breaker.
- `Property`: operator for evaluating triple patterns with unbound predicate with reasoning. Not a pipeline breaker.
- `Service`: SPARQL federation operator which evaluate a pattern against a remote SPARQL endpoint (or a virtual graph (http://docs.stardog.com/#_structured_data_aka_virtual_graphs)).

Using Query Hints

Query hints help Stardog generate optimized query plans. They are implemented as SPARQL comments started with the `pragma` keyword.

The `equality.identity` hint expects a comma-separated list of variables. It tells Stardog that these variables will be bound to RDF terms (IRIs, bnodes, or literals) for which equality coincides with identity (i.e. any term is equal only to itself). This is not true for literals of certain numerical datatypes [cf. Operator Mapping] (https://www.w3.org/TR/sparql11-query/#OperatorMapping (https://www.w3.org/TR/sparql11-query/#OperatorMapping)). However assuming that the listed variables do not take on values of such datatypes can sometimes lead to faster query plans, for example, because of converting some filters to joins and through value inlining.

```
SELECT ?o ?o2 WHERE {
  #pragma equality.identity ?o,?o2
  :a :p  ?o .
  :b :p ?o2 .
}
```

Sometimes our query planner can produce sub-optimal join orderings. The `group.joins` hint introduces an explicit scoping mechanism to help with join order optimization. Patterns in the scope of the hint, given by the enclosing `{}`, will be joined together before being joined with anything else. This way, you can tell the query planner what you think is the optimal way to join variables.

```
select ?s where {
  ?s :p ?o1 .
  {
    #pragma group.joins
    #these patterns will be joined first, before being joined with the other pattern
    ?s :p ?o2 .
    ?o1 :p ?o3 .
  }
}
```

The `push.filters` hint controls how the query optimizer pushes filters down the query plan. There are three possible values: `default`, `aggressive`, and `off`. The `aggressive` option means that the optimizer will push every filter to the deepest operator in the plan which binds variables used in the filter expression. The `off` option turn the optimization off and each filter will be applied to the top operator in the filter's graph pattern (in case there're multiple filters, their order is not specified). Finally, the `default` option (or absence of the hint) means that the optimizer will decide whether to push each filter down the plan based on various factors, e.g. the filter's cost, selectivity of the graph pattern, etc.

```
select ?s where {
  #pragma push.filters off
  #the filter in the top scope will not be pushed into the union
  ?s :p ?o1 .
  FILTER (?o2 > 10)
  {
    #pragma push.filters aggressive
    #the optimizer will place this filter directly on top of ?s :r ?o3
    #and it will be evaluated before the results are joined with ?s :p ?o2
    ?s :p ?o2 ;
       :r ?o3 .
    FILTER (?o3 > 1000)
  }
  UNION
  {
    #pragma push.filters default
    #the optimizer will decide whether to place the filter directly
    #on top of ?s :q ?o3 or leave it on top of the join
    ?s a :Type ;
       :q ?o3 .
    FILTER (?o3 < 50)
  }
}
```

ACID Transactions

What follows is specific guidance about Stardog's transactional semantics and guarantees.[12 (#_footnote_12)]

Atomicity

Databases may guarantee atomicity—groups of database actions (i.e., mutations) are irreducible and indivisible: either all the changes happen or none of them happens. Stardog's transacted writes are atomic. Stardog does not support nested transactions.[13 (#_footnote_13)]

Consistency

Data stored should be valid according to the data model (in this case, RDF) and to the guarantees offered by the database, as well as to any application-specific integrity constraints that may exist. Stardog's transactions are guaranteed not to violate integrity constraints during execution. A transaction that would leave a database in an inconsistent or invalid state is aborted.

See the Validating Constraints (#_validating_constraints) section for a more detailed consideration of Stardog's integrity constraint mechanism.

Isolation

A Stardog connection will run in `READ COMMITTED` (http://en.wikipedia.org/wiki/Isolation_(database_systems)#Read_committed) isolation level if it has not started an explicit transaction and will run in `READ COMMITTED SNAPSHOT` or `SERIALIZABLE` isolation level depending on the value of the `transaction.isolation`. In any of these modes, uncommitted changes will only be visible to the connection that made the changes: no other connection can see those values before they are committed. Thus, "dirty reads" can never occur.

The difference between `READ COMMITTED` and `READ COMMITTED SNAPSHOT` isolation levels is that in the former case a connection will see updates committed by another connection immediately, whereas in the latter case a connection will see a transactionally consistent snapshot of the data as it existed at the start of the transaction and will not see any updates.

We illustrate the difference between these two levels with the following example where initially the database has a single triple `:x :val 1`.

4. Table of the difference between RCI and RCSI

Time	Connection 1	Connection 2	Connection 3
0	`SELECT ?val {?x :val ?val}` ⇐ 1	`SELECT ?val {?x :val ?val}` ⇐ 1	`SELECT ?val {?x :val ?val}` ⇐ 1
1	**BEGIN TX**		
2	`INSERT {:x :value 2} DELETE {:x :value ?old}`		
3	`SELECT ?val {?x :val ?val}` ⇐ 2	`SELECT ?val {?x :val ?val}` ⇐ 1	`SELECT ?val {?x :val ?val}` ⇐ 1
4			**BEGIN TX**
5	**COMMIT**		
6	`SELECT ?val {?x :val ?val}` ⇐ 2	`SELECT ?val {?x :val ?val}` ⇐ 2	`SELECT ?val {?x :val ?val}` ⇐ 1
8			`INSERT {:x :value 3} DELETE {:x :value ?old}`
9			**COMMIT**
10	`SELECT ?val {?x :val ?val}` ⇐ 3	`SELECT ?val {?x :val ?val}` ⇐ 3	`SELECT ?val {?x :val ?val}` ⇐ 3

No locks are taken, or any conflict resolution performed, for concurrent transactions in `READ COMMITTED SNAPSHOT` isolation level. If there are conflicting changes, the latest commit wins which may yield unexpected results since every transaction reads from a snapshot that was created at the time transaction started.

Consider the following query being executed by two concurrent threads in `READ COMMITTED SNAPSHOT` isolation level against a database having the triple `:counter :val 1` initially:

```
INSERT { :counter :val ?newValue }
DELETE { :counter :val ?oldValue }
WHERE  { :counter :val ?oldValue
         BIND (?oldValue+1 AS ?newValue) }
```

Since each transaction will read the current value from its snapshot, it is possible that both transactions will read the value `1` and insert the value `2` even though we expect the final value to be `3`.

Isolation level `SERIALIZABLE` can be used to avoid these situations. In `SERIALIZABLE` mode an exclusive lock needs to be acquired before a transaction begins. This ensures concurrent updates cannot interfere with each other, but as a result update throughput will decrease since only one transaction can run at a time.

Durability

By default Stardog's transacted writes are durable and no other actions are required.

Commit Failure Autorecovery

Stardog's transaction framework is low maintenance; but there are some rare conditions in which manual intervention may be needed.

Stardog's strategy for recovering automatically from commit failure is as follows:

1. Stardog will roll back the transaction upon a commit failure;
2. Stardog takes the affected database offline for maintenance;[14 (#_footnote_14)] then
3. Stardog will begin recovery, bringing the recovered database back online once that task is successful so that operations may resume.

With an appropriate logging configuration for production usage (at least error-level logging), log messages for the preceding recovery operations will occur. If for whatever reason the database fails to be returned automatically to online status, an administrator may use the CLI tools (i.e., `stardog-admin db online`) to try to online the database.

Optimizing Bulk Data Loading

Stardog tries hard to do bulk loading at database creation time in the most efficient and scalable way possible. But data loading time can vary widely, depending on factors in the data to be loaded, including the number of unique resources, etc. Here are some tuning tips that may work for you:

1. Use the `bulk_load` memory configuration for loading large databases (see Memory Configuration (#_memory_configuration) section).
2. Load compressed data (#_loading_compressed_data) since compression minimizes disk access
3. Use a multicore machine since bulk loading is highly parallelized and indexes are built concurrently
4. Load many files together at creation time since different files will be parsed and processed concurrently improving the load speed
5. Turn off strict parsing (see Configuring a Database (#_configuring_a_database) for the details).
6. If you are not using named graphs, use triples only indexing (#_database_create_options).

Memory Management

As of version 5.0, Stardog by default uses a custom memory management approach to minimize GC activity during query evaluation. All intermediate query results are now managed in native (off-heap) memory which is pre-allocated on server start-up and never returned to the OS until server shutdown. Every query, including SPARQL Update queries with the WHERE clause, gets a chunk of memory from that pre-allocated pool to handle intermediate results and will return it back to the pool when it finishes or gets cancelled. More technical details about this GC-less memory management scheme are available in a recent blog post (https://blog.stardog.com/saying-goodbye-to-garbage/).

The main goal of this memory management approach is to improve server's resilience under heavy load. A common problem with JVM applications under load is the notorious Out-Of-Memory (OOM) exceptions which are hard to foresee and impossible to reliably recover from. Also, in the SPARQL world, it is generally difficult to estimate how many intermediate results any particular query will have to process before the query starts (although the selectivity statistics offers great help to this end). As such, the server has to deal with the situation when there is no memory available to continue with the current query. Stardog handles this by placing all intermediate results into custom collections which are tightly integrated with the memory manager. Every collection, e.g. for hashing, sorting, or aggregating binding sets, requests memory blocks from the manager and transparently spills data to disk when such requests are denied.

This helps avoid OOMs at any time during query evaluation since running out of memory only means triggering spilling and the query will continue slower because of additional disk access. This also means Stardog 5.0+ can run harder, e.g. analytic, queries which may exceed the memory capacity on your server. We have also seen performance improvements in specific (but common) scenarios, such as with many concurrent queries, where the GC pressure would considerably slow down the server running on heap. However, everything comes at a price and the custom collections can be slightly slower than those based on JDK collections when the server is under light load, all queries are selective, and there is no GC pressure. For that reason Stardog has a server option `memory.management` which you can set to `JVM` in `stardog.properties` to disable custom memory management and have Stardog run all queries on heap.

The `spilling.dir` server option specifies the directory which will be used for spilling data in case the server runs out of native memory. It may make sense to set this to another disk to minimize disk contention.

Memory Configuration

Stardog 5.0 provides a range of configuration options related to memory management. Query engine by default uses the custom memory management approach described above but it is not the only critical Stardog component which may require a large amount of memory. Memory is also consumed aggressively during bulk loading and updates. Stardog defines three standard *memory consumption modes* to allow users to configure how memory should be distributed based on the usage scenario. The corresponding server property is `memory.mode` which accepts the following values:

1. `default`: This is the default option which provides roughly equal amount of memory for queries and updates (including bulk loading). This should be used either when the server is expected to run both read queries and updates in roughly equal proportion or when the expected load is unknown.
2. `read_optimized`: This option provides more memory to read queries and SPARQL Update queries with the WHERE clause. This minimizes the chance of having to spill data to disk during query execution at the expense of update and bulk loading operations. This option should be used when the transactions will be infrequent or small in size, e.g. up to a thousand triples since such transactions do not use significant amount of memory.
3. `write_optimized`: This option should be used for optimal loading and update performance. Queries may run slower if there is not enough memory for processing intermediate results. It may be also suitable when the server is doing a lot of updates and some read queries but the latter are selective and are not highly concurrent.
4. `bulk_load`: This option should be used for bulk loading very large databases (billions of triples) where there is no other workload on the server. When bulk loading is complete, the memory configuration should be changed and the server restarted.

As with any server option the server has to be restarted after the user changes the memory mode. The `stardog-admin server status` command displays detailed information on memory usage and the current configuration.

Capacity Planning

The primary system resources used by Stardog are CPU, memory, and disk.[15 (#_footnote_15)] Stardog will take advantage of more than one CPU, core, and core-based thread in data loading and in throughput-heavy or multi-user loads. Stardog performance is influenced by the speed of CPUs and cores. But some workloads are bound by main memory or by disk I/O (or both) more than by CPU. Use the fastest CPUs you can afford with the largest secondary caches and the most number of cores and core-based threads of execution, especially in multi-user workloads.

The following subsections provides more detailed guidance for the memory and disk resource requirements of Stardog.

Memory usage

Stardog uses system memory aggressively and the total system memory available to Stardog is often the most important factor in performance. Stardog uses both JVM memory (heap memory) and also the operating system memory outside the JVM (off heap memory). Having more system memory available is always good; however, increasing JVM memory too close to total system memory is not prudent as it may tend to increase Garbage Collection (GC) time in the JVM.

The following table shows recommended JVM memory and system memory requirements for Stardog.[16 (#_footnote_16)]

5. Table of Memory Usage for Capacity Planning

# of Triples	JVM Memory	Off-heap memory
100 million	3GB	3GB
1 billion	4GB	8GB
10 billion	8GB	64GB
20 billion	16GB	128GB
50 billion	16GB	256GB

Out of the box, Stardog sets the maximum JVM memory to 2GB and off-heap memory to 1GB. These settings work fine for most small databases (up to, say, 100 million triples). As the database size increases, we recommend increasing memory. You can increase the memory for Stardog by setting the system property `STARDOG_SERVER_JAVA_ARGS` using the standard JVM options. For example, you can set this property to `"-Xms4g -Xmx4g -XX:MaxDirectMemorySize=8g"` to increase the JVM memory to 4GB and off-heap to 8GB. We recommend setting the minimum heap size (`-Xms` option) and max heap size (`-Xmx` option) to the same value.

System Memory and JVM Memory

Stardog uses an off-heap, custom memory allocation scheme. Please note that the memory provisioning recommendations above are for two kinds of memory allocations for the JVM in which Stardog will run. The first is for memory that the JVM will manage explicitly (i.e., "JVM memory"); and the second, i.e., "Off-heap memory" is for memory that Stardog will manage explicitly, i.e., off the JVM heap, but for which the JVM should be notified via the `MaxDirectMemorySize` property. The sum of these two values should be less than 90% of the total memory available to the underlying operating system as requirements dictate.

Disk usage

Stardog stores data on disk in a compressed format. The disk space needed for a database depends on many factors besides the number of triples, including the number of unique resources and literals in the data, average length of resource identifiers and literals, and how much the data is compressed. The following table shows typical disk space used by a Stardog database.

6. Table of Typical Disk Space Requirements

# of triples	Disk space
1 billion	70GB to 100GB
10 billion	700GB to 1TB

These numbers are given for information purposes only; the actual disk usage for a database may be different in practice. Also it is important to realize the amount of disk space needed at creation time for bulk loading data is higher as temporary files will be created. The extra disk space needed at bulk loading time can be 40% to 70% of the final database size.

Disk space used by a database is non-trivially smaller if triples-only indexing (#_database_create_options) is used. Triples-only indexing reduces overall disk space used by 25% on average; however, note the tradeoff: SPARQL queries involving named graphs perform better with quads indexing.

The disk space used by Stardog is additive for more than one database and there is little disk space used other than what is required for the databases. To calculate the total disk space needed for more than one database, one may sum the disk space needed by each database.

Using Stardog on Windows

Stardog provides batch (`.bat`) files for use on Windows; they offer roughly the same set of functionality provided by the Bash scripts which are used on Unix-like systems. There are, however, a few small differences between the two. When you start a server with `server start` on Windows, this does not detach to the background, it will run in the current console.

To shut down the server correctly, you should either issue a `server stop` command from another window or press `Ctrl` + `C` (and then `Y` when asked to terminate the batch job). Do not under any circumstance close the window without shutting down the server. This will simply kill the process without shutting down Stardog, which may cause your database to be corrupted.

The `.bat` scripts for Windows support our standard `STARDOG_HOME` and `STARDOG_SERVER_JAVA_ARGS` environment variables which can be used to control where Stardog's database is stored and how much memory is given to Stardog's JVM when it starts. By default, the script will use the JVM that is available in the directory from which Stardog is run via the `JAVA_HOME` environment variable. If this is not set, it will simply execute `java` from within that directory.

Running Stardog as a Windows Service

You can run Stardog as a Windows Service using the following configuration. Please, note, that the following assumes commands are executed from a Command Prompt with administrative privileges.

Installing the Service

Change the directory to the Stardog installation directory:

```
cd c:\stardog-$VERSION
```

Configuring the Service

The default settings with which the service will be installed are

- 2048 MB of RAM
- `STARDOG_HOME` is the same as the installation directory
- the name of the installed service will be "Stardog Service"
- Stardog service will write logs to the "logs" directory within the installation directory

To change these settings, set appropriate environment variables:

- `STARDOG_MEMORY`: the amount of memory in MB (e.g., set `STARDOG_MEMORY`=4096)
- `STARDOG_HOME`: the path to `STARDOG_HOME` (e.g., set `STARDOG_HOME`=c:\\stardog-home)
- `STARDOG_SERVICE_DISPLAY_NAME`: a different name to be displayed in the list of services (e.g., set `STARDOG_SERVICE_DISPLAY_NAME`=Stardog Service)
- `STARDOG_LOG_PATH`: a path to a directory where the log files should be written (e.g., set `STARDOG_LOG_PATH`=c:\\stardog-logs)

If you have changed the default administrator password, you also need to change `stop-service.bat` and specify the new username and password there (by passing `-u` and `-p` parameters in the line that invokes `stardog-admin server stop`).

Installing Stardog as a Service

Run the `install-service.bat` script.

At this point the service has been installed, but it is not running. To run it, see the next section or use any Windows mechanism for controlling the services (e.g., type `services.msc` on the command line).

Starting, Stopping, & Changing Service Configuration

Once the service has been installed, execute `stardog-serverw.exe`, which will allow you to configure the service (e.g., set whether the service is started automatically or manually), manually start and stop the service, as well as to configure most of the service parameters.

Uninstalling the Stardog Service

The service can be uninstalled by running `uninstall-service.bat` script.

Using Stardog with Kerberos

Stardog can be configured to run in both MIT and Active Directory Kerberos environments. In order to do so a `keytab` (https://web.mit.edu/kerberos/krb5-1.12/doc/basic/keytab_def.html) file must be properly created.

Once the keytab file is acquired the following options can be set in `stardog.properties`:[17 (#_footnote_17)]

1. `krb5.keytab`: The path to the keytab file for the Stardog server.
2. `krb5.principal`: The principal of the credential stored in the `keytab` file. Often this is of the format HTTP/<canonical DNS name of the host>@<REALM>.
3. `krb5.admin.principal`: The Kerberos principal that will be the default administrator of this service.
4. `krb5.debug`: A boolean value to enable debug logging in the Java Kerberos libraries.

Once Stardog is propery configured for Kerberos Stardog user names should be created that match their associated Kerberos principal names. Authentication will be done based on the Kerberos environment and authorization is done based on the principal names matching Stardog users.

TIP Kerberos: Three-Headed Stardog (https://www.stardog.com/blog/kerberos-three-headed-stardog/)

ENTERPRISE DATA UNIFICATION

Stardog is an Enterprise Knowledge Graph platform, which means that it's also a data unification platform. Enterprises have lots of data and lots of data sources and almost all of them are locked away in IT silos and stovepipes that impede insight, analysis, reporting, compliance, and operations.

State of the art IT management tells us to **organize data, systems, assets, staffs, schedules, and budgets** vertically to mirror lines of business. But increasingly all the internal and external demands on IT are **horizontal** in nature: the data is organized vertically, but the enterprise increasingly needs to access and understand it horizontally.

Structured Data

Stardog supports a set of techniques for unifing **structured enterprise data**, chiefly, **Virtual Graphs**: tabular data declaratively mapped into a Stardog database graph and queries by Stardog *in situ*, typically using SQL. Stardog rewrites (a portion of) SPARQL queries against the Stardog database into SQL, issues that SQL query to an RDBMS, and translates the SQL results into SPARQL results. Virtual Graphs can be used for mapping any tabular data, e.g. CSV, to RDF and Stardog will support mapping other tabular formats to graphs in future releases, including XML, JSON, and enterprise directory services.

A Virtual Graph has three components:

- a unique name
- configuration options (#_virtual_graph_configuration_options)
  - data source connection parameters
  - query and data parameters
- mappings for the data

A virtual graph's name must conform to the regular expression `[A-Za-z]{1}[A-Za-z0-9_-]`. The configuration file includes several parameters, including a JDBC connection. Finally, the mappings define how the tabular data stored in the RDBMS will be represented in RDF. The mappings are defined using the R2RML (http://www.w3.org/TR/r2rml/) mapping language, but a simpler Stardog Mapping Syntax (#_stardog_mapping_syntax) is also supported for serializing R2RML mappings.

Supported RDBMSes

Stardog Virtual Graphs supports the following relational database systems; please inquire (mailto:inquiries@stardog.com) if you need support for another.

- Apache Hive
- Apache/Cloudera Impala
- IBM DB2
- H2 & Derby
- Microsoft SQL Server
- MySQL & MariaDB
- Oracle
- PostgreSQL
- Sybase ASE

Managing Virtual Graphs

In order to query a Virtual Graph it first must be registered with Stardog. Adding a new virtual graph is done via the following command:

```
$ stardog-admin virtual add dept.properties dept.ttl
```

The first argument (`dept.properties`) is the configuration file for the virtual graph and the second argument ('dept.ttl') is the mappings file. The name of the configuration file is used as the name of the virtual graph, so the above command registers a virtual graph named `dept`. Configuration files should be in the Java properties file format and provide JDBC data source and virtual graph configuration. The configuration reference is at Virtual Graph Configuration Options (#_virtual_graph_configuration_options). A minimal example looks like this:

```
jdbc.url=jdbc:mysql://localhost/dept
jdbc.username=admin
jdbc.password=admin
jdbc.driver=com.mysql.jdbc.Driver
```

NOTE Stardog does not ship with JDBC drivers. You need to manually copy the JAR file containing the driver to the `STARDOG/server/dbms/` or `STARDOG_EXT` directory so that it will be available to the Stardog server. The server needs to be restarted after the JAR is copied.

The credentials for the JDBC connection need to be provided in plain text. An alternative way to provide credentials is to use the password file (#_using_a_password_file) mechanism. The credentials should be stored in a password file called `services.sdpass` located in `STARDOG_HOME` directory. The password file entries are in the format `hostname:port:database:username:password` so for the above example there should be an entry `localhost:*:dept:admin:admin` in this file. Then the credentials in the configuration file can be omitted.

The configuration file can also contain a property called `base` to specify a base URI (http://www.w3.org/TR/r2rml/#dfn-base-iri) for resolving relative URIs generated by the mappings (if any). If no value is provided, the base URI will be `virtual://myGraph` where `myGraph` is the name of the virtual graph.

The add command by default assumes the mappings are using the Stardog Mapping Syntax (#_stardog_mapping_syntax). Mappings in standard R2RML syntax can be used by adding the `--format r2rml` option in the above command. If there are no mappings provided to the add commands then the R2RML direct mapping (http://www.w3.org/TR/rdb-direct-mapping/) is used.

Registered virtual graphs can be listed:

```
$ stardog-admin virtual list
1 virtual graphs
dept
```

The commands `virtual mappings` and `virtual options` can be used to retrieve the mappings and configuration options associated with a virtual graph respectively. Registered virtual graphs can be removed using the `virtual remove` command. See the Man Pages (#_man_pages) for the details of these commands.

Querying Virtual Graphs

Querying Virtual Graphs is done by using the `GRAPH` clause, using a special graph URI in the form `virtual://myGraph` to query the Virtual Graph named `myGraph`. The following example shows how to query `dept`:

```
SELECT * {
   GRAPH <virtual://dept> {
      ?person a emp:Employee ;
           emp:name "SMITH"
   }
}
```

Virtual graphs are defined globally in Stardog Server. Once they are registered with the server they can be accessed via any Stardog database as allowed by the access rules (#_virtual_graph_security).

We can query the local Stardog database and virtual graph's remote data in a single query. Suppose we have the `dept` virtual graph, defined as above, that contains employee and department information, and the Stardog database contains data about the interests of people. We can use the following query to combine the information from both sources:

```
SELECT * {
   GRAPH <virtual://dept> {
      ?person a emp:Employee ;
           emp:name "SMITH"
   }
   ?person foaf:interest ?interest
}
```

NOTE Query performance will be best if the `GRAPH` clause for Virtual Graphs is as selective as possible.

Virtual Graph Query Syntax

Virtual Graph queries are implemented by executing a query against the remote data source. This is a powerful feature and care must be taken to ensure peak performance. SPARQL and SQL don't have feature parity, especially given the varying capabilities of SQL implementations. Stardog's query translator supports most of the salient features of SPARQL including:

- Arbitrarily nested subqueries (including solution modifiers)
- Aggregation
- `FILTER` (including most SPARQL functions)
- `OPTIONAL`, `UNION`, `BIND`

That said, there are also limitations on translated queries. This includes:

- SPARQL `MINUS` is not currently translated to SQL
- Transitive reasoning queries are not currently translated to SQL
- Comparisons between objects with different datatypes don't always follow XML Schema semantics
- Named graphs in R2RML are not supported

Virtual Graph Security

Accessing Virtual Graphs can be controlled similar to regular named graphs as explained in the Named Graph Security (#_named_graph_security) section. If named graph security is not enabled for a database, all registered Virtual Graphs in the server will be accessible through that database. If named graph security is enabled for a database, then users will be able to query only the Virtual Graphs for which they have been granted access.

If the virtual graphs contain any sensitive information, then it is recommended to enable named graph security globally by setting `security.named.graphs=true` in `stardog.properties`. Otherwise creating a new database without proper configuration would allow users to access those Virtual Graphs.

Materializing Virtual Graphs

In some cases you need to **materialize the information stored in RDBMS directly into RDF**. There is a special command `virtual import` that can be used to import the contents of the RDBMS into Stardog. The command can be used as follows:

```
$ stardog-admin virtual import myDb dept.properties dept.ttl
```

This command adds all the mapped triples from the RDBMS into the default graph. Similar to `virtual add`, this command assumes Stardog Mapping Syntax (#_stardog_mapping_syntax) by default and can accept R2RML mappings using the `--format r2rml` option.

It is also possible to specify a target named graph by using the `-g`/`--namedGraph` option:

```
$ stardog-admin virtual import -g http://example.com/targetGraph myDb dept.properties dept.ttl
```

This `virtual import` command is equivalent to the following SPARQL update query:

```
ADD <virtual://dept> TO <http://example.com/targetGraph>
```

If the RDBMS contents change over time, and we need to update the materialization results in the future, we can clear the named graph contents and rematerialize again. This can be done by using the `--remove-all` option in `virtual import` or with the following SPARQL query:

```
COPY <virtual://dept> TO <http://example.com/targetGraph>
```

Query performance over materialized graphs will be better as the data will be indexed locally by Stardog, but materialization may not be practical in cases where frequency of change is very high.

CSV as Virtual Graph

Mappings can be used to treat CSV files as Virtual Graphs since they represent tabular data similar to RDBMS tables. But unlike RDBMS tables, CSV files are supported only for importing into Stardog. The same import command above can be used to specify a mappings file and an input CSV file:

```
$ stardog-admin virtual import myDB cars.ttl cars.csv
```

If the input file is using different kind of separators, e.g. tab character, a configuration file can be provided

```
$ stardog-admin virtual import myDB cars.properties cars.ttl cars.tsv
```

The configuration file for CSV files can specify values for the following properties: `csv.separator` (character for separating fields), `csv.quote` (character for strings), `csv.escape` (character for escaping special characters), `csv.header` (boolean value for specifying whether or not the input file has a header line at the beginning). Note that, whitespace characters in Java properties file need to be escaped so if you want to import tab-separated value files set `csv.separator=\t` in the configuration file.

NOTE CSV files are processed on the client side and then sent to the Stardog server. If you run out of memory, you can increase the memory for the client using the `STARDOG_JAVA_ARGS` environment variable. This environment variable should only be set for the `virtual import` command to avoid affecting other client command invocations. This can be done like so:

```
$ STARDOG_JAVA_ARGS="-Xmx1g" stardog-admin virtual import myDB cars.properties cars.ttl cars.csv
```

Stardog Mapping Syntax

The Stardog Mapping Syntax (SMS) is an alternative way to serialize R2RML mappings that is much simpler to read and write and has the same expressive power as R2RML. Mappings written in SMS can be converted to R2RML and vice versa. We will use the example database (http://www.w3.org/TR/r2rml/#example-input-database) from the R2RML specification to explain SMS. The SQL schema that corresponds to this example is

```
CREATE TABLE "DEPT" (
      "deptno" INTEGER UNIQUE,
      "dname" VARCHAR(30),
      "loc" VARCHAR(100));
INSERT INTO "DEPT" ("deptno", "dname", "loc")
       VALUES (10, 'APPSERVER', 'NEW YORK');

CREATE TABLE "EMP" (
      "empno" INTEGER PRIMARY KEY,
      "ename" VARCHAR(100),
      "job" VARCHAR(30),
      "deptno" INTEGER REFERENCES "DEPT" ("deptno"),
      "etype" VARCHAR(30));
INSERT INTO "EMP" ("empno", "ename", "job", "deptno", "etype" )
       VALUES (7369, 'SMITH', 'CLERK', 10, 'PART_TIME');
```

Suppose we would like to represent this information in RDF using the same translation for job codes (http://www.w3.org/TR/r2rml/#example-translationtable) as in the original example:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix emp: <http://example.com/emp/> .
@prefix dept: <http://example.com/dept/> .

dept:10 a dept:Department ;
    dept:location "NEW YORK" ;
    dept:deptno "10"^^xsd:integer .

emp:7369 a emp:Employee ;
    emp:name "SMITH" ;
    emp:role emp:general-office ;
    emp:department dept:10 .
```

SMS looks very similar to the output RDF representation:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix emp: <http://example.com/emp/> .
@prefix dept: <http://example.com/dept/> .
@prefix sm: <tag:stardog:api:mapping:> .

dept:{"deptno"} a dept:Department ;
    dept:location "{\"loc\"}" ;
    dept:deptno "{\"deptno\"}"^^xsd:integer ;
    sm:map [
      sm:table "DEPT" ;
    ] .

emp:{"empno"} a emp:Employee ;
    emp:name "{\"ename\"}" ;
    emp:role emp:{ROLE} ;
    emp:department dept:{"deptno"} ;
    sm:map [
      sm:query """
        SELECT \"empno\", \"ename\", \"deptno\", (CASE \"job\"
            WHEN 'CLERK' THEN 'general-office'
            WHEN 'NIGHTGUARD' THEN 'security'
            WHEN 'ENGINEER' THEN 'engineering'
        END) AS ROLE FROM \"EMP\"
        """ ;
    ] .
```

SMS is based on Turtle, but it’s not valid Turtle since it uses the URI templates (http://www.w3.org/TR/r2rml/#from-template) of R2RML—curly braces can appear in URIs. Other than this difference, we can treat an SMS document as a set of RDF triples. SMS documents use the special namespace `tag:stardog:api:mapping:` that we will represent with the `sm` prefix below.

Every subject in the SMS document that has a `sm:map` property maps a single row from the corresponding table/view to one or more triples. If an existing table/view is being mapped, `sm:table` is used to refer to the table. Alternatively, a SQL query can be provided inline using the `sm:query` property.

The values generated will be a URI, blank node, or a literal based on the type of the value used in the mapping. The column names referenced between curly braces will be replaced with the corresponding values from the matching row.

SMS can be translated to the standard R2RML syntax. For completeness, we provide the R2RML mappings corresponding to the above example:

```
@prefix rr: <http://www.w3.org/ns/r2rml#> .
@prefix emp: <http://example.com/emp#> .
@prefix dept: <http://example.com/dept#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@base <http://example.com/base/> .

<DeptTriplesMap>
    a rr:TriplesMap;
    rr:logicalTable [ rr:tableName "DEPT" ];
    rr:subjectMap [ rr:template "http://data.example.com/dept/{\"deptno\"}" ;
                    rr:class dept:Department ];
    rr:predicateObjectMap [
      rr:predicate       dept:deptno ;
      rr:objectMap    [ rr:column "\"deptno\""; rr:datatype xsd:positiveInteger ]
    ];
    rr:predicateObjectMap [
      rr:predicate      dept:location ;
      rr:objectMap      [ rr:column "\"loc\"" ]
    ].

<EmpTriplesMap>
        a rr:TriplesMap;
    rr:logicalTable [ rr:sqlQuery """
        SELECT "EMP".*, (CASE "job"
            WHEN 'CLERK' THEN 'general-office'
            WHEN 'NIGHTGUARD' THEN 'security'
            WHEN 'ENGINEER' THEN 'engineering'
        END) AS ROLE FROM "EMP"
        """ ];
    rr:subjectMap [
        rr:template "http://data.example.com/employee/{\"empno\"}";
        rr:class emp:Employee
    ];
    rr:predicateObjectMap [
      rr:predicate             emp:name ;
      rr:objectMap    [ rr:column "\"ename\"" ];
    ];
    rr:predicateObjectMap [
        rr:predicate emp:role;
        rr:objectMap [ rr:template "http://data.example.com/roles/{ROLE}" ];
    ];
    rr:predicateObjectMap [
        rr:predicate emp:department;
        rr:objectMap [ rr:template "http://example.com/dept/{\"deptno\"}"; ];
    ].
```

Virtual Graph Configuration Options

The following table lists the allowed options in the virtual graph configuration file. Additionally, connection pool configuration of the built-in Tomcat connection pool are allowed. The set of allowed properties is listed in the Tomcat JDBC Connection Pool (https://tomcat.apache.org/tomcat-9.0-doc/jdbc-pool.html#Common_Attributes) documentation. Any unknown options will be ignored.

7. Table of Virtual Graph Configuration Options

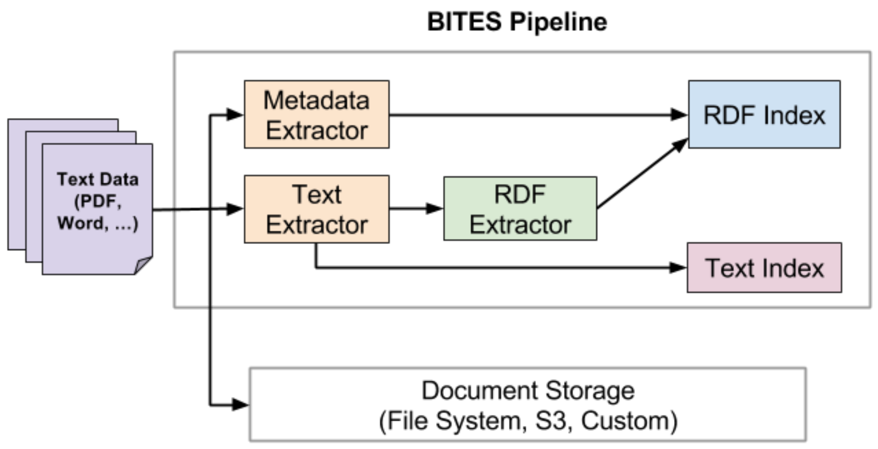
Option	Default
base	
Base IRI used to resolve relative IRIs from virtual graphs.	
`jdbc.url`	

Option	Default
The URL of the JDBC connection.	
`jdbc.username`	
The username used to make the JDBC connection.	
`jdbc.password`	
The password used to make the JDBC connection.	
`jdbc.driver`	
The driver class name used to make the JDBC connection.	
`csv.separator`	`,`
A single-character separator used when importing tabular data files.	
`csv.quote`	`"`
A single character used to used to encapsulate values containing special characters.	
`csv.escape`	
A single character used to escape values containing special characters.	
`csv.header`	`true`
Should the import process read the header row? When headers are enabled the first row of the input file is used to retrieve the column names and mappings can refer to those column names. (`true` / `false`)	
`csv.skip.empty`	`true`
Should empty values be skipped in the CSV file? If `true` no triples will be generated for templates that refer to a column with empty value. (`true` / `false`)	
`percent.encode`	`true`
Should IRI template strings be percent-encoded to be valid IRIs? (`true` / `false`)	
`optimize.import`	`true`
Should `virtual import` and `?s ?p ?o` queries use the optimized translation? (`true` / `false`)	
`parser.sql.quoting`	
If unspecified, R2RML views (using `rr:sqlQuery`) will be parsed using the DB-native identifier quoting convention. For example, MySQL queries will be parsed treating backtick as the identifier quote character. If set to `ANSI` , the ANSI SQL convention of treating a double quote as the identifier quote character will be used instead.	
`query.translation`	
If unspecified, use the core query translation algorithm. If set to `legacy`, the previous algorithm will be used. This option is intended for debugging and not production use.	
`sql.functions`	
A comma-separated list of SQL function names to register with the parser. If an R2RML view (using `rr:sqlQuery`) fails to parse, this option can be set to allow use of non-standard functions.	
`sql.schemas`	
A comma-separated list of schemas to append to the schema search path. This option allows R2RML tables and queries to reference tables that are outside of the default schema for the connected user.	

Option	Default
`default.mapping.include.tables`	
A comma-separated list of tables to include when generating default mappings. If blank, mappings will be generated for all tables in the default schema for the connected user, plus any schemas listed in `sql.schemas` . Cannot be combined with `default.mapping.exclude.tables` .	
`default.mapping.exclude.tables`	
A comma-separated list of tables to exclude when generating default mappings. Mappings will be generated for all tables in the default schema for the connected user, plus any schemas listed in `sql.schemas` , except those tables listed in this option. Cannot be combined with `default.mapping.include.tables` .	

Unstructured Data

Unifying unstructured data is, by necessity, a different process from unifying structured or semistructured data. As of 4.2, Stardog includes a document storage subsystem called **BITES**[18 (#_footnote_18)], which provides configurable storage and processing for unifying unstructured data with the Stardog graph. The following figure shows the main BITES components:



Storage

BITES allows storage and retrieval of documents in the form of files. Stardog treats documents as opaque blobs of data; it defers to the extraction process to make sense of individual documents. Document storage is independent of file and data formats.

Stardog internally stores documents as files. The location of these files defaults to a subdirectory of `STARDOG_HOME` but this can be overridden. Documents can be stored on local filesystem, or an abstraction thereof, accessible from the Stardog server or on Amazon S3 by setting the `docs.filesystem.uri` configuration option. The exact location is given by the `docs.path` configuration option.

Structured Data Extraction

BITES supports an optional processing stage in which a document is processed to extract an RDF graph to add to the database. BITES has the following built-in RDF extractors:

- `tika`: This extractor is based on Apache Tika, collects metadata about the document and asserts this set of RDF statements to a named graph specific to the document.
- `text`: (Since version 5.3) Adds a RDF statement with the full text extracted from the document. Side-effect of this extractor is that a document's text will be indexed by the search index twice: one for the document itself, other for the value of this RDF statement.
- `entities`: [*Beta*] (Since version 5.2) This extractor uses OpenNLP (https://opennlp.apache.org/) to extract all the mentions of named entities from the document and adds this information to the document named graph.

- `linker`: [*Beta*] (Since version 5.2) This extractor works just like `entities` but after it finds a named entity mention in the document it also finds the entity in the database that best matched that mention.
- `dictionary`: [*Beta*] (Since version 5.2.3) Similar to `linker`, but using a user-provided dictionary that maps named entity mentions to IRIs.
- `CoreNLPMentionRDFExtractor`, `CoreNLPMentionRDFExtractor`, and `CoreNLPRelationRDFExtractor` available through the bites-corenlp (https://github.com/stardog-union/bites-corenlp) repository.

See Entity Extraction and Linking (#_entity_extraction_and_linking) section for more details about some of these extractors.

Text Extraction

The document store is fully integrated with Stardog's Search (#_search). As with RDF extraction, text extraction supports arbitrary file formats and pluggable extractors are able to retrieve the textual contents of a document for indexing. Once a document is added to BITES, its contents can be searched in the same way as other literals using the standard `textMatch` predicate in SPARQL queries.

Managing Documents

CRUD operations on documents can be performed from the command line, Java API or HTTP API. Please refer to the StardocsConnection (/docs/5.3.6/java/snarl/com/complexible/stardog/docs/stardocsconnection) API for details of using the document store from Java.

The following is an example session showing how to manage documents from the command line:

```
# We have a document stored in the file `whyfp90.pdf' which we will add to the document store
$ ls -al whyfp90.pdf
-rw-r--r-- 1 user user 200007 Aug 30 09:46 whyfp90.pdf

# We add it to the document store and receive the document's IRI as a return value
$ bin/stardog doc put myDB whyfp90.pdf
Successfully put document in the document store: tag:stardog:api:docs:myDB:whyfp90.pdf

# Adding the same document again will delete all previous extraction results and insert new ones.
# By setting the correct argument, previous assertions will be kept, and new ones appended.
$ bin/stardog doc put myDB —keep-assertions -r text whyfp90.pdf
Successfully put document in the document store: tag:stardog:api:docs:myDB:whyfp90.pdf

# Alternatively, we can add it with a different name. Repeated calls
# will update the document and refresh extraction results
$ bin/stardog doc put myDB --name why-functional-programming-matters.pdf whyfp90.pdf
Successfully put document in the document store: tag:stardog:api:docs:myDB:why-functional-programming-
matters.pdf

# We can subsequently retrieve documents and store them locally
$ bin/stardog doc get myDB whyfp90.pdf
Wrote document 'whyfp90.pdf' to file 'whyfp90.pdf'

# Local files will not be overwritten
$ bin/stardog doc get myDB whyfp90.pdf
File 'whyfp90.pdf' already exists. You must remove it or specify a different filename.

# How many documents are in the document store?
$ bin/stardog doc count myDB
Count: 2 documents

# Removing a document will also clear it's named graph and full-text search index entries
$ bin/stardog doc delete myDB whyfp90.pdf
Successfully executed deletion.

# Re-indexing the docstore allows to apply a different rdf or text extractor
# to all the documents, refreshing extraction results
$ bin/stardog doc reindex myDB -r entities
"Re-indexed 1 documents"
```

See the Man Pages (#_man_pages) for more details about the CLI commands.

Named Graphs and Document Queries

Documents in BITES are identified by IRI. As shown in the command line examples above, the IRI is returned from a document `put` call. The IRI is a combination of a prefix, the database name, and the document name. The CLI uses the document name to refer to the documents. The RDF index, and therefore SPARQL queries, use the IRIs to refer to the documents. RDF assertions extracted from a document are placed into a named graph identified by the document's IRI.

Here we can see the results of querying a document's named graph when using the default metadata extractor:

```
$ stardog query execute myDB "select ?p ?o { graph <tag:stardog:api:docs:myDB:whyfp90.pdf> { ?s ?p ?o } }"

+--------------------------------------------+--------------------------------------+
|                     p                      |                  o                   |
+--------------------------------------------+--------------------------------------+
| rdf:type                                   | http://xmlns.com/foaf/0.1/Document   |
| rdf:type                                   | tag:stardog:api:docs:Document        |
| tag:stardog:api:docs:fileSize              | 200007                               |
| http://purl.org/dc/elements/1.1/identifier | "whyfp90.pdf"                        |
| rdfs:label                                 | "whyfp90.pdf"                        |
| http://ns.adobe.com/pdf/1.3/PDFVersion     | "1.3"                                |
| http://ns.adobe.com/xap/1.0/CreatorTool    | "TeX"                                |
| http://ns.adobe.com/xap/1.0/t/pg/NPages    | 23                                   |
| http://purl.org/dc/terms/created           | "2006-05-19T13:42:00Z"^^xsd:dateTime |
| http://purl.org/dc/elements/1.1/format     | "application/pdf; version=1.3"       |
| http://ns.adobe.com/pdf/1.3/encrypted      | "false"                              |
+--------------------------------------------+--------------------------------------+

Query returned 11 results in 00:00:00.045
```

Entity Extraction and Linking

BITES, by default, uses the `tika` RDF extractor that only extracts metadata from documents. Stardog can be configured to use the OpenNLP (http://opennlp.sourceforge.net) library to detect named entities mentioned in documents and optionally link those mentions to existing resources in the database.

Stardog can also be configured to use Stanford's CoreNLP (https://stanfordnlp.github.io/CoreNLP/) library for entity extraction, linking, and relationship extraction. More information in the bites-corenlp (https://github.com/stardog-union/bites-corenlp) repository.

TIP

- Entity Linking in the Knowledge Graph (https://www.stardog.com/blog/entity-linking-in-the-knowledge-graph/)
- Extending NLP (https://www.stardog.com/blog/extending-nlp/)
- Link All the Entities! (https://www.stardog.com/blog/link-all-the-entities/)
- Augmenting Search (https://www.stardog.com/blog/augmenting-search/)

The first step to use entity extractors is to identify the set of OpenNLP models that will be used. The following models are always required:

- A tokenizer and sentence detector. OpenNLP (http://opennlp.sourceforge.net/models-1.5/) provides models for several languages (e.g., `en-token.bin` and `en-sent.bin`)
- At least one name finder model. Stardog supports both dictionary-based (https://opennlp.apache.org/docs/1.8.2/manual/opennlp.html#tools.cli.dictionary) and custom trained (https://opennlp.apache.org/docs/1.8.2/manual/opennlp.html#tools.namefind.training) models. OpenNLP (http://opennlp.sourceforge.net/models-1.5/) provides models for several types of entities and languages (e.g., `en-ner-person.bin`). We provide our own name finder models (https://complexible.jfrog.io/complexible/stardog-nlp/ner/) created from Wikipedia and DBPedia, which provide high recall / low precision in identifying Person, Organization, and Location types from English language documents.

All this files should be put in the same directory and, after or during database creation time, the configuration option `docs.opennlp.models.path` should be set to its location.

For example, suppose you have a folder `/data/stardog/opennlp` with files `en-token.bin`, `en-sent.bin`, and `en-ner-person.bin`. The database creation command would be as follows:

```
$ stardog-admin db create -o docs.opennlp.models.path=/data/stardog/opennlp -n movies
```

For consistency, model filenames should follow specific patterns:

- `*-token.bin` for tokenizers (e.g., `en-token.bin`)
- `*-sent.bin` for sentence detectors (e.g., `en-sent.bin`)
- `*-ner-*.dict` for dictionary-based name finders (e.g., `dbpedia-en-ner-person.dict`)
- `*-ner-*.bin` for custom trained name finders (e.g., `wikipedia-en-ner-organization.bin`)

Entities

The `entities` extractor detects the mentions of named entities based on the configured models and creates RDF statements for those entities. When we are putting a document we need to specify that we want to use a non-default extractor. We can use both the `tika` metadata extractor and the `entities` extractor at the same time:

```
$ stardog doc put --rdf-extractors tika,entities movies CastAwayReview.pdf
```

The result of entity extraction will be in a named graph where an auto-generated IRI is used for the entity:

```
<tag:stardog:api:docs:movies:CastAwayReview.pdf> {
        <tag:stardog:api:docs:entity:9ad311b4-ddf8-4da2-a49f-3fa8f79813c2> rdfs:label "Wilson" .
        <tag:stardog:api:docs:entity:0d25b4ed-9cd4-4e00-ac3d-f984012b67f5> rdfs:label "Tom Hanks" .
        <tag:stardog:api:docs:entity:e559b828-714f-407d-aa73-7bdc39ee8014> rdfs:label "Robert Zemeckis" .
}
```

Linker

The `linker` extractor performs the same task as `entities` but after the entities are extracted it links those entities to the existing resources in the database. Linking is done by matching the mention text with the identifier and labels of existing resources in the database. This extractor requires the search feature (#_enabling_search) to be enabled to find the matching candidates and uses string similarity metrics to choose the best match. The commonly used properties for labels are supported: `rdfs:label`, `foaf:name`, `dc:title`, `skos:prefLabel` and `skos:altLabel`.

```
$ stardog doc put --rdf-extractors linker movies CastAwayReview.pdf
```

The extraction results of `linker` will be similar to `entities`, but only contain existing resources for which a link was found. The link is available through the `dc:references` property.

```
<tag:stardog:api:docs:movies:CastAwayReview.pdf> {

        <tag:stardog:api:docs:entity:0d25b4ed-9cd4-4e00-ac3d-f984012b67f5> rdfs:label "Tom Hanks" ;
                <http://purl.org/dc/terms/references> <http://www.imdb.com/name/nm0000158> .

        <tag:stardog:api:docs:entity:e559b828-714f-407d-aa73-7bdc39ee8014> rdfs:label "Robert Zemeckis" ;
                <http://purl.org/dc/terms/references> <http://www.imdb.com/name/nm0000709> .
}
```

Dictionary

The `dictionary` extractor fullfills the same purpose as the `linker`, but instead of heuristically trying to match a mention's text with existent resources, it uses a user-defined dictionary to perform that task. The dictionary provides a set of mappings between text and IRIs. Each mention found in the document will be searched in the dictionary and, if found, the IRIs will be added as `dc:references` links.

Dictionaries are `.linker` files, which need to be available in the `docs.opennlp.models.path` folder. Stardog provides several dictionaries (https://complexible.jfrog.io/complexible/stardog-nlp/dictionary/) created from Wikipedia and DBPedia, which allow users to automatically link entity mentions to IRIs in those knowledge bases.

```
$ stardog doc put --rdf-extractors dictionary movies CastAwayReview.pdf
```

When using the `dictionary` option, all `.linker` files in the `docs.opennlp.models.path` folder will be used. The output follows the same syntax as the `linker`.

```
<tag:stardog:api:docs:movies:CastAwayReview.pdf> {

        <tag:stardog:api:docs:entity:0d25b4ed-9cd4-4e00-ac3d-f984012b67f5> rdfs:label "Tom Hanks" ;
                <http://purl.org/dc/terms/references> <http://en.wikipedia.org/wiki/Tom_Hanks> ;
        <http://purl.org/dc/terms/references> <http://dbpedia.org/resource/Tom_Hanks> .
}
```

User-defined dictionaries can be created programmatically. For example, the Java class below will create a dictionary that links every mention of `Tom Hanks` to two IRIs.

```
import java.io.File;
import java.io.IOException;
import com.complexible.stardog.docs.nlp.impl.DictionaryLinker;
import com.google.common.collect.ImmutableMultimap;
import org.openrdf.model.IRI;
import static com.complexible.common.rdf.model.Values.iri;

public class CreateLinker {

        public static void main(String[] args) throws IOException {
                ImmutableMultimap<String, IRI> aDictionary = ImmutableMultimap.<String, IRI>builder()
                                                                    .putAll("Tom Hanks",
iri("https://en.wikipedia.org/wiki/Tom_Hanks"), iri("http://www.imdb.com/name/nm0000158"))
                                                                    .build();

                DictionaryLinker.Linker aLinker = new DictionaryLinker.Linker(aDictionary);

                aLinker.to(new File("/data/stardog/opennlp/TomHanks.linker"));
        }
}
```

SPARQL

Both `entities`, `linker`, and `dictionary` extractors are also available as a SPARQL service, which makes them applicable to any data in the graph, whether stored directly in Stardog or accessed remotely on SPARQL endpoints or virtual graphs.

```
prefix docs: <tag:stardog:api:docs:>

select * {
  ?review :content ?text

  service docs:entityExtractor {
    []  docs:text ?text ;
        docs:mention ?mention
  }
}
```

The `entities` extractor is accessed by using the `docs:entityExtractor` service, which receives one input argument, `docs:text`, with the text to be analyzed. The output will be the extracted named entity mentions, bound to the variable given in the `docs:mention` property.

```
+-------------------------------------------------------------------------------------+------------------+---
------------+
|                                        text                                         |      mention     |
review    |
+-------------------------------------------------------------------------------------+------------------+---
------------+
| "Directed by Robert Zemeckis, featuring Tom Hanks and a volleyball called Wilson" | "Robert Zemeckis"|
:MovieReview  |
| "Directed by Robert Zemeckis, featuring Tom Hanks and a volleyball called Wilson" | "Tom Hanks"      |
:MovieReview  |
| "Directed by Robert Zemeckis, featuring Tom Hanks and a volleyball called Wilson" | "Wilson"         |
:MovieReview  |
+-------------------------------------------------------------------------------------+------------------+---
------------+
```

By adding an extra output variable, `docs:entity`, the `linker` extractor will be used instead.

```
prefix docs: <tag:stardog:api:docs:>

select * {
  ?review :content ?text

  service docs:entityExtractor {
    []  docs:text ?text ;
        docs:mention ?mention ;
        docs:entity ?entity
  }
}
```

```
+-------------------------+------------------+----------------+---------------+
|         text            |      mention     |     entity     |     review    |
+-------------------------+------------------+----------------+---------------+
| "Directed by Robert..." | "Tom Hanks"      | imdb:nm0000158 | :MovieReview  |
| "Directed by Robert..." | "Robert Zemeckis"| imdb:nm0000709 | :MovieReview  |
+-------------------------+------------------+----------------+---------------+
```

The `dictionary` extractor is called in a similar way to `linker`, with an extra argument `docs:mode` set to `docs:Dictionary`.

```
prefix docs: <tag:stardog:api:docs:>

select * {
  ?review :content ?text

  service docs:entityExtractor {
    []  docs:text ?text ;
        docs:mention ?mention ;
        docs:entity ?entity ;
        docs:mode docs:Dictionary
  }
}
```

```
+-------------------------+------------------+---------------------+---------------+
|         text            |      mention     |        entity       |     review    |
+-------------------------+------------------+---------------------+---------------+
| "Directed by Robert..." | "Tom Hanks"      | imdb:nm0000158      | :MovieReview  |
| "Directed by Robert..." | "Tom Hanks"      | wikipedia:Tom_Hanks | :MovieReview  |
+-------------------------+------------------+---------------------+---------------+
```

All extractors accept one more output variable, `docs:type`, which will output the type of entity (e.g., Person, Organization), when available.

```
prefix docs: <tag:stardog:api:docs:>

select * {
  ?review :content ?text

  service docs:entityExtractor {
    []  docs:text ?text ;
        docs:mention ?mention ;
        docs:entity ?entity ;
        docs:type ?type
  }
}
```

```
+-------------------------+------------------+----------------+-----------+---------------+
|         text            |      mention     |     entity     |    type   |     review    |
+-------------------------+------------------+----------------+-----------+---------------+
| "Directed by Robert..." | "Tom Hanks"      | imdb:nm0000158 | :Person   | :MovieReview  |
| "Directed by Robert..." | "Robert Zemeckis"| imdb:nm0000709 | :Person   | :MovieReview  |
+-------------------------+------------------+----------------+-----------+---------------+
```

Custom Extractors

The included extractors are intentionally basic, especially when compared to machine learning or text mining algorithms. A custom extractor connects the document store to algorithms tailored specifically to your data. The extractor SPI allows integration of any arbitrary workflow or algorithm from NLP methods like part-of-speech tagging, entity recognition, relationship learning, or sentiment analysis to machine learning models such as document ranking and clustering.

Extracted RDF assertions are stored in a named graph specific to the document, allowing provenance tracking and versatile querying. The extractor must implement the RDFExtractor (/docs/5.3.6/java/snarl/com/complexible/stardog/docs/extraction/rdfextractor) interface. The convenience class TextProvidingRDFExtractor (/docs/5.3.6/java/snarl/com/complexible/stardog/docs/extraction/tika/textprovidingrdfextractor) is provided which extracts the text from the document before calling the extractor. Entity linking extractors can be tweaked to specific needs by extending their classes, EntityRDFExtractor (java/snarl/com/complexible/stardog/docs/nlp/impl/EntityRDFExtractor.html) and EntityLinkingRDFExtractor (java/snarl/com/complexible/stardog/docs/nlp/impl/EntityLinkingRDFExtractor.html).

The text extractor SPI gives you the opportunity to support arbitrary document formats. Implementations will be given a raw document and be expected to extract a string of text which will be added to the full-text search index. Text extractors should implement the TextExtractor (/docs/5.3.6/java/snarl/com/complexible/stardog/docs/extraction/textextractor) interface.

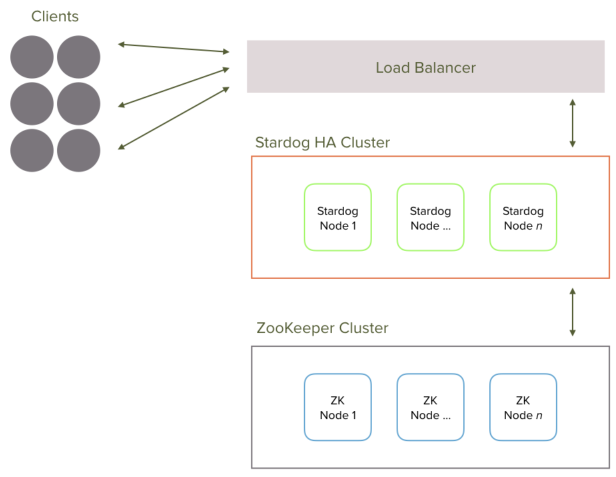
Custom extractors are registered with the Java ServiceLoader under the RDFExtractor (/docs/5.3.6/java/snarl/com/complexible/stardog/docs/extraction/rdfextractor) or TextExtractor (/docs/5.3.6/java/snarl/com/complexible/stardog/docs/extraction/textextractor) class names. Custom extractors can be referred to from the command line or APIs by their fully qualified or "simple" class names.

For an example of a custom extractor, see our github repository (https://github.com/stardog-union/stardog-examples/tree/develop/examples/docs).

HIGH AVAILABILITY CLUSTER

In this section we explain how to configure, use, and administer Stardog Cluster for uninterrupted operations. Stardog Cluster is a collection of Stardog Server instances running on one or more virtual or physical machines that, **from the client's perspective**, behave like a single Stardog Server instance. To fully achieve this effect requires DNS (i.e., with `SRV` records) and proxy configuration that's left as an exercise for the user.

Of course Stardog Cluster should have some different operational properties, the main one of which is high availability. But from the client's perspective Stardog Cluster should be indistinguishable from non-clustered Stardog.[19 (#_footnote_19)] While Stardog Cluster is primarily geared toward HA, it is also important to remember that it should be tuned for your specific use case. Our detailed blog post (https://www.stardog.com/blog/tuning-cluster-for-cloud/) discusses a variety of factors that you should consider when deploying Stardog Cluster as well as some adjustments you should make depending on your workload.



NOTE Stardog Cluster depends on Apache ZooKeeper. High Availability requires **at least** three Stardog and three ZooKeeper nodes in the Cluster. ZooKeeper works best, with respect to fault resiliency, with an ensemble size that is an odd-number greater than or equal to three: 3, 5, 7, etc.**[20]** With respect to performance, larger Stardog clusters perform better than smaller ones for reads, while larger cluster sizes perform worse for writes. It is the responsibility of the administrator to find the right balance.

TIP Tuning Cluster for Cloud (https://www.stardog.com/blog/tuning-cluster-for-cloud/)

Guarantees

A cluster is composed of a set of Stardog servers and a ZooKeeper ensemble running together. One of the Stardog servers is the Coordinator and the others are Participants. The Coordinator orchestrates transactions and maintains consistency by expelling any nodes that fail an operation. An expelled node must sync with a current member to rejoin the cluster.

In case the Coordinator fails at any point, a new Coordinator will be elected out of the remaining available Participants. Stardog Cluster supports both `read` (e.g., querying) and `write` (e.g., adding data) requests. All read and write requests can be handled by any of the nodes in the cluster. When a client commits a transaction (containing a list of `write` requests), it will be acknowledged by the receiving node only **after every non-failing peer node has committed the transaction**. If a peer node fails during the process of committing a transaction, it will be expelled from the cluster by the Coordinator and put in a temporary `failed` state. If the Coordinator fails during the process, the transaction will be aborted. At that point the client can retry the transaction and it should succeed with the new cluster coordinator.

Since `failed` nodes are not used for any subsequent `read` or `write` requests, **if a commit is acknowledged, then Stardog Cluster guarantees that the data has been accordingly modified at every available node in the cluster**.

While this approach is less performant with respect to write operations than eventual consistency used by other distributed databases, typically those databases offer a much less expressive data model than Stardog, which makes an eventually consistency model more appropriate for those systems (and less so for Stardog). But since Stardog's data model is not only richly expressive but rests in part on provably correct semantics, we think that a strong consistency model is worth the cost.**[21 (#_footnote_21)]**

Single Server Migration

It is assumed that Stardog nodes in a Stardog Cluster are always going to be used within a cluster context. Therefore, if you want to migrate from a Stardog instance running in single server mode to running in a cluster, it is advised that you create backups of your current databases and then import them to the cluster in order to be able to provide the guarantees explained above. If you simply add a Stardog instance to cluster that was previously running in single server mode, *it will sync to the state of the cluster*; local data could be removed when syncing with the cluster state.

Configuration

In this section we will explain how to manually deploy a Stardog Cluster using `stardog-admin` commands and some additional configuration. If you are deploying your cluster to AWS then you can use the Stardog Graviton (#_stardog_graviton) that will automate this process.

You can use the `stardog-admin cluster generate` command to bootstrap a cluster configuration and, thus, to ease installation by simply passing a list of hostnames or IP addresses for the cluster's nodes.

```
$ stardog-admin cluster generate --output-dir /home/stardog 10.0.0.1 10.0.0.2 10.0.0.3
```

See the man page (/man/cluster-generate.html) for the details.

In a production environment we strongly recommend that each ZooKeeper process runs in a different machine and, if possible, that ZooKeeper has a separate drive for its data directory. If you need a larger cluster, adjust accordingly.

In the following example we will set up a cluster with total of 6 nodes. Zookeeper will be deployed on nodes 1-3 whereas Stardog will be deployed on nodes 4-6.

1. Install (#_quick_start_guide) Stardog 5.3.6 on each machine in the cluster.

   NOTE The best thing to do here, of course, is to use whatever infrastructure you have in place to automate software installation. Adapting Stardog installation to Chef, Puppet, cfengine, etc. is left as an exercise for the reader.

2. Make sure a valid Stardog license key (whether Developer, Enterprise, or a 30-day eval key) for the size of cluster you're creating exists and resides in `STARDOG_HOME` on each node. You must also have a `stardog.properties` file with the following information **for each Stardog node in the cluster**:

   ```
   # Flag to enable the cluster, without this flag set, the rest of the properties have no effect
   pack.enabled=true
   # this node's IP address (or hostname) where other Stardog nodes are going to connect
   # this value is optional but if provided it should be unique for each Stardog node
   pack.node.address=196.69.68.4
   # the connection string for ZooKeeper where cluster state is stored
   pack.zookeeper.address=196.69.68.1:2180,196.69.68.2:2180,196.69.68.3:2180
   ```

   `pack.zookeeper.address` is a ZooKeeper connection string where cluster stores its state. `pack.node.address` is not a required property. The local address of the node, by default, is `InetAddress.getLocalhost().getAddress()`, which should work for many deployments. However if you're using an atypical network topology and the default value is not correct, you can provide a value for this property.

3. Create the ZooKeeper configuration for each ZooKeeper node. This config file is just a standard ZooKeeper configuration file and the same config file can be used for all ZooKeeper nodes. The following config file should be sufficient for most cases.

   ```
   tickTime=3000
   # Make sure this directory exists and
   # ZK can write and read to and from it.
   dataDir=/data/zookeeperdata/
   clientPort=2180
   initLimit=5
   syncLimit=2
   # This is an enumeration of all Zk nodes in
   # the cluster and must be identical in
   # each node's config.
   server.1=196.69.68.1:2888:3888
   server.2=196.69.68.2:2888:3888
   server.3=196.69.68.3:2888:3888
   ```

   NOTE The `clientPort` specified in `zookeeper.properties` and the ports used in `pack.cluster.address` in `stardog.properties` must be the same.

4. `dataDir` is where ZooKeeper persists cluster state and where it writes log information about the cluster.

   ```
   $ mkdir /data/zookeeperdata # on node 1
   $ mkdir /data/zookeeperdata # on node 2
   $ mkdir /data/zookeeperdata # on node 3
   ```

5. ZooKeeper requires a `myid` file in the `dataDir` folder to identify itself, you will create that file as follows for `node1`, `node2`, and `node3`, respectively:

   ```
   $ echo 1 > /data/zookeeperdata/myid # on node 1
   $ echo 2 > /data/zookeeperdata/myid # on node 2
   $ echo 3 > /data/zookeeperdata/myid # on node 3
   ```

Installation

In the next few steps you will use the Stardog Admin CLI commands to deploy Stardog Cluster: that is, ZooKeeper and Stardog itself. We'll also configure HAProxy as an example of how to use Stardog Cluster behind a proxy for load-balancing and fail-over capability. There's nothing special about HAProxy here; you could implement this proxy functionality in many different ways. For example, Stardog Graviton (#_stardog_graviton) uses Amazon's Elastic Load Balancer.

1. **Start ZooKeeper instances**

   First, you need to start ZooKeeper nodes. You can do this using the standard command line tools that come with ZooKeeper. As a convenience, we provide a `stardog-admin cluster zkstart` subcommand that you can use to start ZooKeeper instances:

   ```
   $ ./stardog-admin cluster zkstart --home ~/stardog # on node 1
   $ ./stardog-admin cluster zkstart --home ~/stardog # on node 2
   $ ./stardog-admin cluster zkstart --home ~/stardog # on node 3
   ```

   This uses the `zookeeper.properties` config file in `~/stardog` and log its output to `~/stardog/zookeeper.log`. If your `$STARDOG_HOME` is set to `~/stardog`, then you don't need to specify the `--home` option. For more info about the command:

   ```
   $ ./stardog-admin help cluster zkstart
   ```

2. **Start Stardog instances**

   Once ZooKeeper is started, you can start Stardog instances:

   ```
   $ ./stardog-admin server start --home ~/stardog --port 5821 # on node 4
   $ ./stardog-admin server start --home ~/stardog --port 5821 # on node 5
   $ ./stardog-admin server start --home ~/stardog --port 5821 # on node 6
   ```

   Important: When starting Stardog instances for the cluster, unlike single server mode, you need to provide the credentials of a superuser that will be used for securing the data stored in ZooKeeper and for intra-cluster communication. Each node should be started with the same superuser credentials. By default, Stardog comes with a superuser `admin` that has password `"admin"` and that is the default credentials used by the above command. For a secure installation of Stardog cluster you should change these credentials by specifying the `pack.zookeeper.auth` setting in stardog.properties and restart the cluster with new credentials:

   ```
   pack.zookeeper.auth=username:password
   ```

   And again, if your `$STARDOG_HOME` is set to `~/stardog`, you don't need to specify the `--home` option.

   NOTE Make sure to allocate roughly twice as much heap for Stardog than you would normally do for single-server operation since there can be an additional overhead involved for replication in the cluster. Also, we start Stardog here on the non-default port (`5821`) so that you can use a proxy or load-balancer in the same machine which can run on the default port (`5820`), meaning that Stardog clients can act normally (i.e., use the default port, `5820`) since they need to interact with HAProxy.

3. **Start HAProxy (or equivalent)**

   In most Unix-like systems, HAProxy is available via package managers, e.g. in Debian-based systems:

   ```
   $ sudo apt-get update
   $ sudo apt-get install haproxy
   ```

   At the time of this writing, this will install HAProxy 1.4. You can refer to the official site (http://www.haproxy.org/) to install a later release.

   Place the following configuration in a file (such as `haproxy.cfg`) in order to point HAProxy to the Stardog Cluster. You'll notice that there are two backends specified in the config file: `stardog_coordinator` and `all_stardogs`. An ACL is used to route all requests containing `transaction` in the path to the coordinator. All other traffic is routed via the default backend, which is simply round-robin across all of the Stardog nodes. For some use cases routing transaction-specific operations (e.g. commit) directly to the coordinator performs slightly better. However, round-robin routing across all of the nodes is generally sufficient.

```
global
    daemon
    maxconn 256

defaults
    # you should update these values to something that makes
    # sense for your use case
    timeout connect 5s
    timeout client 1h
    timeout server 1h
    mode http

# where HAProxy will listen for connections
frontend stardog-in
    option tcpka # keep-alive
    bind *:5820
    # the following lines identify any routes with "transaction"
    # in the path and send them directly to the coordinator, if
    # haproxy is unable to determine the coordinator all requests
    # will fall through and be routed via the default_backend
    acl transaction_route path_sub -i transaction
    use_backend stardog_coordinator if transaction_route
    default_backend all_stardogs

# the Stardog coordinator
backend stardog_coordinator
    option tcpka
    # the following line returns 200 for the coordinator node
    # and 503 for non-coordinators so traffic is only sent
    # to the coordinator
    option httpchk GET /admin/cluster/coordinator
    # the check interval can be increased or decreased depending
    # on your requirements and use case, if it is imperative that
    # traffic be routed to the coordinator as quickly as possible
    # after the coordinator changes, you may wish to reduce this value
    default-server inter 5s
    # replace these IP addresses with the corresponding node address
    # maxconn value can be upgraded if you expect more concurrent
    # connections
    server stardog1 196.69.68.1:5821 maxconn 64 check
    server stardog2 196.69.68.2:5821 maxconn 64 check
    server stardog3 196.69.68.3:5821 maxconn 64 check

# the Stardog servers
backend all_stardogs
    option tcpka # keep-alive
    # the following line performs a health check
    # HAProxy will check that each node accepts connections and
    # that it's operational within the cluster. Health check
    # requires that Stardog nodes do not use --no-http option
    option httpchk GET /admin/healthcheck
    default-server inter 5s
    # replace these IP addresses with the corresponding node address
    # maxconn value can be upgraded if you expect more concurrent
    # connections
    server stardog1 196.69.68.1:5821 maxconn 64 check
    server stardog2 196.69.68.2:5821 maxconn 64 check
    server stardog3 196.69.68.3:5821 maxconn 64 check
```

If you wish to operate the cluster in HTTP-only mode, you can add the `mode http` to backend settings.

Finally,

```
$ haproxy -f haproxy.cfg
```

For more info on configuring HAProxy please refer to the official documentation (http://www.haproxy.org/#docs). In production environments we recommend running multiple proxies to avoid single point of failures, and use DNS solutions for fail-over.

Now Stardog Cluster is running on 3 nodes, one each on 3 machines. Since HAProxy was conveniently configured to use port `5820` you can execute standard Stardog CLI commands to the Cluster:

```
$ ./stardog-admin db create -n myDb
$ ./stardog data add myDb /path/to/my/data
$ ./stardog query myDb "select * { ?s ?p ?o } limit 5"
```

If your cluster is running on another machine then you will need to provide a fully qualified connection string (#_how_to_make_a_connection_string) in the above commands.

Shutdown

In order to shut down the cluster you only need to execute the following command once:

```
$ ./stardog-admin cluster stop
```

The `cluster stop` request will cause all available nodes in the cluster to shutdown. If a node was expelled from the cluster due to a failure it would not receive this command and might need to be shutdown manually. In order to shutdown a single node in the cluster use the regular `server stop` command and be sure to specify the server address:

```
$ ./stardog-admin --server http://localhost:5821 server stop
```

If you send the `server stop` command to the load balancer then **a random node selected by the load balancer will shutdown.**

In addition to the Stardog cluster you still need to shut down the ZooKeeper cluster independently. Refer to the ZooKeeper documentation for details.

Configuration Issues

Topologies & Size

In the configuration instructions above, we assume a particular Cluster topology, which is to say, for each node `n` of a cluster, we run Stardog, ZooKeeper, and a load balancer. But this is not the only topology supported by Stardog Cluster. ZooKeeper nodes run independently, so other topologies—three ZooKeeper servers and five Stardog servers are possible—you just have to point Stardog to the corresponding ZooKeeper ensemble.

To add more Stardog Cluster nodes, simply repeat the steps for Stardog on additional machines. Generally, as mentioned above, Stardog Cluster size should be an odd number greater or equal to 3.

WARNING ZooKeeper uses a very write heavy protocol; having Stardog and ZooKeeper both writing to the **same** disk can yield contention issues, resulting in timeouts at scale. We recommend at a minimum having the two services writing to separate disks to reduce contention or, ideally, have them run on separate nodes entirely.

Open File Limits

If you expect to use Stardog Cluster with heavy concurrent write workloads, then you should probably increase the number of open files that host OS will permit on each Cluster node. You can typically do this on a Linux machine with `ulimit -n` or some variant thereof. Because nodes communicate between themselves and with ZooKeeper, it's important to make sure that there are sufficient file handle resources available.[22 (#_footnote_22)]

Connection/Session Timeouts

Stardog nodes connect to the ZooKeeper cluster and establishes a session (https://zookeeper.apache.org/doc/trunk/zookeeperProgrammers.html#ch_zkSessions). The session is kept alive by PING requests sent by the client. If the Stardog node does not send these requests to the ZooKeeper server (due to network issues, node failure, etc.) the session will timeout and the Stardog node will get into a suspended state and it will reject any queries or transactions until it can establish the session again.

If a Stardog node is overloaded then it might fail to send the PING requests to ZooKeeeper server in a timely manner. This usually happens when Stardog's memory usage is close to the limit and there are frequent GC pauses. This would cause Stardog nodes to be suspended unnecessarily. In order to prevent this problem make sure Stardog nodes have enough memory allocated and tweak the timeout options.

There are two different configuration options that control timeouts for the ZooKeeper server. The `pack.connection.timeout` option specifies the max time that Stardog waits to establish a connection to ZooKeeper. The `pack.session.timeout` option specifies the session timeout explained above. You can set these values in `stardog.properties` as follows:

```
pack.connection.timeout=15s
pack.session.timeout=60s
```

Note that, ZooKeeper has limitations about how these values can be set based on the `tickTime` value specified in the ZooKeeper configuration file. Session timeout needs to be a minimum of 2 times the `tickTime` and a maximum of 20 times the `tickTime`. So a session timeout of `60s` requires the `tickTime` to be at least `3s` (in ZooKepeer configuration file this value should be entered in milliseconds). If the session timeout is not in the allowed range the ZooKeeper will negotiate a new timeout value and Stardog will print a warning about this in the `stardog.log` file.

Client Usage

To use Stardog Cluster with standard Stardog clients and CLI tools in the ordinary way--`stardog-admin` and `stardog`--you must have Stardog installed locally. With the provided Stardog binaries in the Stardog Cluster distribution you can query the state of Cluster:

```
$ ./stardog-admin --server http://<ipaddress>:5820/ cluster info
```

where `ipaddress` is the IP address of any of the nodes in the cluster. This will print the available nodes in the cluster, as well as the roles (participant or coordinator). You can also input the proxy IP address and port to get the same information.

To add or remove data, issue `stardog data add` or `remove` commands to any node in the cluster. Queries can be issued to any node in the cluster using the `stardog query` command. All the `stardog-admin` features are also available in Cluster, which means you can use any of the commands to create databases, administer users, and the rest of the functionality.

Adding Nodes to a Cluster

Stardog cluster stores the UUID of the last committed transaction for each database in ZooKeeper. When a new node is joining the cluster it will compare the local transaction ID of each database with the corresponding transaction ID stored in ZooKeeper. If there is a mismatch the node will synchronize the database contents from another node in the cluster. If there are no nodes in the cluster the new node cannot join the cluster and will shut itself down. For this reason, if you are starting a new cluster then you should make sure that the ZooKeeper state is cleared. If you are retaining an existing cluster then new nodes should be started when there is at least one node in the cluster.

If there are active transactions in the cluster joining node will wait for those transactions to finish and then synchronize its databases. More transactions may take place during synchronization and in that case the joining node will continue synchronization and retrieve the data from new transactions. Thus, it will take longer for a node to join the cluster if there are continuous transactions. Note that, the new node will not be available for requests until all the databases are synchronized. The proxy/load-balancer should perform a health check before forwarding the requests to a new node (as shown in the above configuration) so user requests will always be forwarded to available nodes.

Upgrading the Cluster

The process to upgrade Stardog Cluster is straightfoward; however, there are a few extra steps you should take to ensure the upgrade goes as quickly and smoothly as possible. Before you begin the upgrade, make sure to place the new Stardog binaries on all of the cluster nodes.

Also make sure to note which node is the coordinator since this is the first node that will be started as part of the upgrade. `stardog-admin cluster info` will show the nodes in the cluster and which one is the coordinator.

Next you should ensure that there are no transactions running, e.g., `stardog-admin db status <db name>` will show if there are any open transactions for a database. This step is not strictly required, however, it can minimize downtime and streamline the process, allowing the cluster to stop quickly and helping avoid non-coordinator nodes from having to re-sync when they attempt to join the upgraded cluster.

When you are ready to begin the upgrade, you can shutdown the cluster with `stardog-admin cluster stop`. Once all nodes have stopped, backup the `STARDOG_HOME` directories on all of the nodes.

With the new version of Stardog, bring the cluster up one node at a time, starting with the previous coordinator. As each node starts make sure that it is able to join the cluster cleanly before moving on to the next node.

SEARCH

Stardog's builtin full-text search system indexes data stored in Stardog for information retrieval queries.

Indexing Strategy

The indexing strategy creates a "search document" per RDF literal. Each document consists of two fields: literal ID and literal value. See User-defined Lucene Analyzer (#_user_defined_lucene_analyzer) for details on customizing Stardog's search programmatically.

Enabling Search

Full-text support for a database is disabled by default but can be enabled at any time by setting the configuration option `search.enabled` to true. For example, you can create a database with full-text support as follows:

```
$ stardog-admin db create -o search.enabled=true -n myDb
```

Similarly, you can set the option using `SearchOptions#SEARCHABLE` when creating the database programmatically:

```
aAdminConnection.disk("myDB")
                .set(SearchOptions.SEARCHABLE, true)
                .create()
```

Integration with SPARQL

We use the predicate `tag:stardog:api:property:textMatch` (or `http://jena.hpl.hp.com/ARQ/property#textMatch`) to access the search index in a SPARQL query.

The `textMatch` function has one required argument, the search query in Lucene syntax (http://lucene.apache.org/core/5_3_0/queryparser/org/apache/lucene/queryparser/classic/package-summary.html#package_description) and it returns, by default, all literals matching the query string. For example,

```
SELECT DISTINCT ?s ?score
WHERE {
?s ?p ?l.
(?l ?score) <tag:stardog:api:property:textMatch> 'mac'.
}
```

This query selects all literals which match 'mac'. These literals are then joined with the generic BGP `?s ?p ?l` to get the resources (`?s`) that have those literals. Alternatively, you could use `?s rdf:type ex:Book` if you only wanted to select the books which reference the search criteria; you can include as many other BGPs as you like to enhance your initial search results.

You can change the number of results `textMatch` returns by providing an optional second argument with the limit:

```
SELECT DISTINCT ?s ?score
WHERE {
?s ?p ?l.
(?l ?score) <tag:stardog:api:property:textMatch> ('mac' 100).
}
```

Limit in `textMatch` only limits the number of literals returned, which is different than the number of total results the query will return. When a `LIMIT` is specified in the SPARQL query, it does not affect the full-text search, rather, it only restricts the size of the result set.

Lucene returns a score (https://lucene.apache.org/core/5_3_0/core/org/apache/lucene/search/package-summary.html#scoring) with each match. It is possible to return these scores and define filters based on the score:

```
SELECT DISTINCT ?s ?score
WHERE {
?s ?p ?l.
(?l ?score) <tag:stardog:api:property:textMatch> ('mac' 0.5 10).
}
```

This query returns 10 matching literals where the score is **greater than** 0.5. Note that, as explained in the Lucene documentation (https://lucene.apache.org/core/5_3_0/core/org/apache/lucene/search/package-summary.html#scoring) scoring is very much dependent on the way documents are indexed and the range of scores might change significantly between different databases.

Escaping Characters in Search

The "/" character must be escaped because Lucene says so. In fact, there are several characters that are part of Lucene's query syntax that must be escaped (http://lucene.apache.org/core/5_3_0/queryparser/org/apache/lucene/queryparser/classic/package-summary.html#Escaping_Special_Characters).

Search Syntax

Stardog search is based on Lucene 5.3.0: we support all of the search modifiers (http://lucene.apache.org/core/5_3_0/queryparser/org/apache/lucene/queryparser/classic/package-summary.html#package_description) that Lucene supports, with the exception of fields.

- wildcards: `?` and `*`
- fuzzy: `~` and `~` with similarity weights (e.g. `foo~0.8`)
- proximities: `"semantic web"~5`
- term boosting
- booleans: `OR`, `AND`, `NOT`, +`, and `-`.
- grouping

For a more detailed discussion, see the Lucene docs (http://lucene.apache.org/core/5_3_0/queryparser/org/apache/lucene/queryparser/classic/package-summary.html#package_description).

OWL & RULE REASONING

In this chapter we describe how to use Stardog's reasoning capabilities; we address some common problems and known issues. We also describe Stardog's approach to query answering with reasoning in some detail, as well as a set of guidelines that contribute to efficient query answering with reasoning. If you are not familiar with the terminology, you can peruse the section on terminology.

The semantics of Stardog's reasoning is based in part on the OWL 2 Direct Semantics Entailment Regime (https://www.w3.org/TR/sparql11-entailment/#OWLRDFBSEntRegime). However, the implementation of Stardog's reasoning system is worth understanding as well. For the most part, **Stardog performs reasoning in a lazy and late-binding fashion: it does not materialize inferences; but, rather, reasoning is performed at query time according to a user-specified "reasoning type".** This approach allows for maximum flexibility[23 (#_footnote_23)] while maintaining excellent performance. The one exception to this general approach is equality reasoning which is eagerly materialized. See Same As Reasoning (#_same_as_reasoning) for more details.

TIP	How to Debug Reasoning (https://www.stardog.com/blog/how-to-debug-reasoning/)

Reasoning Types

Reasoning can be enabled or disabled using a simple boolean flag—in HTTP, `reasoning`; in CLI, `-r` or `--reasoning`; in the Web Console, a reasoning button in the query panel; and in Java APIs, a connection option (/docs/5.3.6/java/snarl/com/complexible/stardog/api/connectionconfiguration#reasoning(boolean)) or a query option (/docs/5.3.6/java/snarl/com/complexible/stardog/api/query#reasoning(boolean)):

`false`	No axioms or rules are considered; no reasoning is performed.
`true`	Axioms and rules are considered and reasoning is performed according to the value of the `reasoning.type` database option.

Reasoning is disabled by default; that is, no reasoning is performed without explicitly setting the reasoning flag to "true".

When reasoning is enabled by the boolean flag, the axioms and rules in the database are first filtered according to the value of the `reasoning.type` database option. The default value of `reasoning.type` is `SL` and for the most part users don't need to worry too much about which reasoning type is necessary since `SL` covers all of the OWL 2 profiles as well as user-defined rules via SWRL. However, this value may be set to any other reasoning type that Stardog supports: `RDFS` is the OWL 2 axioms allowed in RDF Schema (http://www.w3.org/TR/rdf-schema/) (mainly subclasses, subproperties, domain, and ranges); `QL` for the OWL 2 QL (http://www.w3.org/TR/owl2-profiles/#OWL_2_QL) axioms; `RL` for the OWL 2 RL (http://www.w3.org/TR/owl2-profiles/#OWL_2_RL) axioms; `EL` for the OWL 2 EL (http://www.w3.org/TR/owl2-profiles/#OWL_2_EL) axioms; `DL` for OWL 2 DL axioms (http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/); and `SL` for a combination of RDFS, QL, RL, and EL axioms, plus SWRL rules (http://www.w3.org/Submission/SWRL/). Any axiom outside the selected type will be ignored by the reasoner.

The `DL` reasoning type behaves significantly different than other types. Stardog normally uses the Query Rewriting (#_query_rewriting) technique for reasoning which scales very well with increasing number of instances; only the schema needs to be kept in memory. But query rewriting cannot handle axioms outside the OWL 2 profiles; however, `DL` reasoning type can be used so that no axiom or rule is ignored as long as they satisfy the OWL 2 DL restrictions (http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/#Global_Restrictions_on_Axioms_in_OWL_2_DL). With `DL` reasoning, both the schema and the instance data need to pulled into memory, which limits its applicability with large number of instances. `DL` reasoning also requires the database to be logically consistent or no reasoning can be performed. Finally, `DL` reasoning requires more computation upfront compared to query rewriting which exhibits a "pay-as-you-go" behavior.

The `reasoning.type` can also be set to the special value `NONE` which will filter all axioms and rules thus effectively disables reasoning. This value can be used for the database option to prevent reasoning to be used by any client even though they might enable it with the boolean flag on the client side.

Using Reasoning

In order to perform query evaluation with reasoning, Stardog requires a schema[24 (#_footnote_24)] to be present in the database. Since schemas are serialized as RDF, they are loaded into a Stardog database in the same way that any RDF is loaded into a Stardog database. Also, note that, since the schema is just more RDF triples, it may change as needed: it is neither fixed nor compiled in any special way.

The schema may reside in the default graph, in a specific named graph, or in a collection of graphs. **You can tell Stardog where the schema is by setting the `reasoning.schema.graphs` property to one or more named graph URIs.** If you want the default graph to be considered part of the schema, then you can use the special built-in URI `tag:stardog:api:context:default`. If you want to use all named graphs (that is, to tell Stardog to look for the schema in every named graph), you can use `tag:stardog:api:context:all`.

NOTE	**The default value for this property is to use all graphs, i.e.,** `tag:stardog:api:context:all`.

This design is intended to support both of Stardog's primary use cases:

1. managing the data that constitutes the schema
2. reasoning with the schema during query evaluation

Query Answering

All of Stardog's interfaces (API, network, and CLI) support reasoning during query evaluation.

Command Line

In order to evaluate queries in Stardog using reasoning via the command line, we use the reasoning flag:

```
$ ./stardog query --reasoning myDB "SELECT ?s { ?s a :C } LIMIT 10"
```

HTTP

For HTTP, the reasoning flag is specified with the other HTTP request parameters:

```
$ curl -u admin:admin -X GET "http://localhost:5822/myDB/query?reasoning=true&query=..."
```

Reasoning Connection API

In order to use the `ReasoningConnection` API one needs to enable reasoning. See the Java Programming (#_java_programming) section for details.

Currently, the API has two methods:

- `isConsistent()`, which can be used to check if the database is (logically) consistent with respect to the reasoning type.
- `isSatisfiable(URI theURIClass)`, which can be used to check if the given class if satisfiable with respect to the database and reasoning type.

Explaining Reasoning Results

Stardog can be used to check if the current database logically entails a set of triples; moreover, Stardog can explain why this is so.[25 (#_footnote_25)] An explanation of an inference is the minimum set of statements explicitly stored in the database that, together with the schema and any valid inferences, logically justify the inference. Explanations are useful for understanding data, schema, and their interactions, especially when large number of statements interact with each other to infer new statements.

Explanations can be retrieved using the CLI by providing an input file that contains the inferences to be explained:

```
$ stardog reasoning explain myDB inference_to_explain.ttl
```

The output is displayed in a concise syntax designed to be legible; but it can be rendered in any one of the supported RDF syntaxes if desired. Explanations are also accessible through Stardog's extended HTTP protocol (#_network_programming) and in Java (#_java_programming). See the examples in the stardog-examples Github repo (https://github.com/Complexible/stardog-examples/) for more details about retrieving explanations programmatically.

Proof Trees

Proof trees are a hierarchical presentation of multiple explanations (of inferences) to make data, schemas, and rules more intelligible. Proof trees[26 (#_footnote_26)] provide an explanation for an inference or an inconsistency as a **hierarchical structure**. Nodes in the proof tree may represent an assertion in a Stardog database. Multiple assertion nodes are grouped under an inferred node.

Example

For example, if we are explaining the inferred triple `:Alice rdf:type :Employee`, the root of the proof tree will show that inference:

```
INFERRED :Alice rdf:type :Employee
```

The children of an inferred node will provide more explanation for that inference:

```
INFERRED :Alice rdf:type :Employee
    ASSERTED :Manager rdfs:subClassOf :Employee
    INFERRED :Alice rdf:type :Manager
```

The fully expanded proof tree will show the asserted triples and axioms for every inference:

```
INFERRED :Alice rdf:type :Employee
    ASSERTED :Manager rdfs:subClassOf :Employee
    INFERRED :Alice rdf:type :Manager
        ASSERTED :Alice :supervises :Bob
        ASSERTED :supervises rdfs:domain :Manager
```

The CLI explanation command prints the proof tree using indented text; but, using the SNARL API, it is easy to create a tree widget in a GUI to show the explanation tree, such that users can expand and collapse details in the explanation.

Another feature of proof trees is the ability to merge multiple explanations into a single proof tree with multiple branches when explanations have common statements. Consider the following example database:

```
#schema
:Manager rdfs:subClassOf :Employee
:ProjectManager rdfs:subClassOf :Manager
:ProjectManager owl:equivalentClass (:manages some :Project)
:supervises rdfs:domain :Manager
:ResearchProject rdfs:subClassOf :Project
:projectID rdfs:domain :Project

#instance data
:Alice :supervises :Bob
:Alice :manages :ProjectX
:ProjectX a :ResearchProject
:ProjectX :projectID "123-45-6789"
```

In this database, there are three different unique explanations for the inference `:Alice rdf:type :Employee`:

Explanation 1

```
:Manager rdfs:subClassOf :Employee
:ProjectManager rdfs:subClassOf :Manager
:supervises rdfs:domain :Manager
:Alice :supervises :Bob
```

Explanation 2

```
:Manager rdfs:subClassOf :Employee
:ProjectManager rdfs:subClassOf :Manager
:ProjectManager owl:equivalentClass (:manages some :Project)
:ResearchProject rdfs:subClassOf :Project
:Alice :manages :ProjectX
:ProjectX a :ResearchProject
```

Explanation 3

```
:Manager rdfs:subClassOf :Employee
:ProjectManager rdfs:subClassOf :Manager
:ProjectManager owl:equivalentClass (:manages some :Project)
:projectID rdfs:domain :Project
:Alice :manages :ProjectX
:ProjectX :projectID "123-45-6789"
```

All three explanations have some triples in common; but when explanations are retrieved separately, it is hard to see how these explanations are related. When explanations are merged, we get a single proof tree where alternatives for subtrees of the proof are shown inline. In indented text rendering, the merged tree for the above explanations would look as follows:

```
INFERRED :Alice a :Employee
   ASSERTED :Manager rdfs:subClassOf :Employee
   1.1) INFERRED :Alice a :Manager
      ASSERTED :supervises rdfs:domain :Manager
      ASSERTED :Alice :supervises :Bob
   1.2) INFERRED :Alice a :Manager
      ASSERTED :ProjectManager rdfs:subClassOf :Manager
      INFERRED :Alice a :ProjectManager
         ASSERTED :ProjectManager owl:equivalentClass (:manages some :Project)
         ASSERTED :Alice :manages :ProjectX
         2.1) INFERRED :ProjectX a :Project
            ASSERTED :projectID rdfs:domain :Project
            ASSERTED :ProjectX :projectID "123-45-6789"
         2.2) INFERRED :ProjectX a :Project
            ASSERTED :ResearchProject rdfs:subClassOf :Project
            ASSERTED :ProjectX a :ResearchProject
```

In the merged proof tree, alternatives for an explanation are shown with a number id. In the above tree, `:Alice a :Manager` is the first inference for which we have multiple explanations so it gets the id `1`. Then each alternative explanation gets an id appended to this (so explanations `1.1` and `1.2` are both alternative explanations for inference `1`). We also have multiple explanations for inference `:ProjectX a :Project` so its alternatives get ids `2.1` and `2.2`.

User-defined Rule Reasoning

Many reasoning problems may be solved with OWL's axiom-based approach; but, of course, not all reasoning problems are amenable to this approach. A user-defined rules approach complements the OWL axiom-based approach nicely and increases the expressive power of a reasoning system from the user's point of view. Many RDF databases support user-defined rules only. Stardog is the only RDF database that comprehensively supports both axioms and rules. Some problems (and some people) are simply a better fit for a rules-based approach to modeling and reasoning than to an axioms-based approach (and, of course, *vice versa*).

NOTE There isn't a one-size-fits-all answer to the question "rules or axioms or both?" Use the thing that makes the most sense given the task at hand. This is engineering, not religion.

Stardog supports user-defined rule reasoning together with a rich set of built-in functions using the SWRL (http://www.w3.org/Submission/SWRL/) syntax and builtins library. In order to apply SWRL user-defined rules, you must include the rules as part of the database's schema: that is, put your rules where your axioms are, i.e., in the schema. Once the rules are part of the schema, they will be used for reasoning automatically when using the **SL** reasoning type.

Assertions implied by the rules **will not** be materialized. Instead, rules are used to expand queries just as regular axioms are used.

NOTE To trigger rules to fire, execute a relevant query—simple and easy as the truth.

Stardog Rules Syntax

Stardog supports two different syntaxes for defining rules. The first is native Stardog Rules syntax and is based on SPARQL, so you can re-use what you already know about SPARQL to write rules. **Unless you have specific requirements otherwise, you should use this syntax for user-defined rules in Stardog.** The second is the **de facto** standard RDF/XML syntax for SWRL. It has the advantage of being supported in many tools; but it's not fun to read or to write. You probably don't want to use it. Better: don't use this syntax!

Stardog Rules Syntax is basically SPARQL "basic graph patterns" (BGPs) plus some very explicit new bits (`IF-THEN`) to denote the head and the body of a rule.[27 (#_footnote_27)] You define URI prefixes in the normal way (examples below) and use regular SPARQL variables for rule variables. As you can see, some SPARQL 1.1 syntactic sugar—property paths, especially, but also bnode syntax—make complex Stardog Rules concise and elegant.

NOTE Starting in Stardog 3.0, it's legal to use any valid Stardog function in Stardog Rules (see rule limitations below for few exceptions).

How to Use Stardog Rules

There are three things to sort out:

1. Where to put these rules?
2. How to represent these rules?
3. What are the gotchas?

First, the rules go into the database, of course. Unless you've changed the value of `reasoning.schema.graphs` option, you can store the rules in any named graph (or the default graph) in the database and you will be fine; that is, just add the rules to the database and it will all work out.[28 (#_footnote_28)]

Second, you include the rules directly in a Turtle file loaded into Stardog. Rules can be mixed with triples in the file. Here's an example:

```
:c a :Circle ;
   :radius 10 .

IF {
        ?r a :Rectangle ;
           :width ?w ;
           :height ?h
    BIND (?w * ?h AS ?area)
}
THEN {
    ?r :area ?area
}
```

That's pretty easy. Third, what are the gotchas?

Rule Representation Options

Inline rules in Turtle data can be named for later reference and management. We assign an IRI, `:FatherRule` in this example, to the rule and use it as the subject of other triples:

```
@prefix : <http://example.org/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

RULE :FatherRule
IF {
   ?x a <http://example.org/Male> , <http://example.org/Parent> .
}
THEN {
   ?x a <http://example.org/Father> .
}

:FatherRule rdfs:comment "This rule defines fathers" ;
        a :MyRule .
```

In addition to the inline Turtle representation of rules, you can represent the rules with specially constructed RDF triples. This is useful for maintaining Turtle compatibility or for use with SPARQL `INSERT DATA`. This example shows the object of a triple which contains **one** rule in Stardog Rules syntax embedded as literal.

```
@prefix rule: <tag:stardog:api:rule:> .

[] a rule:SPARQLRule;
   rule:content """
      IF {
        ?r a :Rectangle ;
           :width ?w ;
           :height ?h
          BIND (?w * ?h AS ?area)
      }
      THEN {
          ?r :area ?area
      }
   """.
```

Rule Limitations & Gotchas

1. The RDF serialization of rules in, say, a Turtle file has to use the `tag:stardog:api:rule:` namespace URI and then whatever prefix, if any, mechanism that's valid for that serialization. In the examples here, we use Turtle. Hence, we use `@prefix`, etc.

2. However, the namespace URIs used by the literal embedded rules can be defined in two places: the string that contains the rule—in the example above, you can see the default namespace is `urn:test:` --or in the Stardog database in which the rules are stored. Either place will work; if there are conflicts, the "closest definition wins", that is, if `foo:Example` is defined in both the rule content and in the Stardog database, the definition in the rule content is the one that Stardog will use.

3. Stardog Rule Syntax has the same expressivity of SWRL (https://www.w3.org/Submission/SWRL/) which means the SPARQL features allowed in rules are limited. Specifically, a triple pattern in a rule should be in one of the following forms:

   a) **term$_1$** `rdf:type` **class-uri**

   b) **term$_1$ prop-uri term$_2$**

   where **class-uri** is a URI referring to a user-defined class and **prop-uri** is a URI referring to a user-defined property.[29 (#_footnote_29)]

   Only type of property paths allowed in rules are inverse paths (`^p`), sequence paths (`p1 / p2`) and alternative paths (`p1 | p2`) but these paths should not violate the above conditions. For example, the property path `rdf:type/rdfs:label` is not valid because according to the SPARQL spec (https://www.w3.org/TR/sparql11-query/#propertypath-syntaxforms) this would mean the object of a `rdf:type` triple pattern is a variable and not a user-defined class.

   Rule body (`IF`) and only rule body may optionally contain `UNION`, `BIND` or `FILTER` clauses. However, functions `EXISTS`, `NOT EXISTS`, or `NOW()` cannot be used in rules. User-defined functions (UDF) may be used in rules but if the UDF is not a pure function (https://en.wikipedia.org/wiki/Pure_function) then the results are undefined.

   Other SPARQL features are not allowed in rules.

4. Having the same predicate both in the rule body (`IF`) and the rule head (`THEN`) are supported in a limited way. Cycles are allowed only if the rule body does not contain type triples or filters and the triples in the rule body are linear (i.e. no cycles in the rule body either).

   In other words, a property used in the rule head depends on a property in the rule body and this dependency graph may contain cycles under some limits. One of these is that a rule body should not contain type triples or filters. Tree-like dependencies are always allowed.

   Of course the rule body may also contain triple patterns, which constitute a different kind of graph: it should be linear when edge directions are ignored. So no cycles or trees are allowed in this graph pattern. Linear when directions are ignored means that `{ ?x :p ?y . ?x :p ?z }` is linear but `{ ?x :p ?y . ?x :p ?z . ?x :p ?t }` is not because there are three edges for the node represented by `?x`.

   The reason for these limits boils down to the fact that recursive rules and axioms are rewritten as SPARQL property paths. This is why rule bodies cannot contain anything but property atoms. Cycles are allowed as long as we can express these as a regular grammar. Another way to think about this is that these rules should be as expressive as OWL property chains and the same restrictions defined for property chains apply here, too.

   Let's consider some examples.

   These rules are acceptable since no cycles appear in dependencies:

   ```
   IF
     { ?x :hasFather ?y . ?y :hasBrother ?z }
   THEN
     { ?x :hasUncle ?z }
   ```

   ```
   IF
     { ?x :hasUncle ?y . ?y :hasWife ?z }
   THEN
     { ?x :hasAuntInLaw ?z }
   ```

   These rules are not acceptable since there is a cycle:

   ```
   IF
     { ?x :hasFather ?y . ?y :hasBrother ?z }
   THEN
     { ?x :hasUncle ?z }
   ```

```
IF
  { ?x :hasChild ?y . ?y :hasUncle ?z }
THEN
  { ?x :hasBrother ?z }
```

This kind of cycle is allowed:

```
IF
  { ?x :hasChild ?y . ?y :hasSibling ?z }
THEN
  { ?x :hasChild ?z }
```

NOTE (3) is a **general** limitation, not specific to Stardog Rules Syntax: recursion or cycles can occur through multiple rules, or it may occur as a result of interaction of rules with other axioms (or just through axioms alone).

Stardog Rules Examples

```
PREFIX rule: <tag:stardog:api:rule:>
PREFIX : <urn:test:>
PREFIX gr: <http://purl.org/goodrelations/v1#>

:Product1 gr:hasPriceSpecification [ gr:hasCurrencyValue 100.0 ] .
:Product2 gr:hasPriceSpecification [ gr:hasCurrencyValue 500.0 ] .
:Product3 gr:hasPriceSpecification [ gr:hasCurrencyValue 2000.0 ] .

IF {
   ?offering gr:hasPriceSpecification ?ps .
   ?ps gr:hasCurrencyValue ?price .
   FILTER (?price >= 200.00).
}
THEN {
   ?offering a :ExpensiveProduct .
}
```

This example is self-contained: it contains some data (the `:Product…` triples) and a rule. It also demonstrates the use of SPARQL's `FILTER` to do numerical (and other) comparisons.

Here's a more complex example that includes four rules and, again, some data.

```
PREFIX rule: <tag:stardog:api:rule:>
PREFIX : <urn:test:>

:c a :Circle ;
   :radius 10 .

:t a :Triangle ;
   :base 4 ;
   :height 10 .

:r a :Rectangle ;
   :width 5 ;
   :height 8 .

:s a :Rectangle ;
   :width 10 ;
   :height 10 .

IF {
   ?r a :Rectangle ;
      :width ?w ;
      :height ?h
   BIND (?w * ?h AS ?area)
}
THEN {
    ?r :area ?area
}

IF {
   ?t a :Triangle ;
      :base ?b ;
      :height ?h
   BIND (?b * ?h / 2 AS ?area)
}
THEN {
    ?t :area ?area
}

IF {
     ?c a :Circle ;
        :radius ?r
     BIND (math:pi() * math:pow(?r, 2) AS ?area)
}
THEN {
    ?c :area ?area
}


IF {
     ?r a :Rectangle ;
        :width ?w ;
        :height ?h
     FILTER (?w = ?h)
}
THEN {
    ?r a :Square
}
```

This example also demonstrates how to use SPARQL's `BIND` to introduce intermediate variables and do calculations with or to them.

Let's look at some other rules, but just the rule content this time for concision, to see some use of other SPARQL features.

This rule says that a person between 13 and 19 (inclusive) years of age is a teenager:

```
PREFIX swrlb: <http://www.w3.org/2003/11/swrlb#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>

IF {
      ?x a :Person; hasAge ?age.
      FILTER (?age >= 13 && ?age <= 19)
}
THEN {
      ?x a :Teenager.
}
```

This rule says that a male person with a sibling who is the parent of a female is an "uncle with a niece":

```
IF {
      ?x a Person; a :Male; :hasSibling ?y;
      ?y :isParentOf ?z;
      ?z a :Female.
}
THEN {
      ?x a :UncleOfNiece.
}
```

We can use SPARQL 1.1 property paths (and bnodes for unnecessary variables (that is, ones that aren't used in the `THEN`)) to render this rule even more concisely:

```
IF {
      ?x a :Person, :Male; :hasSibling/:isParentOf [a :Female]
}
THEN {
      ?x a :UncleOfNiece.
}
```

Aside: that's pure awesome.

And of course a person who's male and has a niece or nephew is an uncle of his niece(s) and nephew(s):

```
IF {
     ?x a :Male; :isSiblingOf/:isParentOf ?z
}
THEN {
      ?x :isUncleOf ?z.
}
```

Next rule example: a super user can read all of the things!

```
IF {
      ?x a :SuperUser.
      ?y a :Resource.
      ?z a <http://www.w3.org/ns/sparql#UUID>.
}
THEN {
      ?z a :Role.
      ?x :hasRole ?z; :readPermission ?y.
}
```

Supported Built-Ins

Stardog supports a wide variety of functions from SPARQL, XPath, SWRL, and some native Stardog functions, too. All of them may be used in either Stardog Rules syntax or in SWRL syntax. The supported functions are enumerated here (#_sparql_query_functions).

Special Predicates

Stardog supports some builtin predicates with special meaning in order to make queries and rules easier to read and write. These special predicates are primarily syntactic sugar for more complex structures.

Direct/Strict Subclasses, Subproperties, & Direct Types

Besides the standard RDF(S) predicates `rdf:type`, `rdfs:subClassOf` and `rdfs:subPropertyOf`, Stardog supports the following special built-in predicates:

- `sp:directType`
- `sp:directSubClassOf`
- `sp:strictSubClassOf`
- `sp:directSubPropertyOf`
- `sp:strictSubPropertyOf`

Where the `sp` prefix binds to `tag:stardog:api:property:`. Stardog also recognizes `sesame:directType`, `sesame:directSubClassOf`, and `sesame:strictSubClassOf` predicates where the prefix `sesame` binds to `http://www.openrdf.org/schema/sesame# (http://www.openrdf.org/schema/sesame#)`.

We show what these each of these predicates means by relating them to an equivalent triple pattern; that is, you can just write the predicate rather than the (more unwieldy) triple pattern.

```
#c1 is a subclass of c2 but not equivalent to c2

:c1 sp:strictSubClassOf :c2      =>       :c1 rdfs:subClassOf :c2 .
                                          FILTER NOT EXISTS {
                                             :c1 owl:equivalentClass :c2 .
                                          }

#c1 is a strict subclass of c2 and there is no c3 between c1 and c2 in
#the strict subclass hierarchy

:c1 sp:directSubClassOf :c2      =>       :c1 sp:strictSubClassOf :c2 .
                                          FILTER NOT EXISTS {
                                             :c1 sp:strictSubClassOf :c3 .
                                             :c3 sp:strictSubClassOf :c2 .
                                          }

#ind is an instance of c1 but not an instance of any strict subclass of c1

:ind sp:directType :c1           =>       :ind rdf:type :c1 .
                                          FILTER NOT EXISTS {
                                             :ind rdf:type :c2 .
                                             :c2 sp:strictSubClassOf :c1 .
                                          }
```

The predicates `sp:directSubPropertyOf` and `sp:strictSubPropertyOf` are defined analogously.

New Individuals with SWRL

Stardog also supports a special predicate that extends the expressivity of SWRL rules. According to SWRL, you can't create new individuals (i.e., new instances of classes) in a SWRL rule.

NOTE Don't get hung up by the tech vocabulary here…"new individual" just means that you can't have a rule that creates a new instance of some RDF or OWL class as a result of the rule firing.

This restriction is well-motivated; without it, you can easily create rules that do not terminate, that is, never reach a fixed point. Stardog's user-defined rules weakens this restriction in some crucial aspects, subject to the following restrictions, conditions, and warnings.

WARNING This special predicate is basically a loaded gun with which you may shoot yourselves in the foot if you aren't very careful.

So despite the general restriction in SWRL, in Stardog we actually can create new individuals with a rule by using the function `UUID()` as follows:

```
IF {
    ?p a :Parent .
    BIND (UUID() AS ?parent) .
}
THEN {
    ?parent a :Person .
}
```

NOTE Alternatively, we can use the predicate `http://www.w3.org/ns/sparql#UUID (http://www.w3.org/ns/sparql#UUID)` as a unary SWRL built-in.

This rule will create a **random** URI for each instance of the class `:Parent` and also assert that each new instance is an instance of `:Person`--parents are people, too!

Remarks

1. The URIs for the generated individuals are meaningless in the sense that they should not be used in further queries; that is to say, these URIs are not guaranteed by Stardog to be stable.
2. Due to normalization, rules with more than one atom in the head are broken up into several rules.

Thus,

```
IF {
    ?person a :Person .
    BIND (UUID() AS ?parent) .
}
THEN {
    ?parent a :Parent ;
            a :Male .
}
```

will be normalized into two rules:

```
IF {
    ?person a :Person .
    BIND (UUID() AS ?parent) .
}
THEN {
    ?parent a :Parent .
}

IF {
    ?person a :Person .
    BIND (UUID() AS ?parent) .
}
THEN {
    ?parent a :Male .
}
```

As a consequence, instead of stating that the new individual is both an instance of `:Male` and `:Parent`, we would create two **different** new individuals and assert that one is male and the other is a parent. If you need to assert various things about the new individual, we recommend the use of extra rules or axioms. In the previous example, we can introduce a new class (`:Father`) and add the following rule to our schema:

```
IF {
    ?person a :Father .
}
THEN {
    ?parent a :Parent ;
            a :Male .
}
```

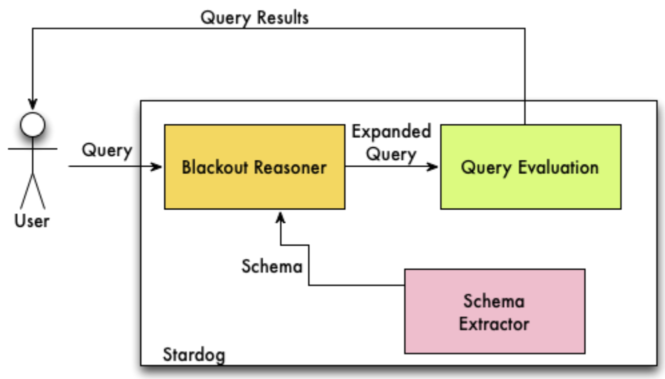
And then modify the original rule accordingly:

```
IF {
    ?person a :Person .
    BIND (UUID() AS ?parent) .
}
THEN {
    ?parent a :Father .
}
```

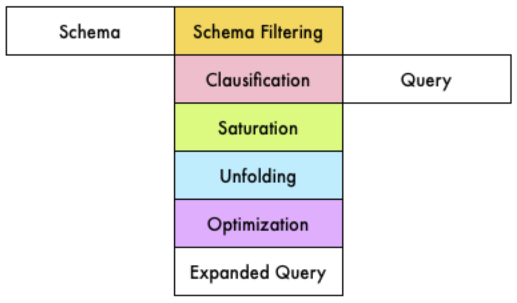
Query Rewriting

Reasoning in Stardog is based (mostly) on a **query rewriting** technique: Stardog rewrites the user's query with respect to any schema or rules, and then executes the resulting expanded query (EQ) against the data in the normal way. This process is completely automated and requires no intervention from the user.

As can be seen in Figure 1, the rewriting process involves five different phases.



1. *Figure 1 Query Answering*



2. *Figure 2. Query Rewriting*

We illustrate the query answering process by means of an example. Consider a Stardog database, MyDB$_1$, containing the following schema:

```
:SeniorManager rdfs:subClassOf :manages some :Manager
:manages some :Employee rdfs:subClassOf :Manager
:Manager rdfs:subClassOf :Employee
```

Which says that a senior manager manages at least one manager, that every person that manages an employee is a manager, and that every manager is also an employee.

Let's also assume that MyDB$_1$ contains the following data assertions:

```
:Bill rdf:type :SeniorManager
:Robert rdf:type :Manager
:Ana :manages :Lucy
:Lucy rdf:type :Employee
```

Finally, let's say that we want to retrieve the set of all employees. We do this by posing the following query:

```
SELECT ?employee WHERE { ?employee rdf:type :Employee }
```

To answer this query, Stardog first **rewrites** it using the information in the schema. So the original query is rewritten into four queries:

```
SELECT ?employee WHERE { ?employee rdf:type :Employee }
SELECT ?employee WHERE { ?employee rdf:type :Manager }
SELECT ?employee WHERE { ?employee rdf:type :SeniorManager }
SELECT ?employee WHERE { ?employee :manages ?x. ?x rdf:type :Employee }
```

Then Stardog executes these queries over the data as if they were written that way to begin with. In fact, Stardog can't tell that they weren't. Reasoning in Stardog **just is query answering** in nearly every case.

The form of the EQ depends on the reasoning type. For OWL 2 QL, every EQ produced by Stardog is **guaranteed to be expanded into a set of queries**. If the reasoning type is OWL 2 RL or EL, then the EQ **may** (but may not) include a recursive rule. ***If a recursive rule is included, Stardog's answers may be incomplete with respect to the semantics of the reasoning type.***

Why Query Rewriting?

Query rewriting has several advantages over materialization. In materialization, the data gets expanded with respect to the schema, not with respect to any actual query. And it's the data—all of the data—that gets expanded, whether any actual query subsequently requires reasoning or not. The schema is used to generate new triples, typically when data is added or removed from the system. However, materialization introduces several thorny issues:

1. **data freshness**. Materialization has to be performed **every time** the data or the schema change. This is particularly unsuitable for applications where the data changes frequently.
2. **data size**. Depending on the schema, materialization can significantly increase the size of the data, sometimes dramatically so. The cost of this data size blowup may be applied to **every** query in terms of increased I/O.
3. **OWL 2 profile reasoning**. Given the fact that QL, RL, and EL are not comparable with respect to expressive power, an application that requires reasoning with more than one profile would need to maintain different corresponding materialized versions of the data.
4. **Resources**. Depending on the size of the original data and the complexity of the schema, materialization may be computationally expensive. And truth maintenance, which materialization requires, is **always** computationally expensive.

Same As Reasoning

Stardog 3.0 adds full support for OWL 2 `sameAs` reasoning. However, `sameAs` reasoning works in a different way than the rest of the reasoning mechanism. The `sameAs` inferences are computed and indexed eagerly so that these materialized inferences can be used directly at query rewriting time. The `sameAs` index is updated automatically as the database is modified so the difference is not of much direct concern to users.

In order to use `sameAs` reasoning, the database configuration option `reasoning.sameas` should be set either at database creation time or at a later time when the database is offline. This can be done through the Web Console or using the command line as follows:

```
$ ./stardog-admin db create -o reasoning.sameas=FULL -n myDB
```

There are legal three values for this option:

- `OFF` disables all `sameAs` inferences, that is, only asserted `sameAs` triples will be included in query results.[30 (#_footnote_30)]
- `ON` computes `sameAs` inferences using only asserted `sameAs` triples, considering the reflexivity, symmetry and transitivity of the `sameAs` relation.
- `FULL` same as `ON` but also considers OWL functional properties, inverse functional properties, and `hasKey` axioms while computing `sameAs` inferences.

NOTE The way `sameAs` reasoning works differs from the OWL semantics slightly in the sense that Stardog designates one canonical individual for each `sameAs` equivalence set and only returns the canonical individual. This avoids the combinatorial explosion in query results while providing the data integration benefits.

Let's see an example showing how `sameAs` reasoning works. Consider the following database where `sameAs` reasoning is set to `ON`:

```
dbpedia:Elvis_Presley
    dbpedia-owl:birthPlace dbpedia:Mississippi ;
    owl:sameAs freebase:en.elvis_presley .

nyt:presley_elvis_per
    nyt:associated_article_count 35 ;
    rdfs:label "Elvis Presley" ;
    owl:sameAs dbpedia:Elvis_Presley .

freebase:en.elvis_presley
        freebase:common.topic.official_website <http://www.elvis.com/> .
```

Now consider the following query and its results:

```
$ ./stardog query --reasoning elvis 'SELECT * { ?s dbpedia-owl:birthPlace ?o; rdfs:label "Elvis Presley" }'
```

```
+-----------------------+---------------------+
|           s           |          o          |
+-----------------------+---------------------+
| nyt:presley_elvis_per | dbpedia:Mississippi |
+-----------------------+---------------------+
```

Let's unpack this carefully. There are three things to note.

First, the query returns only one result even though there are three different URIs that denote Elvis Presley. Second, the URI returned is fixed but chosen randomly. **Stardog picks one of the URIs as the canonical URI and always returns that and only that canonical URI in the results.** If more `sameAs` triples are added the chosen canonical individual may change. Third, it is important to point out that even though only one URI is returned, the effect of `sameAs` reasoning is visible in the results since the `rdfs:label` and `dbpedia-owl:birthPlace` properties were asserted about different instances (i.e., different URIs).

Now, you might be inclined to write queries such as this to get all the properties for a specific URI:

```
SELECT * {
   nyt:presley_elvis_per owl:sameAs ?elvis .
   ?elvis ?p ?o
}
```

However, this is completely unnecessary; rather, you can write the following query and get the same results since `sameAs` reasoning would automatically merge the results for you. Therefore, the query

```
SELECT * {
   nyt:presley_elvis_per ?p ?o
}
```

would return these results:

```
+----------------------------------------+-----------------------+
|                   p                    |           o           |
+----------------------------------------+-----------------------+
| rdfs:label                             | "Elvis Presley"       |
| dbpedia-owl:birthPlace                 | dbpedia:Mississippi   |
| nyt:associated_article_count           | 35                    |
| freebase:common.topic.official_website | http://www.elvis.com/ |
| rdf:type                               | owl:Thing             |
+----------------------------------------+-----------------------+
```

NOTE The URI used in the query does not need to be the same one returned in the results. Thus, the following query would return the exact same results, too:

```
SELECT * {
   dbpedia:Elvis_Presley ?p ?o
}
```

The only time Stardog will return a non-canonical URI in the query results is when you explicitly query for the `sameAs` inferences as in this next example:

```
$ ./stardog query -r elvis 'SELECT * { freebase:en.elvis_presley owl:sameAs ?elvis }'
```

```
+---------------------------+
|           elvis           |
+---------------------------+
| dbpedia:Elvis_Presley     |
| freebase:en.elvis_presley |
| nyt:presley_elvis_per     |
+---------------------------+
```

In the `FULL` `sameAs` reasoning mode, Stardog will also take other OWL axioms into account when computing `sameAs` inferences. Consider the following example:

```
#Everyone has a unique SSN number
:hasSSN a owl:InverseFunctionalProperty , owl:DatatypeProperty .

:JohnDoe :hasSSN "123-45-6789" .
:JDoe :hasSSN "123-45-6789" .

#Nobody can work for more than one company (for the sake of the example)
:worksFor a owl:FunctionalProperty , owl:ObjectProperty ;
        rdfs:domain :Employee ;
        rdfs:range :Company .

:JohnDoe :worksFor :Acme .
:JDoe :worksFor :AcmeInc .

#For each company, there can only be one employee with the same employee ID
:Employee owl:hasKey (:employeeID :worksFor ).

:JohnDoe :employeeID "1234-ABC" .

:JohnD :employeeID "1234-ABC" ;
       :worksFor :AcmeInc .

:JD :employeeID "5678-XYZ" ;
    :worksFor :AcmeInc .

:John :employeeID "1234-ABC" ;
      :worksFor :Emca .
```

For this database, with `sameAs` reasoning set to `FULL`, we would get the following answers:

```
$ ./stardog query -r acme "SELECT * {?x owl:sameAs ?y}"
```

```
+----------+----------+
|    x     |    y     |
+----------+----------+
| :JohnDoe | :JohnD   |
| :JDoe    | :JohnD   |
| :Acme    | :AcmeInc |
+----------+----------+
```

We can follow the chain of inferences to understand how these results were computed:

1. `:JohnDoe owl:sameAs :JohnD` can be computed due to the fact that both have the same SSN numbers and `hasSSN` property is inverse functional.
2. We can infer `:Acme owl:sameAs :AcmeInc` since `:JohnDoe` can work for at most one company.
3. `:JohnDoe owl:sameAs :JohnD` can be inferred using the `owl:hasKey` definition since both individuals are known to work for the same company and have the same employee ID.
4. No more `sameAs` inferences can be computed due to the key definition, since other employees either have different IDs or work for other companies.

Removing Unwanted Inferences

Sometimes reasoning can produce unintended inferences. Perhaps there are modeling errors in the schema or incorrect assertions in the data. After an unintended inference is detected, it might be hard to figure out how to fix it, because there might be multiple different reasons for the inference. The `reasoning explain` command can be used to see the different explanations and the `reasoning undo` command can be used to generate a SPARQL update query that will remove the minimum amount of triples necessary to remove the unwanted inference:

```
$ ./reasoning undo myDB ":AcmeInc a :Person"
```

Performance Hints

The query rewriting approach suggests some guidelines for more efficient query answering.

Hierarchies and Queries

Avoid unnecessarily deep class/property hierarchies.

If you do not need to model several different types of a given class or property in your schema, then don't do that! The reason shallow hierarchies are desirable is that the maximal hierarchy depth in the schema partly determines the maximal size of the EQs produced by Stardog. The larger the EQ, the longer it takes to evaluate, generally.

For example, suppose our schema contains a very thorough and detailed set of subclasses of the class `:Employee`:

```
:Manager rdfs:subClassOf :Employee
:SeniorManager rdfs:subClassOf :Manager
...

:Supervisor rdfs:subClassOf :Employee
:DepartmentSupervisor rdfs:subClassOf :Supervisor
...

:Secretary rdfs:subClassOf :Employee
...
```

If we wanted to retrieve the set of all employees, Stardog would produce an EQ containing a query of the following form for every subclass `:Ci` of `:Employee`:

```
SELECT ?employee WHERE { ?employee rdf:type :Ci }
```

Thus, **ask the most specific query sufficient for your use case**. Why? More general queries—that is, queries that contain concepts high up in the class hierarchy defined by the schema—will typically yield larger EQs.

Domains and Ranges

Specify domain and range of the properties in the schema.

These types of axiom can improve query performance significantly. Consider the following query asking for people and the employees they manage:

```
SELECT ?manager ?employee WHERE
  { ?manager :manages ?employee.
    ?employee rdf:type :Employee. }
```

We know that this query would cause a large EQ given a deep hierarchy of `:Employee` subclasses. However, if we added the following single range axiom:

```
:manages rdfs:range :Employee
```

then the EQ would collapse to

```
 SELECT ?manager ?employee WHERE { ?manager :manages ?employee }
```

which is considerably easier to evaluate.

Very Large Schemas

If you are working with a very large schema like SNOMED then there are couple things to note. First of all, Stardog reasoning works by pulling the complete schema into memory. This means you might need to increase the default memory settings for Stardog for a large schema. Stardog performs all schema reasoning upfront and only once but waits until the first reasoning query arrives. With a large schema, this step can be slow but subsequent reasoning queries will be fast. Also note that, Stardog will update schema reasoning results automatically after the database is modified so there will be some processing time spent then.

Reasoning with very expressive schemas can be time consuming and use a lot of memory. To get the best performance out of Stardog with large schemas, limit the expressivity of your schema to OWL 2 EL (http://www.w3.org/TR/2012/REC-owl2-profiles-20121211/#OWL_2_EL). You can also set the reasoning type of the database to `EL` and Stardog will automatically filter any axiom outside the EL expressivity. See Reasoning Types (#_reasoning_types) for more details on reasoning types. OWL 2 EL allows range declarations for properties and user-defined datatypes but avoiding these two constructs will further improve schema reasoning performance in Stardog.

Not Seeing Expected Results?

Here’s a few things that you might want to consider.

Are variable types ambiguous?

When a SPARQL query gets executed, each variable is bound to a URI, blank node, or to a literal to form a particular result (a collection of these results is a result set). In the context of reasoning, URIs might represent different entities: individuals, classes, properties, etc. According to the relevant standard (http://www.w3.org/TR/sparql11-entailment/#OWLDSEnRegime), **every variable in a SPARQL query must bind to at most one of these types of entity**.

Stardog can often figure out the right entity type from the query itself (e.g., given the triple pattern `?i ?p "a literal"`, we know `?p` is supposed to bind to a data property); however, sometimes this isn’t possible (e.g., `?s ?p ?o`). In case the types can’t be determined automatically, **Stardog logs a message and evaluates the query by making some assumptions, which may not be what the query writer intended, about the types of variables**.

You can add one or more type triples to the query to resolve these ambiguities.[31 (#_footnote_31)]

These "type triples" have the form `?var a TYPE`, where `TYPE` is a URI representing the type of entity to which the variable `?var` is supposed to bind: the most common are `owl:ObjectProperty` or `owl:DatatypeProperty`; in some cases, you might want `owl:NamedIndividual`, or `owl:Class`. For instance, you can use the following query to retrieve all object properties and their characteristics; without the type triple, `?s` will bind only to individuals:

```
SELECT ?o
WHERE {
    ?s rdf:type ?o.
    ?s a owl:ObjectProperty.
}.
```

Since Stardog now knows that `?s` should bind to an object property, it can now infer that `?o` binds to property characteristics of `?s`.

Is the schema where you think it is?

Starting in Stardog 3.0, Stardog will extract the schema from **all named graphs and the default graph**.

If you require that the schema only be extracted from one or more specific named graphs, then you must tell Stardog where to find the schema. See database configuration options (#_configuration_options) for details. You can also use the `reasoning schema` (https://stardog.com/docs/man/reasoning-schema) command to export the contents of the schema to see exactly what is included in the schema that Stardog uses.

Are you using the right reasoning type?

Perhaps some of the modeling constructs (a.k.a. axioms) in your database are being ignored. By default, Stardog uses the `SL` reasoning type. You can find out which axioms are being ignored by looking at the Stardog log file.

Are you using DL?

Stardog supports full OWL 2 DL reasoning but only for data that fits into main memory.

Are you using SWRL?

SWRL rules—whether using SWRL syntax or Stardog Rules Syntax—are only taken into account using the **SL** reasoning type.

Do you know what to expect?

The OWL 2 primer (http://www.w3.org/TR/owl2-primer/) is a good place to start.

Known Issues

Stardog 5.3.6 does not

- Follow ontology `owl:imports` statements automatically; any imported OWL ontologies that are required must be loaded into a Stardog database in the normal way.
- Handle recursive queries. If recursion is necessary to answer the query with respect to the schema, results will be sound (*no wrong answers*) but potentially incomplete (*some correct answers not returned*) with respect to the requested reasoning type.

Terminology

This chapter uses the following terms of art.

Databases

A **database** (DB), a.k.a. ontology, is composed of two different parts: the schema or **Terminological Box** (TBox) and the data or **Assertional Box** (ABox). Analogus to relational databases, the TBox can be thought of as the schema, and the ABox as the data. In other words, the TBox is a set of **axioms**, whereas the ABox is a set of **assertions**.

As we explain in OWL 2 Profiles (#_owl_2_profiles), the kinds of assertion and axiom that one might use for a particular database are determined by the fragment of OWL 2 to which you'd like to adhere. In general, you should choose the OWL 2 profile that most closely fits the data modeling needs of your application.

The most common data assertions are class and property assertions. Class assertions are used to state that a particular individual is an instance of a given class. Property assertions are used to state that two particular individuals (or an individual and a literal) are related via a given property. For example, suppose we have a DB $MyDB_2$ that contains the following data assertions. We use the usual standard prefixes for RDF(S) and OWL.

```
:complexible rdf:type :Company
:complexible :maintains :Stardog
```

Which says that `:complexible` is a company, and that `:complexible` maintains `:Stardog`.

The most common schema axioms are subclass axioms. Subclass axioms are used to state that every instance of a particular class is also an instance of another class. For example, suppose that $MyDB_2$ contains the following TBox axiom:

```
:Company rdfs:subClassOf :Organization
```

stating that companies are a type of organization.

Queries

When reasoning is enabled, Stardog executes SPARQL queries depending on the type of Basic Graph Patterns they contain. A BGP is said to be an "ABox BGP" if it is of one of the following forms:

- **term**$_1$ `rdf:type` **uri**

- **term$_1$ uri term$_2$**
- **term$_1$** `owl:differentFrom` **term$_2$**
- **term$_1$** `owl:sameAs` **term$_2$**

A BGP is said to be a TBox BGP if it is of one of the following forms:

- **term$_1$** `rdfs:subClassOf` **term$_2$**
- **term$_1$** `owl:disjointWith` **term$_2$**
- **term$_1$** `owl:equivalentClass` **term$_2$**
- **term$_1$** `rdfs:subPropertyOf` **term$_2$**
- **term$_1$** `owl:equivalentProperty` **term$_2$**
- **term$_1$** `owl:inverseOf` **term$_2$**
- **term$_1$** `owl:propertyDisjointWith` **term$_2$**
- **term$_1$** `rdfs:domain` **term$_2$**
- **term$_1$** `rdfs:range` **term$_2$**

A BGP is said to be a Hybrid BGP if it is of one of the following forms:

- **term$_1$** `rdf:type` **?var**
- **term$_1$?var term$_2$**

where **term** (possibly with subscripts) is either an URI or variable; **uri** is a URI; and **?var** is a variable.

When executing a query, ABox BGPs are handled by Stardog. TBox BGPs are executed by Pellet embedded in Stardog. Hybrid BGPs by a combination of both.

Reasoning

Intuitively, reasoning with a DB means to make implicit knowledge explicit. There are two main use cases for reasoning: to infer implicit knowledge and to discover modeling errors.

With respect to the first use case, recall that MyDB$_2$ contains the following assertion and axiom:

```
 :complexible rdf:type :Company
 :Company rdfs:subClassOf :Organization
```

From this DB, we can use Stardog in order to **infer** that `:complexible` is an organization:

```
:complexible rdf:type :Organization
```

Using reasoning in order to infer implicit knowledge in the context of an enterprise application can lead to simpler queries. Let us suppose, for example, that MyDB$_2$ contains a complex class hierarchy including several types of organization (including company). Let us further suppose that our application requires to use Stardog in order to get the list of all considered organizations. If Stardog were used **with reasoning**, then we would need only issue the following simple query:

```
SELECT ?org WHERE { ?org rdf:type :Organization}
```

In contrast, if we were using Stardog **with no reasoning**, then we would have to issue a more complex query that considers all possible types of organization, thus coupling queries to domain knowledge in a tight way:

```
SELECT ?org WHERE
              { { ?org rdf:type :Organization } UNION
              { ?org rdf:type :Company } UNION
...
}
```

Which of these queries seems more loosely coupled and more resilient to change?

Stardog can also be used in order to discover modeling errors in a DB. The most common modeling errors are **unsatisfiable** classes and **inconsistent** DBs.

An unsatisfiable class is simply a class that cannot have any instances. Say, for example, that we added the following axioms to MyDB$_2$:

```
:Company owl:disjointWith :Organization
:LLC owl:equivalentClass :Company and :Organization
```

stating that companies cannot be organizations and vice versa, and that an LLC is a company and an organization. The disjointness axiom causes the class `:LLC` to be unsatisfiable because, for the DB to be free of any logical contradiction, there can be no instances of `:LLC`.

Asserting (or inferring) that an unsatisfiable class has an instance, causes the DB to be **inconsistent**. In the particular case of MyDB$_2$, we know that `:complexible` is a company **and** an organization; therefore, we also know that it is an instance of `:LLC`, and as `:LLC` is known to be unsatisfiable, we have that MyDB$_2$ is inconsistent.

Using reasoning in order to discover modeling errors in the context of an enterprise application is useful in order to maintain a correct contradiction-free model of the domain. In our example, we discovered that `:LLC` is unsatisfiable and MyDB$_2$ is inconsistent, which leads us to believe that there is a modeling error in our DB. In this case, it is easy to see that the problem is the disjointness axiom between `:Company` and `:Organization`.

OWL 2 Profiles

As explained in the OWL 2 Web Ontology Language Profiles Specification (http://www.w3.org/TR/owl2-profiles/), an OWL 2 profile is a reduced version of OWL 2 that trades some expressive power for efficiency of reasoning. There are three OWL 2 profiles, each of which achieves efficiency differently.

- OWL 2 QL (http://www.w3.org/TR/owl2-profiles/#OWL_2_QL) is aimed at applications that use very large volumes of instance data, and where query answering is the most important reasoning task. The expressive power of the profile is necessarily limited; however, it includes most of the main features of conceptual models such as UML class diagrams and ER diagrams.
- OWL 2 EL (http://www.w3.org/TR/owl2-profiles/#OWL_2_EL) is particularly useful in applications employing ontologies that contain very large numbers of properties and classes. This profile captures the expressive power used by many such ontologies and is a subset of OWL 2 for which the basic reasoning problems can be performed in time that is polynomial with respect to the size of the ontology.
- OWL 2 RL (http://www.w3.org/TR/owl2-profiles/#OWL_2_RL) is aimed at applications that require scalable reasoning without sacrificing too much expressive power. It is designed to accommodate OWL 2 applications that can trade the full expressivity of the language for efficiency, as well as RDF(S) applications that need some added expressivity.

Each profile restricts the kinds of axiom and assertion that can be used in a DB. Colloquially, QL is the least expressive of the profiles, followed by RL and EL; however, strictly speaking, no profile is more expressive than any other as they provide incomparable sets of constructs.

Stardog supports the three profiles of OWL 2. Notably, since TBox BGPs are handled completely by Pellet, Stardog supports reasoning for the whole of OWL 2 for queries containing TBox BGPs only.

VALIDATING CONSTRAINTS

Stardog Integrity Constraint Validation ("ICV") validates RDF data stored in a Stardog database according to data rules (i.e., "constraints") described by users and that make sense for their domain, application, and data. These constraints may be written in SPARQL, OWL, or SWRL. This chapter explains how to use ICV.

The use of high-level languages (OWL 2, SWRL, and SPARQL) to validate RDF data using **closed world semantics** is one of Stardog’s unique capabilities. Using high level languages like OWL, SWRL, and SPARQL as schema or constraint languages for RDF and Linked Data has several advantages:

- Unifying the domain model with data quality rules
- Aligning the domain model and data quality rules with the integration model and language (i.e., RDF)
- Being able to query the domain model, data quality rules, integration model, mapping rules, etc with SPARQL
- Being able to use automated reasoning about all of these things to insure logical consistency, explain errors and problems, etc.

TIP Data Quality with ICV (https://www.stardog.com/blog/data-quality-with-icv/)

Getting Started with ICV

See the extended ICV tutorial (https://github.com/Complexible/stardog-examples/tree/master/examples/cli/icv) in the stardog-examples repo on Github for more details.

ICV & OWL 2 Reasoning

An integrity constraint may be satisfied or violated in either of two ways: by an explicit statement in a Stardog database or by a statement that's been validly inferred by Stardog.

When ICV is enabled for a Stardog database, it has to be enabled with respect to a reasoning type or level. The valid choices of reasoning type are any type or kind of reasoning supported by Stardog. See OWL & Rule Reasoning (#_owl_rule_reasoning) for the details.

So ICV is performed with three inputs:

1. a Stardog database,
2. a set of constraints, and
3. a reasoning type (which may be, of course, no reasoning).

This is the case because domain modelers, ontology developers, or integrity constraint authors must consider the interactions between explicit and inferred statements and how these are accounted for in integrity constraints.

Using ICV from CLI

To add constraints to a database:

```
$ stardog-admin icv add myDb constraints.rdf
```

To drop all constraints from a database:

```
$ stardog-admin icv drop myDb
```

To remove one or more specific constraints from a database:

```
$ stardog-admin icv remove myDb constraints.rdf
```

To convert new or existing constraints into SPARQL queries for export:

```
$ stardog icv convert myDb constraints.rdf
```

To explain a constraint violation:

```
$ stardog icv explain --contexts http://example.org/context1 http://example.org/context2 -- myDb
```

To export constraints:

```
$ stardog icv export myDb constraints.rdf
```

To validate a database (or some named graphs) with respect to constraints:

```
$ stardog icv validate --contexts http://example.org/context1 http://example.org/context2 -- myDb
```

ICV Guard Mode

Stardog will also apply constraints as part of its transactional cycle and fail transactions that violate constraints. We call this "guard mode". It must be enabled explicitly in the database configuration options. Using the command line, these steps are as follows:

```
$ ./stardog-admin db offline --timeout 0 myDb #take the database offline
$ ./stardog-admin metadata set -o icv.enabled=true myDb #enable ICV
$ ./stardog-admin db online myDb #put the database online
```

In the Web Console you can set the database offline, click [**Edit**] , change the "ICV Enable" value, click [**Save**] and set the database online again.

Once guard mode is enabled, modifications of the database (via SPARQL Update or any other method), whether adds or deletes, that violate the integrity constraints will cause the transaction to fail.

Explaining ICV Violations

ICV violations can be explained using Stardog’s Proof Trees (#_proof_trees). The following command will explain the IC violations for constraints stored in the database:

```
$ stardog icv explain --reasoning "myDB"
```

The command is flexible to change the number of violations displayed, and to explain violations for external constraints by passing the file with constraints as an additional argument:

```
$ stardog icv explain --reasoning --limit 2 "myDB" constraints.ttl
```

Security Note

WARNING There is a security implication in this design that may not be obvious. Changing the reasoning type associated with a database and integrity constraint validation may have serious security implications with respect to a Stardog database and, thus, may only be performed by a user role with sufficient privileges for that action.

Repairing ICV Violations

Stardog 3.0 adds support for automatic repair of some kinds of integrity violation. This can be accomplished programmatically via API, as well as via CLI using the `icv fix` subcommand.

```
$ stardog help icv fix
```

Repair plans are emitted as a sequence of SPARQL Update queries, which means they can be applied to any system that understands SPARQL Update. If you pass `--execute` the repair plan will be applied immediately.

`icv fix` will repair violations of all constraints in the database; if you'd prefer to fix the violations for only some constraints, you can pass those constraints as an additional argument. Although a possible (but trivial) fix for any violation is to remove one or more constraints, `icv fix` does not suggest that kind of repair, even though it may be appropriate in some cases.

ICV Examples

Stardog ICV has a formal semantics (/icv/icv-specification.html). But let's just look at some examples instead; these examples use OWL 2 Manchester syntax, and they assume a simple data schema, which is available as an OWL ontology (/icv/company.owl) and as a UML diagram (/icv/ClassDiagram.png). The examples assume that the default namespace is `http://example.com/company.owl# (http://example.com/company.owl#)` and that `xsd:` is bound to the standard, `http://www.w3.org/2001/XMLSchema# (http://www.w3.org/2001/XMLSchema#)`.

Reference Java code (https://gist.github.com/1333767) is available for each of the following examples and is also distributed with Stardog.

Subsumption Constraints

This kind of constraint guarantees certain subclass and superclass (i.e., subsumption) relationships exist between instances.

Managers must be employees.

Constraint

```
:Manager rdfs:subClassOf :Employee
```

Database A (invalid)

```
:Alice a :Manager .
```

Database B (valid)

```
:Alice a :Manager , :Employee .
```

This constraint says that if an RDF individual is an instance of `Manager`, then it must also be an instance of `Employee`. In A, the only instance of `Manager`, namely `Alice`, is not an instance of `Employee`; therefore, A is invalid. In B, `Alice` is an instance of Database both `Manager` and `Employee`; therefore, B is valid.

Domain-Range Constraints

These constraints control the types of subjects and objects used with a property.

Only project leaders can be responsible for projects.

Constraint

```
:is_responsible_for rdfs:domain :Project_Leader ;
                    rdfs:range :Project .
```

Database A (invalid)

```
:Alice :is_responsible_for :MyProject .

:MyProject a :Project .
```

Database B (invalid)

```
:Alice a :Project_Leader ;
        :is_responsible_for :MyProject .
```

Database C (valid)

```
:Alice a :Project_Leader ;
        :is_responsible_for :MyProject .

:MyProject a :Project .
```

This constraint says that if two RDF instances are related to each other via the property `is_responsible_for`, then the subject must be an instance of `Project_Leader` and the object must be an instance of `Project`. In Database A, there is only one pair of individuals related via `is_responsible_for`, namely `(Alice, MyProject)`, and `MyProject` is an instance of `Project`; but `Alice` is not an instance of `Project_Leader`. Therefore, A is invalid. In B, `Alice` is an instance of `Project_Leader`, but `MyProject` is not an instance of `Project`; therefore, B is not valid. In C, `Alice` is an instance of `Project_Leader`, and `MyProject` is an instance of `Project`; therefore, C is valid.

Only employees can have an SSN.

Constraint

```
:ssn rdfs:domain :Employee
```

Database A (invalid)

```
:Bob :ssn "123-45-6789" .
```

Database B (valid)

```
:Bob a :Employee ;
        :ssn "123-45-6789" .
```

This constraint says that if an RDF instance `i` has a data assertion via the the property `SSN`, then `i` must be an instance of `Employee`. In A, `Bob` is not an instance of `Employee` but has `SSN`; therefore, A is invalid. In B, `Bob` is an instance of `Employee`; therefore, B is valid.

A date of birth must be a date.

Constraint

```
:dob rdfs:range xsd:date
```

Database A (invalid)

```
:Bob :dob "1970-01-01" .
```

Database B (valid)

```
:Bob :dob "1970-01-01"^^xsd:date
```

This constraint says that if an RDF instance `i` is related to a literal `l` via the data property `DOB`, then `l` must have the XML Schema type `xsd:date`. In A, `Bob` is related to the untyped literal `"1970-01-01"` via `DOB` so A is invalid. In B, the literal `"1970-01-01"` is properly typed so it's valid.

Participation Constraints

These constraints control whether or not an RDF instance participates in some specified relationship.

Each supervisor must supervise at least one employee.

Constraint

```
#this constraint is very concise in Terp syntax:
#:Supervisor rdfs:subClassOf (:supervises some :Employee)

:Supervisor rdfs:subClassOf
              [ a owl:Restriction ;
                owl:onProperty :supervises ;
                owl:someValuesFrom :Employee
              ] .
```

Database A (valid)

```
:Alice a owl:Thing .
```

Database B (invalid)

```
:Alice a :Supervisor .
```

Database C (invalid)

```
:Alice a :Supervisor ;
        :supervises :Bob .
```

Database D (valid)

```
:Alice a :Supervisor ;
        :supervises :Bob .

:Bob a :Employee
```

This constraint says that if an RDF instance `i` is of type `Supervisor`, then `i` must be related to an individual `j` via the property `supervises` and also that `j` must be an instance of `Employee`. In A, `Supervisor` has no instances; therefore, A is trivially valid. In B, the only instance of `Supervisor`, namely `Alice`, is related to no individual; therefore, B is invalid. In C, `Alice` is related to `Bob` via `supervises`, but `Bob` is not an instance of `Employee`; therefore, C is invalid. In D, `Alice` is related to `Bob` via `supervises`, and `Bob` is an instance of `Employee`; hence, D is valid.

Each project must have a valid project number.

Constraint

```
#Again, this constraint in Terp syntax rocks the hizzous:
#:Project rdfs:subClassOf (:number some xsd:integer[>= 0, < 5000])

:Project rdfs:subClassOf
              [ a owl:Restriction ;
                owl:onProperty :number ;
                owl:someValuesFrom
                        [ a rdfs:Datatype ;
                          owl:onDatatype xsd:integer ;
                          owl:withRestrictions ([xsd:minInclusive 0] [ xsd:maxExclusive 5000])
                        ]
              ] .
```

Database A (valid)

```
:MyProject a owl:Thing .
```

Database B (invalid)

```
:MyProject a :Project
```

Database C (invalid)

```
:MyProject a :Project ;
        :number "23" .
```

Database D (invalid)

```
:MyProject a :Project ;
        :number "6000"^^xsd:integer .
```

Database E (valid)

```
:MyProject a :Project ;
        :number "23"^^xsd:integer .
```

This constraint says that if an RDF instance `i` is of type `Project`, then `i` must be related via the property `number` to an integer between `0` and `5000` (inclusive)—that is, projects have project numbers in a certain range. In A, the individual `MyProject` is not known to be an instance of `Project` so the constraint does not apply at all and A is valid. In B, `MyProject` is an instance of `Project` but doesn't have any data assertions via `number` so A is invalid. In C, `MyProject` does have a data property assertion via `number` but the literal `"23"` is untyped—that is, it's not an integer —therefore, C is invalid. In D, `MyProject` is related to an integer via `number` but it is out of the range: D is invalid. Finally, in E, `MyProject` is related to the integer `23` which is in the range of `[0,5000]` so E is valid.

Cardinality Constraints

These constraints control the number of various relationships or property values.

Employees must not work on more than 3 projects.

Constraint

```
#Constraint in Terp syntax:
#:Employee rdfs:subClassOf (:works_on max 3 :Project)

:Employee rdfs:subClassOf
              [ a owl:Restriction ;
                owl:onProperty :works_on;
                owl:maxQualifiedCardinality "3"^^xsd:nonNegativeInteger ;
                owl:onClass :Project
              ] .
```

Database A (valid)

```
:Bob a owl:Thing.
```

Database B (valid)

```
:Bob a :Employee ;
        :works_on :MyProject .

:MyProject a :Project .
```

Database C (invalid)

```
:Bob a :Employee ;
        :works_on :MyProject , :MyProjectFoo , :MyProjectBar , :MyProjectBaz .

:MyProject a :Project .

:MyProjectFoo a :Project .

:MyProjectBar a :Project .

:MyProjectBaz a :Project .
```

If an RDF instance `i` is an `Employee`, then `i` must not be related via the property `works_on` to more than 3 instances of `Project`. In A, `Bob` is not known to be an instance of `Employee` so the constraint does not apply and the A is valid. In B, `Bob` is an instance of `Employee` but is known to work on only a single project, namely `MyProject`, so B is valid. In C, `Bob` is related to 4 instances of `Project` via `works_on`.

NOTE Stardog ICV implements a weak form of the **unique name assumption**, that is, it assumes that things which have different names are, in fact, different things.[32]

Since Stardog ICV uses closed world (instead of open world) semantics,[33 (#_footnote_33)] it assumes that the different projects with different names are, in fact, separate projects, which (**in this case**) violates the constraint and makes C invalid.

Departments must have at least 2 employees.

Constraint

```
#Constraint in Terp syntax:
#:Department rdfs:subClassOf (inverse :works_in min 2 :Employee)

:Department rdfs:subClassOf
              [ a owl:Restriction ;
                owl:onProperty [owl:inverseOf :works_in] ;
                owl:minQualifiedCardinality "2"^^xsd:nonNegativeInteger ;
                owl:onClass :Employee
              ] .
```

Database A (valid)

```
:MyDepartment a owl:NamedIndividual .
```

Database B (invalid)

```
:MyDepartment a :Department .

:Bob a :Employee ;
        :works_in :MyDepartment .
```

Database C (valid)

```
:MyDepartment a :Department .

:Alice a :Employee ;
        :works_in :MyDepartment .

:Bob a :Employee ;
        :works_in :MyDepartment .
```

This constraint says that if an RDF instance `i` is a `Department`, then there should exist at least 2 instances `j` and `k` of class `Employee` which are related to `i` via the property `works_in` (or, equivalently, `i` should be related to them via the inverse of `works_in`). In A, `MyDepartment` is not known to be an instance of `Department` so the constraint does not apply. In B, `MyDepartment` is an instance of `Department` but only one instance of `Employee`, namely `Bob`, is known to work in it, so B is invalid. In C, `MyDepartment` is related to the individuals `Bob` and `Alice`, which are both instances of `Employee` and (again, due to weak Unique Name Assumption in Stardog ICV), are assumed to be distinct, so C is valid.

Managers must manage exactly 1 department.

Constraint

```
#Constraint in Terp syntax:
#:Manager rdfs:subClassOf (:manages exactly 1 :Department)

:Manager rdfs:subClassOf
              [ a owl:Restriction ;
                owl:onProperty :manages ;
                owl:qualifiedCardinality "1"^^xsd:nonNegativeInteger ;
                owl:onClass :Department
              ] .
```

Database A (valid)

```
    Individual: Isabella
```

Database B (invalid)

```
:Isabella a :Manager .
```

Database C (invalid)

```
:Isabella a :Manager ;
        :manages :MyDepartment .
```

Database D (valid)

```
:Isabella a :Manager ;
        :manages :MyDepartment .

:MyDepartment a :Department .
```

Database E (invalid)

```
:Isabella a :Manager ;
        :manages :MyDepartment , :MyDepartment1 .

:MyDepartment a :Department .

:MyDepartment1 a :Department .
```

This constraint says that if an RDF instance `i` is a `Manager`, then it must be related to exactly 1 instance of `Department` via the property `manages`. In A, the individual `Isabella` is not known to be an instance of `Manager` so the constraint does not apply and A is valid. In B, `Isabella` is an instance of `Manager` but is not related to any instances of `Department`, so B is invalid. In C, `Isabella` is related to the individual `MyDepartment` via the property `manages` but `MyDepartment` is not known to be an instance of `Department`, so C is invalid. In D, `Isabella` is related to exactly one instance of `Department`, namely `MyDepartment`, so D is valid. Finally, in E, `Isabella` is related to two (assumed to be) distinct (again, because of weak UNA) instances of `Department`, namely `MyDepartment` and `MyDepartment1`, so E is invalid.

Entities may have no more than one name.

Constraint

```
:name a owl:FunctionalProperty .
```

Database A (valid)

```
:MyDepartment a owl:Thing .
```

Database B (valid)

```
:MyDepartment :name "Human Resources" .
```

Database C (invalid)

```
:MyDepartment :name "Human Resources" , "Legal" .
```

This constraint says that no RDF instance `i` can have more than one assertion via the data property `name`. In A, `MyDepartment` does not have any data property assertions so A is valid. In B, `MyDepartment` has a single assertion via `name`, so the ontology is also valid. In C, `MyDepartment` is related to 2 literals, namely `"Human Resources"` and `"Legal"`, via `name`, so C is invalid.

Property Constraints

These constraints control how instances are related to one another via properties.

The manager of a department must work in that department.

Constraint

```
:manages rdfs:subPropertyOf :works_in .
```

Database A (invalid)

```
:Bob :manages :MyDepartment
```

Database B (valid)

```
:Bob :works_in :MyDepartment ;
        :manages :MyDepartment .
```

This constraint says that if an RDF instance `i` is related to `j` via the property `manages`, then `i` must also be related to `j` via the property `works_in`. In A, `Bob` is related to `MyDepartment` via `manages`, but not via `works_in`, so A is invalid. In B, `Bob` is related to `MyDepartment` via both `manages` and `works_in`, so B is valid.

Department managers must supervise all the department's employees.

Constraint

```
:is_supervisor_of owl:propertyChainAxiom (:manages [owl:inverseOf :works_in]) .
```

Database A (invalid)

```
:Jose :manages :MyDepartment ;
        :is_supervisor_of :Maria .

:Maria :works_in :MyDepartment .

:Diego :works_in :MyDepartment .
```

Database B (valid)

```
:Jose :manages :MyDepartment ;
        :is_supervisor_of :Maria , :Diego .

:Maria :works_in :MyDepartment .

:Diego :works_in :MyDepartment .
```

This constraint says that if an RDF instance `i` is related to `j` via the property `manages` and `k` is related to `j` via the property `works_in`, then `i` must be related to `k` via the property `is_supervisor_of`. In A, `Jose` is related to `MyDepartment` via `manages`, `Diego` is related to `MyDepartment` via `works_in`, but `Jose` is not related to `Diego` via any property, so A is invalid. In B, `Jose` is related to `Maria` and `Diego`--who are both related to `MyDepartment` by way of `works_in`--via the property `is_supervisor_of`, so B is valid.

Complex Constraints

Constrains may be arbitrarily complex and include many conditions.

Employee Constraints

Each employee works on at least one project, or supervises at least one employee that works on at least one project, or manages at least one department.

Constraint

```
#Constraint in Terp syntax:
#how are you not loving Terp by now?!
#:Employee rdfs:subClassOf (:works_on some (:Project or
#(:supervises some (:Employee and (:works_on some :Project))) or (:manages some :Department)))

:Employee rdfs:subClassOf
              [ a owl:Restriction ;
                owl:onProperty :works_on ;
                owl:someValuesFrom
                        [ owl:unionOf (:Project
                                      [ a owl:Restriction ;
                                        owl:onProperty :supervises ;
                                        owl:someValuesFrom
                                              [ owl:intersectionOf (:Employee
                                                                    [ a owl:Restriction ;
                                                                      owl:onProperty :works_on ;
                                                                      owl:someValuesFrom :Project
                                                                    ])
                                              ]
                                      ]
                                      [ a owl:Restriction ;
                                        owl:onProperty :manages ;
                                        owl:someValuesFrom :Department
                                      ])
                        ]
              ] .
```

Database A (invalid)

```
:Esteban a :Employee .
```

Database B (invalid)

```
:Esteban a :Employee ;
        :supervises :Lucinda .

:Lucinda a :Employee .
```

Database C (valid)

```
:Esteban a :Employee ;
        :supervises :Lucinda .

:Lucinda a :Employee ;
        :works_on :MyProject .

:MyProject a :Project .
```

Database D (valid)

```
:Esteban a :Employee ;
        :manages :MyDepartment .

:MyDepartment a :Department .
```

Database E (valid)

```
:Esteban a :Employee ;
        :manages :MyDepartment ;
        :works_on :MyProject .

:MyDepartment a :Department .

:MyProject a :Project .
```

This constraint says that if an individual `i` is an instance of `Employee`, then at least one of three conditions must be met:

- it is related to an instance of `Project` via the property `works_on`
- it is related to an instance `j` via the property `supervises` and `j` is an instance of `Employee` and is also related to some instance of `Project` via the property `works_on`
- it is related to an instance of `Department` via the property `manages`.

A and B are invalid because none of the conditions are met. C meets the second condition: `Esteban` (who is an `Employee`) is related to `Lucinda` via the property `supervises` whereas `Lucinda` is both an `Employee` and related to `MyProject`, which is a `Project`, via the property `works_on`. D meets the third condition: `Esteban` is related to an instance of `Department`, namely `MyDepartment`, via the property `manages`. Finally, E meets the first and the third conditions because in addition to managing a department `Esteban` is also related an instance of `Project`, namely `MyProject`, via the property `works_on`.

Employees and US government funding

Only employees who are American citizens can work on a project that receives funds from a US government agency.

Constraint

```
#Constraint in Terp syntax:
#:Project and (:receives_funds_from some :US_Government_Agency)) rdfs:subClassOf
#(inverse :works_on only (:Employee and (:nationality value "US")))

[ owl:intersectionOf (:Project
                       [ a owl:Restriction ;
                         owl:onProperty :receives_funds_from ;
                         owl:someValuesFrom :US_Government_Agency
                       ]) .
] rdfs:subClassOf
              [ a owl:Restriction ;
                owl:onProperty [owl:inverseOf :works_on] ;
                owl:allValuesFrom [ owl:intersectionOf (:Employee
                                                      [ a owl:Restriction ;
                                                        owl:hasValue "US" ;
                                                        owl:onProperty :nationality
                                                      ])
                                  ]
              ] .
```

Database A (valid)

```
:MyProject a :Project ;
        :receives_funds_from :NASA .

:NASA a :US_Government_Agency
```

Database B (invalid)

```
:MyProject a :Project ;
        :receives_funds_from :NASA .

:NASA a :US_Government_Agency .

:Andy a :Employee ;
        :works_on :MyProject .
```

Database C (valid)

```
:MyProject a :Project ;
        :receives_funds_from :NASA .

:NASA a :US_Government_Agency .

:Andy a :Employee ;
        :works_on :MyProject ;
        :nationality "US" .
```

Database D (invalid)

```
:MyProject a :Project ;
        :receives_funds_from :NASA .

:NASA a :US_Government_Agency .

:Andy a :Employee ;
        :works_on :MyProject ;
        :nationality "US" .

:Heidi a :Supervisor ;
        :works_on :MyProject ;
        :nationality "US" .
```

Database E (valid)

```
:MyProject a :Project ;
        :receives_funds_from :NASA .

:NASA a :US_Government_Agency .

:Andy a :Employee ;
        :works_on :MyProject ;
        :nationality "US" .

:Heidi a :Supervisor ;
        :works_on :MyProject ;
        :nationality "US" .

:Supervisor rdfs:subClassOf :Employee .
    SubClassOf: Employee
```

This constraint says that if an individual `i` is an instance of `Project` and is related to an instance of `US_Government_Agency` via the property `receives_funds_from`, then any individual `j` which is related to `i` via the property `works_on` must satisfy two conditions:

- it must be an instance of `Employee`
- it must not be related to any literal other than `"US"` via the data property `nationality`.

A is valid because there is no individual related to `MyProject` via `works_on`, so the constraint is trivially satisfied. B is invalid since `Andy` is related to `MyProject` via `works_on`, `MyProject` is an instance of `Project` and is related to an instance of `US_Government_Agency`, that is, `NASA`, via `receives_funds_from`, but `Andy` does not have any data property assertions. C is valid because both conditions are met. D is not valid because `Heidi` violated the first condition: she is related to `MyProject` via `works_on` but is not known to be an instance of `Employee`. Finally, this is fixed in E—by way of a handy OWL axiom—which states that every instance of `Supervisor` is an instance of `Employee`, so `Heidi` is inferred to be an instance of `Employee` and, consequently, E is valid.[34 (#_footnote_34)]

If you made it this far, you deserve a drink!

Constraints Formats

In addition to OWL, ICV constraints can be expressed in SPARQL and Stardog Rules. In both cases, the constraints define queries and rules to find violations. These constraints can be added individually, or defined together in a file as shown below:

```
@prefix rule: <tag:stardog:api:rule:> .
@prefix icv: <tag:stardog:api:icv:> .

# Rule Constraint
[] a rule:SPARQLRule ;

   rule:content """

    prefix : <http://example.org/>

        IF {
            ?x a :Employee .
        }
        THEN {
                ?x :employeeNum ?number .
        }

   """ .


# SPARQL Constraint
[] a icv:Constraint ;

   icv:query """
    prefix : <http://example.org/>

        select * {

        ?x a :Employee .

            FILTER NOT EXISTS {
                ?x :employeeNum ?number .
            }
        }

   """ .
```

Using ICV Programmatically

Here we describe how to use Stardog ICV via the SNARL APIs. For more information on using SNARL in general, please refer to the chapter on Java Programming (#_java_programming).

There is command-line interface support for many of the operations necessary to using ICV with a Stardog database; please see Administering Stardog (#_administering_stardog) for details.

To use ICV in Stardog, one must:

1. create some constraints
2. associate those constraints with a Stardog database

Creating Constraints

`Constraints` (/docs/5.3.6/java/snarl/com/complexible/stardog/icv/constraint) can be created using the `ConstraintFactory` (/docs/5.3.6/java/snarl/com/complexible/stardog/icv/constraintfactory) which provides methods for creating integrity constraints. `ConstraintFactory` expects your constraints, if they are defined as OWL axioms, as RDF triples (or graph). To aid in authoring constraints in OWL, `ExpressionFactory` (/docs/5.3.6/java/snarl/com/complexible/common/openrdf/util/expressionfactory) is provided for building the RDF equivalent (http://www.w3.org/TR/owl2-mapping-to-rdf/) of the OWL axioms of your constraint.

You can also write your constraints in OWL in your favorite editor and load them into the database from your OWL file.

We recommend defining your constraints as OWL axioms, but you are free to define them using SPARQL `SELECT` queries. If you choose to define a constraint using a SPARQL `SELECT` query, please keep in mind that if your query returns results, those are interpreted as the violations of the integrity constraint.

An example of creating a simple constraint using `ExpressionFactory`:

```
Unresolved directive in icv.ad -
include::https://gist.githubusercontent.com/mhgrove/1333767/raw/CreateConstraint.java[]
```

Adding Constraints to Stardog

The ICVConnection (/docs/5.3.6/java/snarl/com/complexible/stardog/icv/api/icvconnection) interface provides programmatic access to the ICV support in Stardog. It provides support for adding, removing and clearing integrity constraints in your database as well as methods for checking whether or not the data is valid; and when it's not, retrieving the list of violations.

This example shows how to add an integrity constraint to a Stardog database.

```
Unresolved directive in icv.ad -
include::https://gist.githubusercontent.com/mhgrove/1333767/raw/AddConstraint.java[]
```

Here we show how to add a set of constraints as defined in a local OWL ontology.

```
Unresolved directive in icv.ad -
include::https://gist.githubusercontent.com/mhgrove/1333767/raw/AddConstraint2.java[]
```

IC Validation

Checking whether or not the contents of a database are valid is easy. Once you have an `ICVConnection` (/docs/5.3.6/java/snarl/com/complexible/stardog/icv/api/icvconnection) you can simply call its `isValid()` (/docs/5.3.6/java/snarl/com/complexible/stardog/icv/api/icvconnection#isValid()) method which will return whether or not the contents of the database are valid with respect to the constraints associated with that database. Similarly, you can provide some `constraints` (/docs/5.3.6/java/snarl/com/complexible/stardog/icv/constraint) to the `isValid()` method to see if the data in the database is invalid for those specific constraints; which can be a subset of the constraints associated with the database, or they can be new constraints you are working on.

If the data is invalid for some constraints—either the explicit constraints in your database or a new set of constraints you have authored—you can get some information about what the violation was from the SNARL IC Connection. `ICVConnection.getViolationBindings()` (/docs/5.3.6/java/snarl/com/complexible/stardog/icv/api/icvconnection#getViolationBindings()) will return the constraints which are violated, and for each constraint, you can get the violations as the set of bindings that satisfied the constraint query. You can turn the bindings into the individuals which are in the violation using `ICV.asIndividuals()` (/docs/5.3.6/java/snarl/com/complexible/stardog/icv/icv#asIndividuals()).

ICV and Transactions

In addition to using the ICConnection as a data oracle to tell whether or not your data is valid with respect to some constraints, you can also use Stardog's ICV support to protect your database from invalid data by using ICV as a guard within transactions.

When guard mode for ICV is enabled in Stardog, each commit is inspected to ensure that the contents of the database are valid for the set of constraints that have been associated with it. Should someone attempt to commit data which violates one or more of the constraints defined for the database, the commit will fail and the data will not be added/removed from your database.

By default, reasoning is not used when you enable guard mode, however you are free to specify any of the reasoning types supported by Stardog when enabling guard mode. If you have provided a specific reasoning type for guard mode it will be used during validation of the integrity constraints. This means you can author your constraints with the expectation of inference results satisfying a constraint.

```
Unresolved directive in icv.ad -
include::https://gist.githubusercontent.com/mhgrove/1333782/raw/CreateDiskAndICV.java[]
```

This illustrates how to create a persistent disk database with ICV guard mode and reasoning enabled. Guard mode can also be enabled when the database is created on the CLI (#_command_line_interface).

Terminology

This chapter may make more sense if you read this section on terminology a few times.

ICV, Integrity Constraint Validation

The process of checking whether some Stardog database is valid with respect to some integrity constraints. The result of ICV is a boolean value (true if valid, false if invalid) and, optionally, an `explanation of constraint violations`.

Schema, TBox

A schema (or "terminology box" a.k.a., TBox) is a set of statements that define the relationships between data elements, including property and class names, their relationships, etc. In practical terms, schema statements for a Stardog database are RDF Schema and OWL 2 terms, axioms, and definitions.

Data, ABox

All of the triples in a Stardog database that aren't part of the schema are part of the data (or "assertional box" a.k.a. ABox).

Integrity Constraint

A declarative expression of some rule or constraint which data must conform to in order to be valid. Integrity Constraints are typically domain and application specific. They can be expressed in OWL 2 (any legal syntax), SWRL rules, or (a restricted form of) SPARQL queries.

Constraints

Constraints that have been associated with a Stardog database and which are used to validate the data it contains. Each Stardog may optionally have one and only one set of constraints associated with it.

Closed World Assumption, Closed World Reasoning

Stardog ICV assumes a closed world with respect to data and constraints: that is, it assumes that all relevant data is known to it and included in a database to be validated. It interprets the meaning of Integrity Constraints in light of this assumption; if a constraint says a value `must` be present, the absence of that value is interpreted as a constraint violation and, hence, as invalid data.

Open World Assumption, Open World Reasoning

A legal OWL 2 inference may violate or satisfy an Integrity Constraint in Stardog. In other words, you get to have your cake (OWL as a constraint language) and eat it, too (OWL as modeling or inference language). This means that constraints are applied to a Stardog database `with respect to an OWL 2 profile`.

Monotonicity

OWL is a monotonic language: that means you can never `add` anything to a Stardog database that causes there to be `fewer` legal inferences. Or, put another way, the only way to decrease the number of legal inferences is to `delete` something.

Monotonicity interacts with ICV in the following ways:

1. Adding data to or removing it from a Stardog database may make it invalid.
2. Adding schema statements to or removing them from a Stardog database may make it invalid.
3. Adding new constraints to a Stardog database may make it invalid.
4. Deleting constraints from a Stardog database cannot make it invalid.

GRAPHQL QUERIES

NOTE Stardog Web Console does not support executing GraphQL queries.

Introduction

Stardog supports querying data stored (or mapped) in a Stardog database using GraphQL (http://graphql.org/) queries. You can load data into Stardog as usual and execute GraphQL queries without creating a GraphQL schema. You can also associate one or more GraphQL schemas (#_graphql_schemas) with a database and execute GraphQL queries against one of those schemas.

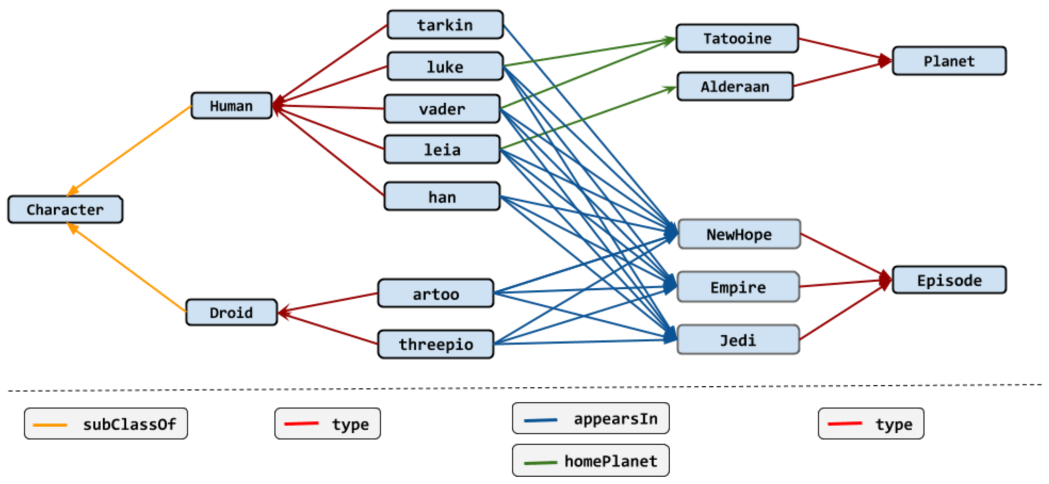
The following table shows the correspondence between RDF concepts and GraphQL:

RDF	GraphQL
Node	Object
Class	Type
Property	Field
Literal	Scalar

Execution of GraphQL queries in Stardog does not follow the procedural rules defined in the GraphQL spec (https://facebook.github.io/graphql/October2016/#sec-Executing-Selection-Sets). Instead Stardog translates GraphQL queries to SPARQL and then SPARQL results to GraphQL results based on the correspondences shown in the preceding table. Each RDF node represents a GraphQL object. Properties of the node are the fields of the object with the exception of `rdf:type` property which represents the type of the object. Literals in RDF are mapped to GraphQL scalars.

RDF	GraphQL
`xsd:integer`	`IntValue`
`xsd:float`	`FloatValue`
`xsd:string`	`StringValue`
`xsd:boolean`	`BooleanValue`
UNDEF	`NullValue`
IRI	`EnumValue`

In the following sections we will use a slightly modified version of the canonical GraphQL Star Wars example to explain how GraphQL queries work in Stardog. The following graph shows the core elements of the dataset and links between those nodes:



3. Subset of the Star Wars Graph

The full dataset in Turtle format is available in the examples repo (https://github.com/stardog-union/stardog-examples/blob/develop/examples/api/data/starwars.ttl).

Executing GraphQL Queries

GraphQL queries can be run via the CLI (/docs/5.3.6/man/data-add), the Java API (/docs/5.3.6/java/snarl/com/complexible/stardog/api/graphql/graphqlconnection) or the HTTP API (#_execute_graphql_query).

The GraphQL command can be executed by providing a query string:

```
$ stardog graphql starwars "{ Human { name }}"
```

or a file containing the query:

```
$ stardog graphql starwars query.file
```

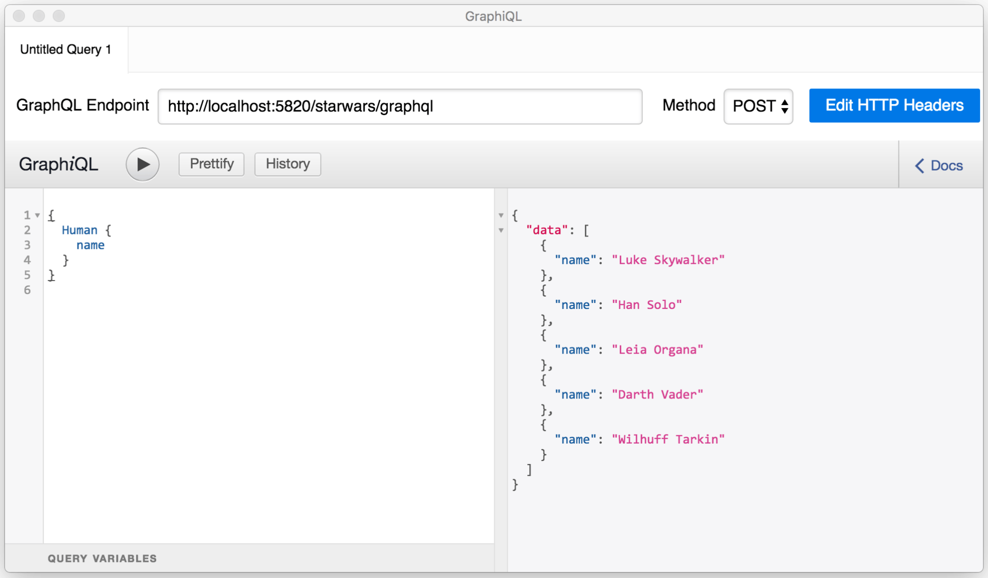
The `--reasoning` flag can be used with the CLI command to enable reasoning.

The HTTP command can be used to execute GraphQL queries. The endpoint for GraphQL queries is `http://HOST:PORT/{db}/graphql (http://HOST:PORT/{db}/graphql)`. The following command uses `curl` to execute a GraphQL query:

```
$ curl -G -vsku admin:admin --data-urlencode query="{ Human { name }}" localhost:5820/starwars/graphql
```

Reasoning can be enabled by setting a special variable `@reasoning` in the GraphQL query variables.

Any standard GraphQL client, like GraphiQL (https://github.com/graphql/graphiql), can be used with Stardog:



http://localhost:5820/starwars/graphql

```
{
  Human {
    name
  }
}
```

```
{
  "data": [
    {
      "name": "Luke Skywalker"
    },
    {
      "name": "Han Solo"
    },
    {
      "name": "Leia Organa"
    },
    {
      "name": "Darth Vader"
    },
    {
      "name": "Wilhuff Tarkin"
    }
  ]
}
```

NOTE Stardog by default uses HTTP basic access authentication (https://en.wikipedia.org/wiki/Basic_access_authentication#Client_side). In order to use GraphiQL with Stardog you either need to start the Stardog server with `--disable-security` option so it won't require credentials or set the HTTP header `Authorization` in the request. If the default credentials `admin/admin` are being used in non-production settings, the HTTP header `Authorization` may be set to the value `Basic YWRtaW46YWRtaW4=` in the GraphiQL UI. The `curl` example above can be used to see the correct value of the header for your credentials.

Fields and Selection Sets

A top-level element in GraphQL by default represents a type and will return all the nodes with that type. The fields in the query will return matching properties:

Query	Result

```
{
   Human {
     name
   }
}
```

```
{
  "data" : [ {
    "name" : "Luke Skywalker"
  }, {
    "name" : "Han Solo"
  }, {
    "name" : "Leia Organa"
  }, {
    "name" : "Darth Vader"
  }, {
    "name" : "Wilhuff Tarkin"
  } ]
}
```

Each field in the query is treated as a required property of the node (unless an `@optional` directive is used) so any node without corresponding properties will not appear in the results:

Query	Result

```
{
   Human {
     name
     homePlanet
   }
}
```

```
{
  "data" : [ {
    "name" : "Luke Skywalker",
    "homePlanet" : "Tatooine"
  }, {
    "name" : "Leia Organa",
    "homePlanet" : "Alderaan"
  }, {
    "name" : "Darth Vader",
    "homePlanet" : "Tatooine"
  } ]
}
```

If a node in the graph has multiple properties, then in the query results those results will be returned as an array:

Query	Result

```
{
   Droid {
     name
     friends
   }
}
```

```
{
  "data" : [ {
    "name" : "C-3PO",
    "friends" : [ "luke", "han", "leia", "artoo"
]
  }, {
    "name" : "R2-D2",
    "friends" : [ "luke", "han", "leia" ]
  } ]
}
```

Also note that Stardog does **not** enforce the GraphQL requirement that leaf fields must be scalars (https://facebook.github.io/graphql/October2016/#sec-Leaf-Field-Selections). In the previous example friends of a droid are objects but the query does not provide any fields. In those cases, the identifier of the node will be returned as a string.

Arguments

In GraphQL fields are, conceptually, functions which return values and may accept arguments (https://facebook.github.io/graphql/October2016/#sec-Language.Arguments) that alter their behavior. Arguments have no predefined semantics but the typical usage is for defining lookup values for fields. Stardog adopts this usage and treats arguments as filters for the query. The following query return only the node whose `id` field is `1000`:

Query	Result

```
{
  Human(id: 1000) {
    id
    name
    homePlanet
  }
}
```

```
{
  "data": {
    "name": "Luke Skywalker",
    "id": 1000,
    "homePlanet": "Tatooine"
  }
}
```

Arrays can be used to specify multiple values for a field in which case nodes matching any field will be returned:

Query

```
{
  Human(id: [1000, 1003]) {
    id
    name
    homePlanet
  }
}
```

Result

```
{
  "data": [
    {
      "name": "Luke Skywalker",
      "id": 1000,
      "homePlanet": "Tatooine"
    },
    {
      "name": "Leia Organa",
      "id": 1003,
      "homePlanet": "Alderaan"
    }
  ]
}
```

Reasoning

GraphQL queries by default only return results based on explicit nodes and edges in the graph. Reasoning may be enabled in the usual ways to run queries with inferred nodes and edges, e.g. to perform type inheritance. In the example graph, `Human` and `Droid` are defined as subclasses of the `Character` class. The following query will return no results without reasoning but when reasoning is enabled `Character` will act like a GraphQL interface (https://facebook.github.io/graphql/October2016/#sec-Interfaces) and the query will return both humans and droids:

Query

```
{
   Character {
     name
   }
}
```

Query Variables

```
{
   "@reasoning": true
}
```

Result

```
{
  "data" : [ {
    "name" : "Luke Skywalker"
  }, {
    "name" : "Han Solo"
  }, {
    "name" : "Leia Organa"
  }, {
    "name" : "C-3PO"
  }, {
    "name" : "R2-D2"
  }, {
    "name" : "Darth Vader"
  }, {
    "name" : "Wilhuff Tarkin"
  } ]
}
```

Fragments

Stardog supports GraphQL fragments (both inline (https://facebook.github.io/graphql/October2016/#sec-Inline-Fragments) or via fragment definitions (https://facebook.github.io/graphql/October2016/#sec-Language.Fragments)). This query shows how fragments can be combined with reasoning to select different fields for subtypes:

Query	Result

```
{
  Character {
    name
    ... on Human {
      friends
    }
    ... on Droid {
      primaryFunction
    }
  }
}
```

```
{
  "data" : [ {
    "name" : "Luke Skywalker",
    "friends" : [ "threepio", "artoo", "han",
"leia" ]
  }, {
    "name" : "Han Solo",
    "friends" : [ "leia", "artoo", "luke" ]
  }, {
    "name" : "Leia Organa",
    "friends" : [ "threepio", "artoo", "luke",
"han" ]
  }, {
    "name" : "C-3PO",
    "primaryFunction" : "Protocol"
  }, {
    "name" : "R2-D2",
    "primaryFunction" : "Astromech"
  }, {
    "name" : "Darth Vader",
    "friends" : "tarkin"
  }, {
    "name" : "Wilhuff Tarkin",
    "friends" : "vader"
  } ]
}
```

Aliases

By default, the key in the response object will use the field name queried. However, you can define a different name by specifying an alias (https://facebook.github.io/graphql/October2016/#sec-Field-Alias). The following query renames both of the fields in the query:

Query	Result

```
{
  Human {
    fullName: name
    bornIn: homePlanet
  }
}
```

```
{
  "data": [
    {
      "fullName": "Luke Skywalker",
      "bornIn": "Tatooine"
    },
    {
      "fullName": "Leia Organa",
      "bornIn": "Alderaan"
    },
    {
      "fullName": "Darth Vader",
      "bornIn": "Tatooine"
    }
  ]
}
```

Variables

A GraphQL query can be parameterized with variables (https://facebook.github.io/graphql/October2016/#sec-Language.Variables) which must be defined at the top of an operation. Variables are in scope throughout the execution of that operation. A value should be provided for GraphQL variables before execution or an error will occur. The following query will return a single result when executed with the input `{"id": 1000}`:

Query	Result

```
query getHuman($id: Integer) {
  Human(id: $id) {
    id
    name
  }
}
```

```
{
  "data": {
    "name": "Luke Skywalker",
    "id": 1000
  }
}
```

Query Variables

```
{
   "id": 1000
}
```

Ordering Results

The results of GraphQL queries may be randomly ordered. A special argument `orderBy` can be used at the top level to specify which field to use for ordering the results. The following query uses the values of the `name` field for ordering the results:

Query

```
{
  Human(orderBy: name) {
    name
  }
}
```

Result

```
{
  "data": [
    { "name": "Darth Vader" },
    { "name": "Han Solo" },
    { "name": "Leia Organa" },
    { "name": "Luke Skywalker" },
    { "name": "Wilhuff Tarkin" }
  ]
}
```

The results are ordered in ascending order by default. We can sort results in descending order as follows:

Query

```
{
  Human(orderBy: {field: name, desc: true}) {
    name
  }
}
```

Result

```
{
  "data": [
    { "name": "Wilhuff Tarkin" },
    { "name": "Luke Skywalker" },
    { "name": "Leia Organa" },
    { "name": "Han Solo" },
    { "name": "Darth Vader" }
  ]
}
```

Multiple ordering criteria can be used:

Query | **Result**

```
{
  Human(orderBy: [homePlanet,
                  {field: name, desc: false}]) {
    name
    homePlanet @optional
  }
}
```

```
{
  "data": [
    { "name": "Wilhuff Tarkin" },
    { "name": "Han Solo" },
    { "name": "Leia Organa",
      "homePlanet": "Alderaan" },
    { "name": "Luke Skywalker",
      "homePlanet": "Tatooine" },
    { "name": "Darth Vader",
      "homePlanet": "Tatooine" }
  ]
}
```

We first use the `homePlanet` field for ordering and the results with no home planet come up first. If two results have the same value for the first order criteria, e.g. `Luke Skywalker` and `Darth Vader`, then the second criteria is used for ordering.

Paging Results

Paging through the GraphQL results is accomplished with `first` and `skip` arguments used at the top level. The following query returns the first three results:

Query

```
{
  Human(orderBy: name, first: 3) {
    name
  }
}
```

Result

```
{
  "data": [
    { "name": "Darth Vader" },
    { "name": "Han Solo" },
    { "name": "Leia Organa" }
  ]
}
```

The following query skips the first result and returns the next two results:

Query

```
{
  Human(orderBy: name, skip:1, first: 2) {
    name
  }
}
```

Result

```
{
  "data": [
    { "name": "Han Solo" },
    { "name": "Leia Organa" }
  ]
}
```

Directives

Directives provide a way to describe alternate runtime execution and type validation behavior in GraphQL. The spec defines two built-in directives (https://facebook.github.io/graphql/October2016/#sec-Type-System.Directives): `@skip` and `@include`. Stardog supports both directives and introduces several others.

`@skip(if: EXPR)`

The `skip` directive includes a field value in the result conditionally. If the provided expression evaluates to `true` the field will not be included. Stardog allows arbitrary SPARQL expressions (https://www.w3.org/TR/sparql11-query/#expressions) to be used as the conditions. Any of the supported SPARQL Query Functions (#_sparql_query_functions) can be used in these expressions. The expression can refer to any field in the same selection set and is not limited to the field directive is attached to. The following query returns the name field only if the name does not start with the letter `L`:

Query

```
{
  Human {
    id
    name @skip(if: "strstarts($name, 'L')")
  }
}
```

Result

```
{
  "data": [
    {
      "id": 1000
    },
    {
      "name": "Han Solo",
      "id": 1002
    },
    {
      "id": 1003
    },
    {
      "name": "Darth Vader",
      "id": 1001
    },
    {
      "name": "Wilhuff Tarkin",
      "id": 1004
    }
  ]
}
```

`@include(if: EXPR)`

The `@include` directive works negation of the `@skip` directive; that is, the field is included only if the expression evaluates to `true`. We can use variables (#_variables) inside the conditions, too. The following example executed with input `{"withFriends": false}` will not include friends in the results:

Query

```
query HumanAndFriends($withFriends: Boolean) {
  Human @type {
    name
    friends @include(if: $withFriends) {
      name
    }
  }
}
```

Query Variables

```
{
   "withFriends": false
}
```

Result

```
{
  "data": [
    {
      "name": "Luke Skywalker"
    },
    {
      "name": "Han Solo"
    },
    {
      "name": "Leia Organa"
    },
    {
      "name": "Darth Vader"
    },
    {
      "name": "Wilhuff Tarkin"
    }
  ]
}
```

`@filter(if: EXPR)`

The `@filter` directive looks similar to `@skip` and `@include` but filters the whole object instead of just a single field. In that regard it works more like arguments (#_arguments) but arbitrary expressions can be used to select specific nodes. The next query returns all humans whose `id` is less than `1003`:

Query	Result

```
{
  Human {
    name
    id @filter(if: "$id < 1003")
  }
}
```

```
{
  "data": [
    {
      "name": "Luke Skywalker",
      "id": 1000
    },
    {
      "name": "Han Solo",
      "id": 1002
    },
    {
      "name": "Darth Vader",
      "id": 1001
    }
  ]
}
```

Unlike the previous two filters it doesn't matter which field the `@filter` directive is syntactically adjacent to since it applies to the whole selection set.

`@optional`

Stardog treats every field as required by default and will not return any nodes if they don't have a matching value for the fields in the selection set. The `@optional` directive can be used to mark a field as optional. The following query returns the home planets for humans if it exists but skips that field if it doesn't:

Query

```
{
  Human {
    name
    homePlanet @optional
  }
}
```

Result

```
{
  "data": [
    {
      "name": "Luke Skywalker",
      "homePlanet": "Tatooine"
    },
    {
      "name": "Han Solo"
    },
    {
      "name": "Leia Organa",
      "homePlanet": "Alderaan"
    },
    {
      "name": "Darth Vader",
      "homePlanet": "Tatooine"
    },
    {
      "name": "Wilhuff Tarkin"
    }
  ]
}
```

`@type`

By default every field in the GraphQL query other than the topmost field represents a property in the graph. Sometimes we might want to filter some nodes based on their types; that is, based on the values of the special built-in property `rdf:type`. Stardog provides a directive as a shortcut for this purpose. The following query returns only the droid friends of humans because the `Droid` field is marked with the `@type` directive:

Query	Result

```
{
  Human {
    name
    friends {
      Droid @type
      name
    }
  }
}
```

```
{
  "data": [
    {
      "name": "Luke Skywalker",
      "friends": [
        {
          "name": "R2-D2"
        },
        {
          "name": "C-3PO"
        }
      ]
    },
    {
      "name": "Han Solo",
      "friends": {
        "name": "R2-D2"
      }
    },
    {
      "name": "Leia Organa",
      "friends": [
        {
          "name": "R2-D2"
        },
        {
          "name": "C-3PO"
        }
      ]
    }
  ]
}
}
```

`@bind(to: EXPR)`

Fields bind to properties in the graph but it is also possible to have fields with computed values. When `@bind` directive is used for a field the value of that field will be compared by evaluating the given SPARQL expression (https://www.w3.org/TR/sparql11-query/#expressions). The following example splits the name field on a space to compute `firstName` and `lastName` fields:

Query

```
{
  Human {
    name @hide
    firstName @bind(to: "strbefore($name, ' ')")
    lastName @bind(to: "strafter($name, ' ')")
  }
}
```

Result

```
{
  "data": [
    {
      "firstName": "Luke",
      "lastName": "Skywalker"
    },
    {
      "firstName": "Han",
      "lastName": "Solo"
    },
    {
      "firstName": "Leia",
      "lastName": "Organa"
    },
    {
      "firstName": "Darth",
      "lastName": "Vader"
    },
    {
      "firstName": "Wilhuff",
      "lastName": "Tarkin"
    }
  ]
}
```

`@hide`

Query results can be flattened using the `@hide` directive. For example, in our data characters are linked to episode instances that have an `index` property. The following query retrieves the episode indexes but, by hiding the intermediate episode instances, humans are directly linked to the episode index:

Query	Result

```
{
  Human {
    name
    appearsIn @hide {
      episodes: index
    }
  }
}
```
	```
	{
	  "data": [
	    {
	      "name": "Luke Skywalker",
	      "episodes": [4, 5, 6]
	    },
	    {
	      "name": "Han Solo",
	      "episodes": [4, 5, 6]
	    },
	    {
	      "name": "Leia Organa",
	      "episodes": [4, 5, 6]
	    },
	    {
	      "name": "Darth Vader",
	      "episodes": [4, 5, 6]
	    },
	    {
	      "name": "Wilhuff Tarkin",
	      "episodes": 4
	    }
	  ]
	}
	```

Namespaces

RDF uses IRIs as identifiers whereas in GraphQL we have simple names as identifiers. The examples so far use a single default namespace where names in GraphQL are treated as local names in that namespace. If a Stardog graph uses multiple namespaces, then it is possible to use them in GraphQL queries in several different ways.

If there are stored namespaces (#_namespacing) in the database then the associated prefixes can be used in the queries. For example, suppose we have the prefix `foaf` associated with the namespace `http://xmlns.com/foaf/0.1/ (http://xmlns.com/foaf/0.1/)` in the database. In SPARQL the prefixed name `foaf:Person` would be used for the IRI `http://xmlns.com/foaf/0.1/Person (http://xmlns.com/foaf/0.1/Person)`. In GraphQL, the `:` character cannot be used in field names so instead Stardog uses the `_` character: the prefixed name here would be `foaf_Person`. The query using FOAF namespace would look like this:

```
{
  foaf_Person {
    foaf_name
    foaf_mbox
  }
}
```

If the namespace is not stored in the database an inline prefix definition can be provided with the `@prefix` directive:

```
query withPrefixes @prefix(foaf: "http://xmlns.com/foaf/0.1/") {
  foaf_Person {
    foaf_name
    foaf_mbox
  }
}
```

NOTE Sometimes field names might use the underscore character and it might not indicate a prefix. To differentiate two cases Stardog looks at the prefix before the underscore and checks if it is defined in the query or if it is stored in the database. In some cases the IRI local name might be using characters like `-` that is not allowed in GraphQL names. In those cases an alias can be defined to map a field name to an IRI. These aliases are defined in a `@config` directive at the query level as follows:

```
query withAliases @config(alias: {myType: "http://example.com/my-type",
                                  myProp: "http://example.com/my-prop"})
{
  myType {
    myProp
  }
}
```

Named Graphs

GraphQL queries by default are evaluated over the union of all graphs stored in the Stardog database. It is possible to limit the scope of the query to one or more specific named graphs. Suppose we partition the Star Wars dataset by moving instances of each type to a different named graph using the following SPARQL update query:

```
DELETE { ?s ?p ?o }
INSERT { GRAPH ?type { ?s ?p ?o } }
WHERE { ?s a ?type ; ?p ?o }
```

The following queries (with reasoning) will return 5 humans, 2 droids and all 7 characters respectively:

Query

```
query onlyHumanGraph @config(graph: Human) {
  Character {
    name
  }
}
```

Result

```
{
  "data": [
    { "name": "Luke Skywalker" },
    { "name": "Han Solo" },
    { "name": "Leia Organa" },
    { "name": "Darth Vader" },
    { "name": "Wilhuff Tarkin" }
  ]
}
```

Query

```
query onlyDroidGraph @config(graph: Droid) {
  Character {
    name
  }
}
```

Result

```
{
  "data": [
    { "name": "C-3PO" },
    { "name": "R2-D2" }
  ]
}
```

Query

```
query bothGraphs @config(graph: [Human, Droid])
{
  Character {
    name
  }
}
```

Result

```
{
  "data": [
    { "name": "Luke Skywalker" },
    { "name": "Han Solo" },
    { "name": "Leia Organa" },
    { "name": "C-3PO" },
    { "name": "R2-D2" },
    { "name": "Darth Vader" },
    { "name": "Wilhuff Tarkin" }
  ]
}
```

GraphQL Schemas

GraphQL is a strongly-typed language where the fields used in a query should conform to the type definitions in a GraphQL schema (http://graphql.org/learn/schema/). By default, Stardog relaxes this restriction and allows queries to be executed without an explicit schema. However, if desired, one or more GraphQL schemas can be added to the database and used during query execution. The benefits of using an explicit schema are as follows:

- Queries will be validated with strict typing
- Default translation of RDF values to GraphQL values can be overridden
- Only the parts of the graph defined in the schema will be exposed to the user

Here is an example schema that can be used with the Star Wars dataset:

```
schema {
    query: QueryType
}

type QueryType {
    Character: Character
    Human(id: Int, first: Int, skip: Int, orderBy: ID): Human
    Droid(id: Int): Droid
}

interface Character {
    id: Int!
    name: String!
    friends(id: Int): [Character]
    appearsIn: [Episode]
}

type Human implements Character {
    iri: ID!
    id: Int!
    name: String!
    friends(id: Int): [Character]
    appearsIn: [Episode]
}

type Droid implements Character {
    id: Int!
    name: String!
    friends(id: Int): [Character]
    appearsIn: [Episode]
    primaryFunction: String
}

type Episode {
  index: Int!
  name: String!
}
```

Each GraphQL schema defines a query type which specifies the top-level field that can be used in a query. In Stardog the query type is simply an enumeration of classes in the database that we want to expose in queries. For example, the schema defines the `Episode` type but does not list it under `QueryType` which means you cannot query for episodes directly.

Note that, without a schema each top-level type can have various built-in arguments like `first` or `skip`. In this schema we chose to define them for the `Human` type but not for others. This means a query like `{ Droid(first: 1) { name } }` will be invalid with respect to this schema and rejected even though it is valid if executed without a schema.

This schema can be added to the database by giving it a name:

```
$ stardog graphql schema --add characters starwars characters.graphqls
Added schema characters
```

We can then execute the query by specifying the schema name along with the query:

```
$ stardog graphql --schema characters starwars "{ Human { name friends { name } } }"
```

When a schema is specified for a query it gets added to the query parameters using a special variable named `@schema`. When using the HTTP API directly this variable can be set to choose the schema for a query by sending the query variable `{"schema": "characters" }`.

Query

```
{
  Human(id: 1004) {
    name
    friends {
      name
    }
  }
}
```

Query Variables

```
{
   "@schema": "characters"
}
```

Result

```
{
  "data": [
    {
      "name": "Wilhuff Tarkin",
      "friends": [
        {
          "name": "Darth Vader"
        }
      ]
    }
  ]
}
```

Note that the friends field in the result is an array value due to the corresponding definition in the schema. This query executed with a schema would return the single object value for the field.

An important point about schemas is that the types defined in the schema do not filter the query results. For example, we can define a much simpler `humans` schema against the Star Wars dataset:

```
schema {
  query: QueryType
}

type QueryType {
  Human(id: [Int]): Human
}

type Human {
  id: Int!
  name: String!
  friends: [Human]
}
```

This query allows only `Human` instances to be queried at the top level and declares that the friend of each `Human` is also a `Human`. This schema definition is incompatible with the data since humans have droid friends. Stardog does not check if the schema is correct with respect to the data and will not enforce type restrictions in the results. So if we ask for the friends of a human, then the droids will also be returned in the results:

Query

```
{
  Human(id: 1000) {
    name
    friends {
      name
    }
  }
}
```

Query Variables

```
{ "@schema": "humans" }
```

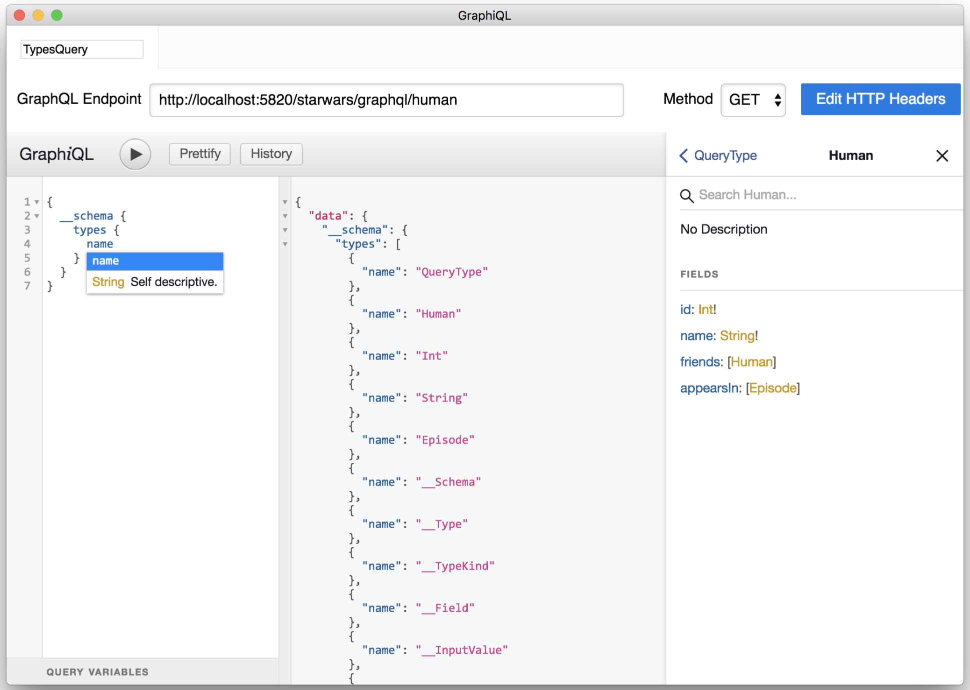
Result

```
{
  "data": [
    {
      "name": "Luke Skywalker",
      "friends": [
        { "name": "Han Solo" },
        { "name": "Leia Organa" },
        { "name": "C-3PO" },
        { "name": "R2-D2" }
      ]
    }
  ]
}
```

Introspection

Stardog supports GraphQL introspection (http://graphql.org/learn/introspection/) which means GraphQL tooling works out of the box with Stardog. Introspection allows schema queries to be discovered, exposed, and executed and to retrieve information about the types and fields defined in a schema. This feature is used in GraphQL tools to support features like autocompletion, query validation, etc.

Stardog supports introspection queries for the GraphQL schemas (#_graphql_schemas) registered in the system. There is a separate dedicated endpoint for each schema registered in the system in the form `http://HOST:PORT/{db}/graphql/{schema}` (http://HOST:PORT/{db}/graphql/{schema}). The introspection queries executed against this endpoint will be answered using the corresponding schema.



GraphQL Endpoint http://localhost:5820/starwars/graphql/human

Method GET

```
{
  __schema {
    types {
      name
    }
  }
}
```

```
{
  "data": {
    "__schema": {
      "types": [
        {
          "name": "QueryType"
        },
        {
          "name": "Human"
        },
        {
          "name": "Int"
        },
        {
          "name": "String"
        },
        {
          "name": "Episode"
        },
        {
          "name": "__Schema"
        },
        {
          "name": "__Type"
        },
        {
          "name": "__TypeKind"
        },
        {
          "name": "__Field"
        },
        {
          "name": "__InputValue"
        },
        {
```

name
String Self descriptive.

Search Human...

No Description

FIELDS

id: Int!
name: String!
friends: [Human]
appearsIn: [Episode]

Introspection queries are not supported by the default GraphQL endpoint as there is no dedicated schema associated with the default endpoint.

Implementation

Stardog translates GraphQL queries to SPARQL and SPARQL results to JSON. The CLI command `graphql explain` can be used to see the generated SPARQL query and the low-level query plan created for the SPARQL query which is useful for debugging correctness and performance issues:

```
$ stardog graphql explain starwars  "{
   Human(id: 1000) {
     name
     knows: friends {
       name
     }
   }
}"
SPARQL:
SELECT *
FROM <tag:stardog:api:context:all>
{
?0 rdf:type :Human .
?0 :id "1000"^^xsd:integer .
?0 :name ?1 .
?0 :friends ?2 .
?2 :name ?3 .
}

FIELDS:
0 -> {1=name, 2=knows}
2 -> {3=name}

PLAN:
prefix : <http://api.stardog.com/>

From all
Projection(?0, ?1, ?2, ?3) [#3]
`─ MergeJoin(?2) [#3]
   +─ Sort(?2) [#3]
   │  `─ NaryJoin(?0) [#3]
   │     +─ Scan[POSC](?0, :id, "1000"^^<http://www.w3.org/2001/XMLSchema#integer>) [#1]
   │     +─ Scan[POSC](?0, <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>, :Human) [#5]
   │     +─ Scan[PSOC](?0, :name, ?1) [#10]
   │     `─ Scan[PSOC](?0, :friends, ?2) [#20]
   `─ Scan[PSOC](?2, :name, ?3) [#10]
```

The variables in the SPARQL query will be mapped to objects and field values in the JSON results. The binding for variable `0` will be the root object. The `FIELDS` output show that `0` is linked to `1` via the `name` field and linked to `2` via the `knows` field (note that `knows` is an alias and in the actual query we have the pattern `?0 :friends ?2`).

The GraphQL query plans can also be retrieved by setting the special query variable `@explain` to true when executing a query.

PATH QUERIES

Stardog extends SPARQL to find paths between nodes in the RDF graph, which we call *path queries*. They are similar to SPARQL 1.1 property paths (https://www.w3.org/TR/sparql11-query/#propertypaths) which traverse an RDF graph and find pairs of nodes connected via a complex path of edges. But SPARQL property paths only return the start and end nodes of a path and do not allow variables in property path expressions. Stardog path queries return *all* intermediate nodes on each path—that is, they return a path from start to end—and allow arbitrary SPARQL graph patterns to be used in the query.

TIP

- A Path of Our Own (https://www.stardog.com/blog/a-path-of-our-own/)
- GraphQL and Paths (https://www.stardog.com/blog/graphql-and-paths/)

Path Query Syntax

We add path queries as a new top-level query form, i.e. separate from `SELECT`, `CONSTRUCT` or other query types. The syntax is as follows:

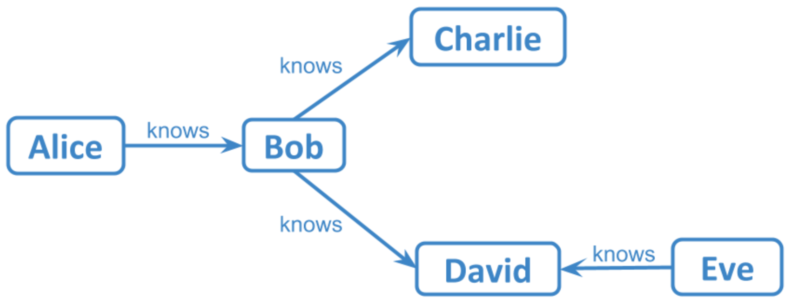
```
PATHS [SHORTEST|ALL] [CYCLIC] [<DATASET>]
START ?s [ = <IRI> | <GRAPH PATTERN> ] END ?e [ = <IRI> | <GRAPH PATTERN> ]
VIA <GRAPH PATTERN> | <VAR> | <PATH>
[MAX LENGTH <int>]
[ORDER BY <condition>]
[OFFSET <int>]
[LIMIT <int>]
```

The graph pattern in the `VIA` clause *must* bind both `?s` and `?e` variables.

Next we informally present examples of common path queries and finally the formal Path Query Evaluation Semantics (#_path_query_evaluation_semantics).

Shortest Paths

Suppose we have a simple social network where people are connected via different relationships:



4. Simple Graph

If we want to find all the people Alice is connected to and how she is connected to them we can use the following path query:

```
PATHS START ?x = :Alice END ?y VIA ?p
```

We specify a start node for the path query but the end node is unrestricted. So all paths starting from Alice will be returned. Note that we use the shortcut `VIA ?p` instead of a graph pattern to match each edge in the path. This is a syntactic sugar for `VIA { ?s ?p ?e }`. Similarly we could use a predicate, e.g. `VIA :knows` or a property path expression, e.g. `VIA :knows | :worksWith`.

This query is effectively equivalent to the SPARQL property path `:Alice :knows+ ?y`, but the results will include the nodes in the path(s). The path query results are printed in a tabular format by default:

```
+----------+------------+----------+
|    x     |     p      |    y     |
+----------+------------+----------+
| :Alice   | :knows     | :Bob     |
|          |            |          |
| :Alice   | :knows     | :Bob     |
| :Bob     | :knows     | :David   |
|          |            |          |
| :Alice   | :knows     | :Bob     |
| :Bob     | :worksWith | :Charlie |
|          |            |          |
| :Alice   | :knows     | :Bob     |
| :Bob     | :worksWith | :Charlie |
| :Charlie | :parentOf  | :Eve     |
+----------+------------+----------+

Query returned 4 paths in 00:00:00.055
```

Each row of the result table shows one edge and adjacent edges on a path are printed on subsequent rows of the table. Multiple paths in the results are separated by an empty row. We can change the output format to `text` which serializes the results in a property graph like syntax:

```
$ stardog query -f text exampleDB "PATHS START ?x = :Alice END ?y VIA ?p"
(:Alice)-[p=:knows]->(:Bob)

(:Alice)-[p=:knows]->(:Bob)-[p=:knows]->(:David)

(:Alice)-[p=:knows]->(:Bob)-[p=:worksWith]->(:Charlie)

(:Alice)-[p=:knows]->(:Bob)-[p=:worksWith]->(:Charlie)-[p=:parentOf]->(:Eve)

Query returned 4 paths in 00:00:00.047
```

Execution happens by recursively evaluating the graph pattern in the query and replacing the start variable with the binding of the end variable in the previous execution. If the query specifies a start node, that value is used for the first evaluation of the graph pattern. If the query specifies an end node, which our example doesn't, execution stops when we reach the end node. Only simple cycles, i.e. paths where the start and the end nodes coincide, are allowed in the results.

NOTE The Stardog optimizer may choose to traverse paths backwards, i.e. from the end node to the start, for performance reasons but it does not affect the results.

We can specify the end node in the query and restrict the kind of patterns in paths to a specific property as in the next example that queries how `Alice` is connected to `David` via `knows` relationships:

```
PATHS START ?x = :Alice END ?y = :David VIA :knows
```

This query would return a single path with two edges:

```
+--------+--------+
|   x    |   y    |
+--------+--------+
| :Alice | :Bob   |
| :Bob   | :David |
+--------+--------+
```

Complex Paths

Graph patterns inside the path queries can be arbitrarily complex. Suppose, we want to find undirected paths between `Alice` and `David` in this graph. Then we can make the graph pattern to match both outgoing and incoming edges:

```
$ stardog query exampleDB "PATHS START ?x = :Alice END ?y = :David VIA ^:knows | :knows"
+--------+--------+
|   x    |   y    |
+--------+--------+
| :Alice | :Bob   |
| :Bob   | :David |
+--------+--------+
```

Sometimes a relationship between two nodes might be implicit and there might not be an explicit link between those two nodes in the RDF graph. Consider the following set of triples that show some movies and actors who starred in those movies:

```
:Apollo_13 a :Film ; :starring :Kevin_Bacon , :Gary_Sinise .

:Spy_Game a :Film ; :starring :Brad_Pitt , :Robert_Redford .

:Sleepers a :Film ; :starring :Kevin_Bacon , :Brad_Pitt .

:A_Few_Good_Men a :Film ; :starring :Kevin_Bacon , :Tom_Cruise .

:Lions_for_Lambs a :Film ; :starring :Robert_Redford , :Tom_Cruise .

:Captain_America a :Film ; :starring :Gary_Sinise , :Robert_Redford .
```

There is an implicit relationship between actors based on the movies they appeared together. We can use a basic graph pattern with multiple triple patterns in the path query to extract this information:

```
PATHS START ?x = :Kevin_Bacon END ?y = :Robert_Redford
VIA { ?movie a :Film ; :starring ?x , ?y  }
```

This query executed against the above set of triples would return three paths:

```
+--------------+------------------+-----------------+
|      x       |      movie       |        y        |
+--------------+------------------+-----------------+
| :Kevin_Bacon | :Apollo_13       | :Gary_Sinise    |
| :Gary_Sinise | :Captain_America | :Robert_Redford |
|              |                  |                 |
| :Kevin_Bacon | :Sleepers        | :Brad_Pitt      |
| :Brad_Pitt   | :Spy_Game        | :Robert_Redford |
|              |                  |                 |
| :Kevin_Bacon | :A_Few_Good_Men  | :Tom_Cruise     |
| :Tom_Cruise  | :Lions_for_Lambs | :Robert_Redford |
+--------------+------------------+-----------------+
```

If the movie is irrelevant, then a more concise version can be used:

```
PATHS START ?x = :Kevin_Bacon END ?y = :Robert_Redford VIA  ^:starring/:starring
```

All Paths

Path queries return only shortest paths by default. We can use the `ALL` keyword in the query to retrieve all paths between two nodes. For example, the query above returned only one path between `Alice` and `David`. We can get all paths as follows:

```
$ stardog query exampleDB "PATHS ALL START ?x = :Alice END ?y = :David
VIA { {?x ?p ?y} UNION {?y ?p ?x} }"
+----------+------------+----------+
|    x     |     p      |    y     |
+----------+------------+----------+
| :Alice   | :knows     | :Bob     |
| :Bob     | :knows     | :David   |
|          |            |          |
| :Alice   | :knows     | :Bob     |
| :Bob     | :worksWith | :Charlie |
| :Charlie | :parentOf  | :Eve     |
| :Eve     | :knows     | :David   |
+----------+------------+----------+
```

CAUTION The `ALL` qualifier can dramatically increase the number of paths so use with caution.

Cyclic Paths

There's a special keyword `CYCLIC` to specifically query for cyclic paths in the data. For example, there might be a `dependsOn` relationship in the database and we might want to query for cyclic dependencies:

```
PATHS CYCLIC START ?start END ?end VIA :dependsOn
```

Again, arbitrary cycles in the paths are not allowed to ensure a finite number of results.

Limiting Paths

In a highly connected graph the number of possible paths between two nodes can be impractically high. There are two different ways we can limit the results of path queries. The first possibility is to use the `LIMIT` keyword just like in other query types. We can ask for at most 2 paths starting from `Alice` as follows:

```
PATHS START ?x = :Alice END ?y VIA ?p LIMIT 2
```

This query would return 2 results as expected :

```
+----------+------------+----------+
|    x     |     p      |    y     |
+----------+------------+----------+
| :Alice   | :knows     | :Bob     |
|          |            |          |
| :Alice   | :knows     | :Bob     |
| :Bob     | :knows     | :David   |
+----------+------------+----------+
```

Note that, the path from `Alice` to `Charlie` is not included in this result even though it is not any longer than the path between `Alice` and `David`. This is because with `LIMIT` the query will stop producing results as soon as the maximum number of paths are returned.

The other alternative for limiting the results is by specifying the maximum length of paths that can be returned. The following query shows how to query for paths thar are at most 2-edge long:

```
PATHS START ?x = :Alice END ?y VIA ?p MAX LENGTH 2
```

This time we will get 3 results:

```
+----------+------------+----------+
|    x     |     p      |    y     |
+----------+------------+----------+
| :Alice   | :knows     | :Bob     |
|          |            |          |
| :Alice   | :knows     | :Bob     |
| :Bob     | :knows     | :David   |
|          |            |          |
| :Alice   | :knows     | :Bob     |
| :Bob     | :worksWith | :Charlie |
+----------+------------+----------+
```

It is also possible to use both `LIMIT` and `MAX LENGTH` keywords in a single query.

Path Queries With Start and End Patterns

In all examples presented so far the start and end variables were either free variables or bound to a single IRI. This is insufficient for navigating paths which must begin at multiple nodes satisfying certain conditions and terminate at nodes satisfying some other conditions. Assume the movie and actor data above is extended with information about the date of birth of each actor:

```
:Kevin_Bacon :birthdate "1958-07-08"^^xsd:date

:Gary_Sinise :birthdate "1957-03-17"^^xsd:date

:Brad_Pitt :birthdate "1963-12-18"^^xsd:date

:Robert_Redford :birthdate "1936-08-18"^^xsd:date

:Tom_Cruise :birthdate "1962-07-03"^^xsd:date
```

Now, having only variables and constants as valid path start and end expressions would make it hard to write a query to find all connections between Kevin Bacon and actors over 80 years old. The following attempt, for example, won't match any data:

```
PATHS START ?x = :Kevin_Bacon END ?y VIA {
  ?movie a :Film ; :starring ?x , ?y .
  ?y :birthdate ?date .
  FILTER (year(?date) - year(now()) >= 80)
}
```

The problem is that the age filter is applied at each recursive step, i.e. the query is looking for paths where every intermediate actor is over 80, but none of those co-starred with Kevin Bacon (in our toy dataset). Instead we need a query which checks the condition only at candidate end nodes:

```
PATHS START ?x = :Kevin_Bacon
END ?y { ?y :birthdate ?date .
         FILTER (year(?date) - year(now()) >= 80) }
VIA ^:starring/:starring
```

This query will return the expected results along with the date of birth for end nodes:

```
+------------------+---------------------+------------------------+
|        x         |          y          |          date          |
+------------------+---------------------+------------------------+
| test:Kevin_Bacon | test:Gary_Sinise    |                        |
| test:Gary_Sinise | test:Robert_Redford | "1936-08-18"^^xsd:date |
|                  |                     |                        |
| test:Kevin_Bacon | test:Brad_Pitt      |                        |
| test:Brad_Pitt   | test:Robert_Redford | "1936-08-18"^^xsd:date |
|                  |                     |                        |
| test:Kevin_Bacon | test:Tom_Cruise     |                        |
| test:Tom_Cruise  | test:Robert_Redford | "1936-08-18"^^xsd:date |
+------------------+---------------------+------------------------+
```

Path Queries With Reasoning

As other kinds of queries, path queries can be evaluated with reasoning. If reasoning is enabled, a path query will return paths in the *inferred* graph, i.e. each edge corresponds to a relationship between the nodes which is *inferred* from the data based on the schema.

Consider the following example:

```
:Arlington :partOf :DCArea .

:DCArea :locatedIn :EastCoast .

:EastCoast :partOf :US .

:US :locatedIn :NorthAmerica .
```

Adding the following rule (or an equivalent OWL sub-property chain axiom) infers `:partOf` edges based on compositions of `:partOf` and `:locatedIn` edges:

```
IF
  { ?x :partOf ?y . ?y :locatedIn ?z }
THEN
  { ?x :partOf ?z }
```

Now the following path query will find the *inferred* path from `:Arlington` to `:NorthAmerica` via `:DCArea` and `:US`:

```
PATHS START ?x = :Arlington END ?y = :NorthAmerica VIA {
  ?x :partOf ?y
}
```

NOTE This feature should be used with care. There may be a lot more paths than one expects. Also keep in mind that some patterns are particularly expensive with reasoning, e.g. triple patterns with the unbound predicate variable or with a variable in the object position of `rdf:type`.

Path Query Evaluation Semantics

Given a pair of variable names `s` and `e` a *path* is a sequence of SPARQL solutions `S[1], …, S[n]` s.t. `S[i](t) = S[i-1](s)` for `i` from `2` to `n`. We call the `S[0](s)` and `S[n](t)` values the *start* and *end* nodes of the path, resp. Each solution in the sequence is called an *edge*.

The evaluation semantics of path queries is based on the following recursive extension of SPARQL solution:

```
(1) Solution := { (V -> Value)* }    // solution: mapping from variables to values (as in SPARQL)
(2) Value := RDF-Term                // an RDF term is a value (as in SPARQL)
(3) Value := Solution                // a solution is a value (extension)
(4) Value := [ Value* ]              // an array of values is a value (extension)
```

Informally such extensions allow us to represent each path as a single solution where a distinguished variable (in the sequel called *path variable*) is mapped to an ordered array of solutions representing edges.

We first consider simple path queries for `ALL` paths with only variables after the `START` and `END` keywords, i.e. queries of the form `PQ(s, e, p, P)`, where `s` and `e` are start and end variable names, `p` is a path variable name, and `P` is a SPARQL graph pattern. Given a dataset `D` with the active graph `G`, abbreviated as `D(G)`, we define `eval(PQ(s, e, P), D(G))` as a set of all such (extended) solutions `S` that:

```
(1) S(p) is a path Sp[1] ... Sp[n] w.r.t. s and e
(2) Sp(1) is in eval(P, D(G))
(3) Sp[i] is a solution to eval(sub(P, s, Sp[i-1](e), D(G)) for i = 2 ... n
(4) S(s) = Sp[1](s)
(5) S(e) = Sp[n](e)
(6) All terms which s and e bind to in all Sp[i] are unique except that Sp[1](s) could be equal to Sp[n](e)
```

where `sub(P, var, t)` is a graph pattern obtained by substituting the variable `var` by the fixed RDF term `t`.

Informally conditions (2) and (3) state that each edge in a path is obtained by evaluating the path pattern with the start variable substituted by the end variable value of the previous edge (to ensure connectedness). The conditions (4) and (5) bind the `s` and `e` variables in the top level solution.

Next we define the semantics of path queries with start and end patterns:

```
eval(PQ(s, PS, e, PE, PQ) = Join(PS, Join(PE, eval(PQ(s, e, PQ), DG)))
```

where `PS` and `PE` are start and end graph patterns which must bind `s` and `e` variables, respectively. Here `Join` stands for the standard SPARQL join semantics which does not require extensions since joins are performed on variables `s` and `e` which bind to RDF terms only, rather than arrays or solutions (conditions (4) and (5) above ensure that).

Finally we note that path queries with start or end constants are a special case of path queries with the corresponding singleton `VALUES` patterns, e.g.

```
PATHS START ?s = :Alice END ?e = :Dave VIA :knows
```

is a syntactic sugar for

```
PATHS START ?s { VALUES ?s { :Alice } } END ?e { VALUES ?e { :Dave } } VIA :knows
```

Keywords `SHORTEST` (default) and `CYCLIC` are self-explanatory and place further restrictions on each `S(p)`: the sequence should be the shortest among all results or represent a simple cycle. The solution modifiers `ORDER BY`, `LIMIT`, and `OFFSET` have the exact same semantics as in SPARQL 1.1.

GEOSPATIAL QUERY

Stardog supports geospatial queries over data encoded using WGS 84 (http://www.w3.org/2003/01/geo/) or the OGC's GeoSPARQL vocabulary (http://www.opengeospatial.org/standards/geosparql). Any RDF data stored in Stardog using one or both of these vocabularies will be automatically indexed for geospatial queries.

TIP Geospatial: A Primer (https://www.stardog.com/blog/geospatial-a-primer/)

Enabling Geospatial Support

To get started using Stardog's geospatial support, you'll need to create a database with geospatial support enabled. You can do this by setting the option `spatial.enabled` to `true`:

```
stardog-admin db create -o spatial.enabled=true -n mySpatialDb
```

Similarly, you can set the option using `GeospatialOptions#SPATIAL_ENABLED` when creating the database programmatically:

```
aAdminConnection.disk("mySpatialDb")
                                 .set(GeospatialOptions.SPATIAL_ENABLED, true)
                                 .create()
```

Precision & Accuracy

When creating a database with geospatial support, you can specify the precision with which the features are indexed. The database property `spatial.precision` or programmatically via `GeospatialOptions#SPATIAL_PRECISION`, which can only be specified when the database is created, can control the index precision. The default value is `11` which yields sub-meter precision; a value of `8` will give a precision +/- 50m. Setting the precision value lower than the default can **improve the performance of spatial queries at the cost of accuracy**.

Geospatial Data

The WGS84 or OGC vocabularies can be used to encode geospatial features within your dataset. When data is committed, Stardog will look for these vocabularies and automatically extract all features and insert them into the geospatial index. Here is an example of using WKT to define the location of the White House:

```
:whiteHouse a geo:Feature ;
    rdfs:label "White House" ;
    geo:hasGeometry :whiteHouseGeo .

:whiteHouseGeo a geo:Geometry ;
    geo:asWKT "Point(-77.03653 38.897676 )"^^geo:wktLiteral .
```

Note that for WKT formatted points, the location is `<long, lat>`. The location of the White House can also be encoded using the WGS 84 vocabulary:

```
:whiteHouse a :Location ;
    rdfs:label "White House" ;
    wgs:lat "38.897676"^^xsd:float ;
    wgs:long "-77.03653"^^xsd:float .
```

SPARQL Integration

Once your data has been indexed, you can perform several type of geospatial queries on the data. These are seamlessly integrated into SPARQL so you can query for non-spatial information about features in your dataset alongside the geospatial queries.

The operators supported by Stardog are `geof:relate`, `geof:distance`, `geof:within`, `geof:nearby` and `geof:area`. The `geof` namespace is `http://www.opengis.net/def/function/geosparql/ (http://www.opengis.net/def/function/geosparql/)`.

This query gets all features within 2km of Complexible HQ in DC:

```
select ?name where {
  ?loc rdfs:label ?name .
  ?loc geo:hasGeometry ?feature .
  ?hq geo:hasGeometry ?hqGeo ; rdfs:label "Complexible Headquarters" .
  ?feature geof:nearby (?hqGeo 2 <http://qudt.org/vocab/unit#Kilometer>).
}
```

More query examples can be found on our blog (https://www.stardog.com/blog/geospatial-a-primer).

Geospatial Datatypes

The QUDT (http://www.qudt.org/) ontology, namespace `http://qudt.org/vocab/unit# (http://qudt.org/vocab/unit#)`, is used to specify units for distances; `Kilometer`, `Meter`, `Centimeter`, `MileUSStatute`, `Yard`, `Foot`, `Inch`. Additionally, the OGC units vocabulary `http://www.opengis.net/def/uom/OGC/1.0/ (http://www.opengis.net/def/uom/OGC/1.0/)` defines `degree`, `radian` and `metre`.

Enhanced Polygons

Stardog's geospatial support covers the use of basic WKT formatted shapes; specifically points and rectangles. However, WKT can encode more complex spatial structures, most notably, polygons.

To enable support for these more complex shapes, download JTS (http://central.maven.org/maven2/com/vividsolutions/jts-core/1.14.0/jts-core-1.14.0.jar) and include the JAR in Stardog's classpath by placing it into the `server/ext` folder of the installation (you may need to create this folder) or into the folder specified by the `STARDOG_EXT` environment variable. Then set `spatial.use.jts=true` in your `stardog.properties` file. When you restart Stardog, it will pick up JTS and you'll be able to use more complex WKT formatted shapes.

MACHINE LEARNING

In this section, you'll learn how to use Stardog's machine learning capabilities for the general problem of predictive analytics. We'll show you how to build a machine learning model and use it for prediction, plus best practices on modelling your data and improving the quality of results.

TIP

- Machine Learning Tutorial (https://github.com/stardog-union/stardog-examples/tree/develop/examples/machinelearning)
- Learning to Predict (https://www.stardog.com/blog/learning-to-predict/)
- Boosting Machine Learning (https://www.stardog.com/blog/boosting-machine-learning/)

Predictive Analytics

Suppose you have data about movies. But that data is incomplete; some movies are missing the `genre` field. Filling out that missing data is time consuming, and you would like to do it automatically using all the information you already have about the movies. This is where Stardog's predictive analytics comes into the game. Using the data you have about movies with genre, you can create a machine learning model that will predict the genre for the movies that are missing it. Isn't that sweet?

Supervised learning is the basis of this capability. You give Stardog some data about the domain you're interested in, and it will learn a model that can be used to make predictions about properties of that data.

Learning a Model

First step is learning a model, by defining which data will be used in the learning and the target that we are actually trying to predict.

With Stardog, all this is naturally done via SPARQL. The best way to understand the syntax is through an example. Here, we learn a model to predict the genre of a movie given its director, year, and studio.

```
prefix spa: <tag:stardog:api:analytics:>

INSERT {
  graph spa:model {
    :myModel  a spa:ClassificationModel ;
              spa:arguments (?director ?year ?studio) ;
              spa:predict ?genre .
  }
}
WHERE {
   ?movie :directedBy ?director ;
          :year ?year ;
          :studio ?studio ;
          :genre ?genre .
}
```

The `WHERE` clause selects the data and a special graph, `spa:model`, is used to specify the parameters of the training. `:myModel` is the unique identifier given to this model and is composed of 3 mandatory properties.

First, we need to define the type of learning we are performing:

- classification, `spa:ClassificationModel`, if we are interested in predicting a categorical value that has a limited set of possible values (e.g., genre of a movie)
- regression, `spa:RegressionModel`, if we predict a numerical value that can naturally have an unlimited set of values (e.g., box office of a movie)
- similarity, `spa:SimilarityModel`, if we want to predict the degree of similarity between two objects (e.g., most similar movies)

The second property, `spa:arguments`, defines the variables from the `WHERE` clause that will be used as features when learning the model. Here is where you define the data that you think will help to predict the third property, given by `spa:predict`.

In this case, our model will be trained to predict the value of `?genre` based on the values of `?director`, `?year`, and `?studio`.

Properly defining this 3 properties is the main task when creating any model. Using more advanced parameters is covered in the Mastering the Machine (#_mastering_the_machine) section.

Making Predictions

Now that we've learned a model, we can move on to more exciting stuff and use it to actually predict things.

```
prefix spa: <tag:stardog:api:analytics:>

SELECT * WHERE {
  graph spa:model {
      :myModel  spa:arguments (?director ?year ?studio) ;
                spa:predict ?predictedGenre .
  }

  :TheGodfather :directedBy ?director ;
                :year ?year ;
                :studio ?studio ;
                :genre ?originalGenre .
}
```

We select a movie's properties and use them as arguments to the model Stardog previously learned. The magic comes with the `?predictedGenre` variable; during query execution, its value is not going to come from the data itself (like `?originalGenre`), but will instead be predicted by the model, based on the values of the arguments.

The result of the query will look like this:

```
| director            | year | studio             | originalGenre | predictedGenre |
| ------------------- | ---- | ------------------ | ------------- | -------------- |
| :FrancisFordCoppola | 1972 | :ParamountPictures | Drama         | Drama          |
```

Our model seems to be predicting correctly the genre for The Godfather. Yee!

Query Syntax Restrictions

At this point, only basic graph patterns can be used directly inside the prediction query. If more advanced constructs, like `OPTIONAL` or `FILTER`, are necessary, that part of the query needs to be in a sub-query, e.g.:

```
prefix spa: <tag:stardog:api:analytics:>

SELECT * WHERE {
  graph spa:model {
      :myModel  spa:arguments (?director ?year ?studio) ;
                spa:predict ?predictedGenre .
  }

  {
    SELECT * WHERE {
        ?movie  :directedBy ?director ;
                :year ?year ;
                :genre ?originalGenre .
        OPTIONAL { ?movie :studio ?studio }
        FILTER (?year > 2000)
    }
  }
}
```

Selecting a Library

For classification and regression, Stardog can use two distinct machine learning libraries under the covers: Vowpal Wabbit (https://github.com/JohnLangford/vowpal_wabbit) (default) and XGBoost (https://xgboost.readthedocs.io/en/latest/). Both support the same set of functionalities, and can be used interchangeably.

At model creation, the desired library can be selected with the `spa:library` property: `spa:VowpalWabbit` or `spa:XGBoost`.

```
prefix spa: <tag:stardog:api:analytics:>

INSERT {
  graph spa:model {
      :myModel  a spa:ClassificationModel ;
                spa:library spa:XGBoost ;
                spa:arguments (?director ?year ?studio) ;
                spa:predict ?genre .
  }
}
...
```

Vowpal Wabbit is recommended for large, sparse, datasets, while XGBoost is known to perform better in domains with numeric values. We recommend testing both libraries, as their strengths are largely dependent on particularities of the data.

NOTE Learning models with large datasets might exceed the default max query execution time, especially with XGBoost. In those cases, it is recommended to increase the value for the `query.timeout` configuration. Increasing the amount of memory available to Stardog might also make the learning faster.

Assessing Model Quality

Metrics

We provide some special aggregate operators that help quantify the quality of a model.

For classification and similarity problems, one of the most important measures is `accuracy`, that is, the frequency that we predict the target variable correctly.

```
prefix spa: <tag:stardog:api:analytics:>

SELECT (spa:accuracy(?originalGenre, ?predictedGenre) as ?accuracy) WHERE {
  graph spa:model {
      :myModel  spa:arguments (?director ?year ?studio) ;
                spa:predict ?predictedGenre .
  }

  ?movie  :directedBy ?director ;
          :year ?year ;
          :studio ?studio ;
          :genre ?originalGenre .
}
```

```
+---------------------+
| accuracy            |
| ------------------- |
| 0.92488254018       |
+---------------------+
```

For regression, we provide three different measures:

- Mean absolute error, or, on average, how far away is the prediction from the real target number: `spa:mae(?originalValue, ?predictedValue)`
- Mean square error, on average, how much is the squared difference between prediction and the target number: `spa:mse(?originalValue, ?predictedValue)`
- Root mean square error, the square root of the mean square error: `spa:rmse(?originalValue, ?predictedValue)`

Automatic Validation

Classification and regression models are automatically validated with the data used in their training. The score and respective metric can be queried from `spa:model`.

```
prefix spa: <tag:stardog:api:analytics:>

SELECT * WHERE {
  graph spa:model {
    :myModel  spa:validationMetric ?metric ;
              spa:validationScore ?score .
  }
}
```

```
+------------------------------------+-------+
|               metric               | score |
+------------------------------------+-------+
| tag:stardog:api:analytics:accuracy | 1.0   |
+------------------------------------+-------+
```

By default, `spa:accuracy` is used for classification problems, and `spa:mae` for regression. This metric can be changed during model learning, by setting the `spa:validationMetric` argument.

```
prefix spa: <tag:stardog:api:analytics:>

INSERT {
  graph spa:model {
    :myModel  a spa:RegressionModel ;
              spa:validationMetric spa:rmse ;
              ...
  }
}
...
```

Cross Validation

The default automatic validation technique of measuring the accuracy of the model on the same data as training might be prone to overfitting. The most accurate measure we can have is testing on data that the model has never seen before.

We provide a `spa:crossValidation` property, which will automatically apply K-Fold cross validation on the training data, with the number of folds given as an argument.

```
prefix spa: <tag:stardog:api:analytics:>

INSERT {
  graph spa:model {
    :myModel  a spa:RegressionModel ;
              spa:crossValidation 10 ;
              spa:validationMetric spa:rmse ;
              ...
  }
}
  ...
```

```
prefix spa: <tag:stardog:api:analytics:>

SELECT * WHERE {
   graph spa:model {
       :myModel  spa:validation ?validation ;
                 spa:validationMetric ?metric ;
                 spa:validationScore ?score .
      }
}
```

```
+-------------+------------------------------------+-------+
| validation  |               metric               | score |
+-------------+------------------------------------+-------+
| "KFold=10"  | tag:stardog:api:analytics:rmse     | 0.812 |
+-------------+------------------------------------+-------+
```

Modelling Data

The way you input data into Stardog during model learning is of utmost importance in order to achieve good quality predictions.

Data Representation

For better results, each individual you are trying to model should be encoded in a single SPARQL result.

For example, suppose you want to add information about actors into the previous model. The query selecting the data would look as follow:

```
SELECT * WHERE {
   ?movie :actor ?actor ;
          :directedBy ?director ;
          :year ?year ;
          :studio ?studio ;
          :genre ?genre .
}
```

```
| movie         | actor         | director            | year | studio             | genre  |
| ------------- | ------------- | ------------------- | ---- | ------------------ | ------ |
| :TheGodfather | :MarlonBrando | :FrancisFordCoppola | 1972 | :ParamountPictures | Drama  |
| :TheGodfather | :AlPacino     | :FrancisFordCoppola | 1972 | :ParamountPictures | Drama  |
```

Due to the nature of relational query languages like SPARQL, results are returned for all the combinations between the values of the selected variables.

In order to properly model relational domains like this, we introduced a special aggregate operator, `set`. Used in conjunction with `GROUP BY`, we can easily model this kind of data as a single result per individual.

```
prefix spa: <tag:stardog:api:analytics:>

SELECT ?movie (spa:set(?actor) as ?actors) ?director ?studio ?genre WHERE {
   ?movie :actor ?actor ;
          :directedBy ?director ;
          :year ?year ;
          :studio ?studio ;
          :genre ?genre .
}
GROUP BY ?movie ?director ?studio ?genre
```

```
| movie         | actors                     | director             | year | studio              | genre  |
| ------------- | -------------------------- | -------------------- | ---- | ------------------- | ------ |
| :TheGodfather | [:MarlonBrando :AlPacino]  | :FrancisFordCoppola  | 1972 | :ParamountPictures  | Drama  |
```

Data Types

Carefully modelling your data with the correct datatypes can dramatically increase the quality of your model.

As of 5.3.6, Stardog does special treatment on values of the following types:

- Numbers, such as `xsd:int`, `xsd:short`, `xsd:byte`, `xsd:float`, and `xsd:double`, are treated internally as weights and properly model the difference between values
- Strings, `xsd:string` and `rdf:langString`, are tokenized and used in a bag-of-words fashion
- Sets, created with the `spa:set` operator, are interpreted as a bag-of-words of categorical features
- Booleans, `xsd:boolean`, are modeled as binary features

Everything else is modeled as categorical features.

Setting the correct data type for the target variable, given through `spa:predict`, is extremely important:

- with regression, make sure values are numeric
- with classification, individuals of the same class should have consistent data types and values
- with similarity, use values that uniquely identify an object, e.g., an IRI

For evertything else, using the datatype that is closer to its original meaning is a good rule of thumb.

Mastering the Machine

Let’s look at some other issues around the daily care and feeding of predictive analytics and models in Stardog.

Overwriting Models

By default, you cannot create a new model with the same identifier as an already existent one. If you try to do so, you’ll be greeted with a `Model already exists` error.

In order to reuse an existent identifier, users can set the `spa:overwrite` property to `True`. This will delete the previous model and save the new one in its place.

```
prefix spa: <tag:stardog:api:analytics:>

INSERT {
  graph spa:model {
    :myModel  a spa:RegressionModel ;
              spa:overwrite True ;
              ...
  }
}
  ...
```

Deleting Models

Finding good models is an iterative process, and sometimes you'll want to delete your old---not as awesome and now unnecessary---models. This can be achieved with `DELETE DATA` and the `spa:deleteModel` property applied to the model identifier.

```
prefix spa: <tag:stardog:api:analytics:>

DELETE DATA {
  graph spa:model {
      [] spa:deleteModel :myModel .
  }
}
```

Classification and Similarity with Confidence Levels

Sometimes, besides predicting the most probable value for a property, you will be interested to know the confidence of that prediction. By providing the `spa:confidence` property, you can get confidence levels for all the possible predictions.

```
prefix spa: <tag:stardog:api:analytics:>

SELECT * WHERE {
  graph spa:model {
      :myModel  spa:arguments (?director ?year ?studio) ;
                spa:confidence ?confidence ;
                spa:predict ?predictedGenre .
  }

  :TheGodfather :directedBy ?director ;
          :year ?year ;
          :studio ?studio .
}
ORDER BY DESC(?confidence)
LIMIT 3
```

```
| director            | year | studio             | predictedGenre | confidence     |
| ------------------- | ---- | ------------------ | -------------- | -------------- |
| :FrancisFordCoppola | 1972 | :ParamountPictures | Drama          | 0.649688932    |
| :FrancisFordCoppola | 1972 | :ParamountPictures | Crime          | 0.340013045    |
| :FrancisFordCoppola | 1972 | :ParamountPictures | Sci-fi         | 0.010298023    |
```

These values can be interpreted as the probability of the given prediction being the correct one and are useful for tasks like ranking and multi-label classification.

Tweaking Parameters

Both Vowpal Wabbit (https://github.com/JohnLangford/vowpal_wabbit/wiki/Command-line-arguments), XGBoost (https://xgboost.readthedocs.io/en/latest//parameter.html), and similarity search can be configured with the `spa:parameters` property.

```
prefix spa: <tag:stardog:api:analytics:>

INSERT {
  graph spa:model {
    :myModel  a spa:ClassificationModel ;
              spa:library spa:VowpalWabbit ;
              spa:parameters [
                spa:learning_rate 0.1 ;
                spa:sgd True ;
                spa:hash 'all'
              ] ;
              spa:arguments (?director ?year ?studio) ;
              spa:predict ?genre .
  }
}
...
```

Parameter names for both libraries are valid properties in the `spa` prefix, and their values can be set during model creation.

Vowpal Wabbit

By default, models are learned with `[ spa:loss_function "logistic"; spa:probabilities true; spa:oaa true ]` in classification mode, and `[ spa:loss_function "squared" ]` in regression. Those parameters are overwritten when using the `spa:arguments` property with regression, and appended in classification.

Check the official documentation (https://github.com/JohnLangford/vowpal_wabbit/wiki/Command-line-arguments) for a full list of parameters. Some tips that might help with your choices:

- Use cross-validation when tweaking parameters. Otherwise, make sure your testing set is not biased and represents a true sample of the original data.
- The most important parameter to tweak is the learning rate `spa:l`. Values between 1 and 0.01 usually give the best results.
- To prevent overfitting, set `spa:l1` or `spa:l2` parameters, preferably with a very low value (e.g., 0.000001).
- If number of distinct features is large, make sure to increase the number of bits `spa:b` to a larger value (e.g., 22).
- Each argument given with `spa:arguments` has its own namespace, identified by its numeric position in the list (starting with 0). For example, to create quadratic features between `?director` and `?studio`, set `spa:q "02"`.
- If caching is enabled (e.g., with `spa:passes`), always use the `[ spa:k true; spa:cache_file "fname" ]` parameters, where `fname` is a unique filename for that model.
- In regression, the target variable given with `spa:predict` is internally normalized into the `[0-1]` range, and denormalized back to its normal range during query execution. For certain problems where numeric arguments have large values, performance might be improved by performing a similar normalization as a pre-processing step.

XGBoost

Models are learned with `spa:objective "multi:softprob"` in classification, and `spa:objective "reg:linear"` in regression. See this list (https://xgboost.readthedocs.io/en/latest//parameter.html) for a complete set of available parameters.

Similarity Search

The underlying algorithm is based on cluster pruning (https://nlp.stanford.edu/IR-book/html/htmledition/cluster-pruning-1.html), an approximate search algorithm which groups items based on their similarity in order to speed up query performance.

The minimum number of items per cluster can be configured with the `spa:minClusterSize` property, which is set to 100 by default.

```
prefix spa: <tag:stardog:api:analytics:>

INSERT {
  graph spa:model {
    :myModel  a spa:SimilarityModel ;
              spa:parameters [
                spa:minClusterSize 100 ;
              ] ;
              spa:arguments (?director ?year ?studio) ;
              spa:predict ?movie .
  }
}
...
```

This number should be increased with datasets containing many near-duplicate items.

During prediction, there are two parameters available:

- `spa:limit`, which restricts the number of top N items to return; by default, it returns only the top item, or all items if using `spa:confidence`.
- `spa:clusters`, which sets the number of similarity clusters used during the search, with a default value of 1. Larger numbers will increase recall, at the expense of slower query time.

For example, the following query will return the top 3 most similar items and their confidence scores, restricting the search to 10 clusters.

```
prefix spa: <tag:stardog:api:analytics:>

SELECT * WHERE {
  graph spa:model {
    :myModel  spa:parameters [
                spa:limit 3 ;
                spa:clusters 10 .
              ] ;
              spa:confidence ?confidence ;
              spa:arguments (?director ?year ?studio) ;
              spa:predict ?similar .
  }
}
...
```

Hyperparameter Optimization

Finding the best parameters for a model is a time consuming, laborious, process. Stardog helps to ease the pain by performing an exhaustive search through a manually specified subset of parameter values.

```
prefix spa: <tag:stardog:api:analytics:>

INSERT {
  graph spa:model {
    :myModel  a spa:ClassificationModel ;
              spa:library spa:VowpalWabbit ;
              spa:parameters [
                spa:learning_rate (0.1 1 10) ;
                spa:hash ('all' 'strings')
              ] ;
              spa:arguments (?director ?year ?studio) ;
              spa:predict ?genre .
  }
}
...
```

All possible sets of parameter configurations that can be built from the given values (`spa:learning_rate 0.1 ; spa:hash 'all'`, `spa:learning_rate 1 ; spa:hash 'all'`, and so on) will be evaluated (#_automatic_validation). The best configuration will be chosen, and its model saved in the database.

Afterwards, parameters are available for querying, just like any other model metadata.

```
prefix spa: <tag:stardog:api:analytics:>

SELECT * WHERE {
    graph spa:model {
        :myModel  spa:parameters [ ?parameter ?value ]
    }
}
```

```
+-------------------+-------+
|     parameter     | value |
+-------------------+-------+
| spa:hash          | "all" |
| spa:learning_rate | 1     |
+-------------------+-------+
```

Native Library Errors

Stardog ships with a pre-compiled version of Vowpal Wabbit (VW) that works out of the box with most MacOSX/Linux 64bit distributions.

If you have a 32 bit operating system, or an older version of Linux, you will be greeted with a `Unable to load analytics native library` error when trying to create your first model.

```
Exception in thread "main" java.lang.RuntimeException: Unable to load analytics native library. Please
refer to http://www.stardog.com/docs/#_native_library_errors
        at vowpalWabbit.learner.VWLearners.loadNativeLibrary(VWLearners.java:94)
        at vowpalWabbit.learner.VWLearners.initializeVWJni(VWLearners.java:76)
        at vowpalWabbit.learner.VWLearners.create(VWLearners.java:44)
        ...
Caused by: java.lang.RuntimeException: Unable to load vw_jni library for Linux (i386)
```

In this case, you will need to install VW manually. Fear not! Instructions are easy to follow.

```
git clone https://github.com/cpdomina/vorpal.git
cd vorpal/build-jni/
./build.sh
sudo cp transient/lib/vw_wrapper/vw_jni.lib /usr/lib/libvw_jni.so
```

You might need to install some dependencies, namely `zlib-devel`, `automake`, `libtool`, and `autoconf`.

After this process is finished, restart the Stardog server and everything should work as expected.

PROPERTY GRAPHS

In addition to RDF, SPARQL, OWL, and SNARL, Stardog supports the non-semantic property graph model (http://tinkerpop.incubator.apache.org/docs/3.0.2-incubating/#intro), Gremlin graph traversal language, and Apache TinkerPop 3 (http://tinkerpop.com/) APIs. For information on how to use the TinkerPop 3, please refer to its documentation (http://tinkerpop.incubator.apache.org/docs/3.0.2-incubating). Details about Stardog's support for TinkerPop 3 Features (http://tinkerpop.incubator.apache.org/docs/3.0.2-incubating/#_features) can be found in Stardog Feature Set (/docs/5.3.6/java/snarl/com/complexible/stardog/gremlin/features/stardogfeatureset).

NOTE Stardog `5.3.6` supports TinkerPop `3.0.2-incubating`.

Motivation & Implementation

Stardog's implementation of TinkerPop 3 is based ultimately on a (seamless and opaque) translation to and from RDF, in which Stardog persists all vertices, edges and properties. In order to support edge properties in the RDF model, Stardog includes a reification function which allows statement identifiers to be used as the subject of an RDF quad; this extends the RDF Quad model used in Stardog to have a notion of virtual "quints".

Having virtual quints in Stardog lets us manipulate existing RDF content as a property graph; but, most importantly, it lets us use Stardog capabilities (reasoning, ICV, etc) with property graphs. Reification extends existing Stardog graph database and let users add edge properties if required via the TinkerPop 3 or even SPARQL.

Okay, so why add property graph support to Stardog? A few reasons:

1. sometimes you need to traverse, rather than query, a graph
2. sometimes you need to traverse a **semantic** graph

Example

Loading the TinkerGraph Modern (http://tinkerpop.incubator.apache.org/docs/3.0.2-incubating/#intro) graph via TinkerPop 3 (using Gremlin Console), using the described Graph Configuration (#_graph_configuration):

```
Graph graph = StardogGraphFactory.open(...)  (1)
graph.io(graphml()).readGraph('data/tinkerpop-modern.xml')  (2)
```

1. Get the Graph from the `StardogGraphFactory`
2. Load the graph `tinkerpop-modern` included in Gremlin Console distribution at `data/tinkerpop-modern.xml`.

That produces the following internal representation in Stardog:

```
Unresolved directive in property-graph.ad -
include::https://gist.githubusercontent.com/edgarRd/c20e5bd963e7526c54f3/raw/13a54e8fc771a33c2fa6ef42f98904
c7d7265c1b/tinkerpop-modern.ttl[]
```

WARNING This translation between RDF and property graph models is transparent to the user. It just works. But, of course, since in the end it's just RDF, you can always query or interact with it as RDF directly using SPARQL, Jena, Sesame, or SNARL code, etc. However, the mapping between Property Graphs and RDF is **not** considered part of Stardog's contract so it may change without notice. You've been warned!

Getting properties for in-edges for a vertex from the previous graph, using the TinkerPop 3 API:

```
g = graph.traversal()                          (1)
g.V('https://www.tinkerpop.com/software-5964a0af-1bb4-4469-b362-6b7db5e617e2').inE().properties()  (2)
g.V().has('name','lop').inE().properties()  (3)
```

1. Get a traversal that can be reused
2. Get a vertex using its IRI Id and list in-edge properties
3. Get a vertex filtering by name and list in-edge properties

Integration with SPARQL

Access to the reification function is available via SPARQL in order to be able to query edge properties created via the TinkerPop 3 API, e.g. query to find the first 10 edge properties, excluding the label:

```
select ?srcName ?edgeLabel ?destName ?edgeProp ?val where {
  ?src ?pred ?dest .
  ?src tp:name ?srcName .
  ?dest tp:name ?destName .
  BIND(stardog:identifier(?src, ?pred, ?dest) as ?edgeId) . (1)
  ?edgeId rdfs:label ?edgeLabel .
  ?edgeId ?edgeProp ?val .
  FILTER (?edgeProp != rdfs:label) .
} limit 10
```

1. Using the `stardog:identifier()` (aka "reification") function.

Database Configuration

Any Stardog database should work out-of-the-box with the Stardog TinkerPop 3 implementation, but given that Stardog enables by default RDF literal canonicalization (/docs/5.3.6/java/snarl/com/complexible/stardog/index/indexoptions#CANONICAL_LITERALS), some property value types may not be as expected when fetching them from the TinkerPop 3 graph. To allow for better compatibility between TinkerPop 3 and Stardog, the setting `index.literals.canonical` must be disabled in the database at creation time, using the following command:

```
$ stardog-admin db create -o index.literals.canonical=false -n <dbname>
```

Graph Configuration

In order to create TinkerPop 3 graphs, a configuration object must be created to set up the graph. The TinkerPop 3 implementation for Stardog contains a tool for creating this configuration easily, supporting many of the features available in Stardog, such as reasoning and named-graphs. The StardogGraphConfiguration (/docs/5.3.6/java/snarl/com/complexible/stardog/gremlin/stardoggraphconfiguration), is available via the API or the Gremlin Console in Groovy.

```
gremlin> graphConf = StardogGraphConfiguration.builder()
...
gremlin> graphConf.connectionString("http://localhost:5820/mygraph").credentials("admin", "admin")
...
gremlin> graphConf.baseIRI("http://tinkerpop.incubator.apache.org/").reasoning(false)
==>gremlin.graph=tag:stardog:api:context:default
stardog.computer.cache_size=5000
stardog.label_iri=http://www.w3.org/2000/01/rdf-schema#label
stardog.connection=http://localhost:5820/mygraph
stardog.user=admin
stardog.password=admin
stardog.base_iri=http://tinkerpop.incubator.apache.org/
stardog.reasoning_enabled=false
gremlin> graph = StardogGraphFactory.open(graphConf.build())
==>cachedstardoggraph[cachedstardoggraph]
```

Stardog & Gremlin Console

Stardog's TinkerPop 3 implementation includes a plugin for Gremlin Console (http://tinkerpop.incubator.apache.org/docs/3.0.2-incubating/#gremlin-console).

Installation

The following steps describe how to install the Stardog plugin into the Gremlin console:

1. Create `stardog-gremlin/plugin` directory within the `ext/` directory in the Gremlin console directory.

```
~/gremlin-console/ext/$ mkdir -p stardog-gremlin/plugin
```

2. Flat-copy all Stardog client jar files to the directory created in the previous step.

```
~/gremlin-console/ext/stardog-gremlin/plugin$ find stardog/client -iname '*.jar' -exec cp \{\} . \;
```

3. Make sure the jar file `stardog-gremlin-X.X.X.jar` is contained in the `stardog-gremlin/plugin` directory along with all other Stardog jars; copy the jar if it doesn't exist.
4. Start the Gremlin Console and make sure the `complexible.stardog` plugin has been loaded.

```
~/gremlin-console$ bin/gremlin.sh
         \,,,/
         (o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin> :plugin list
==>tinkerpop.server[active]
==>tinkerpop.gephi
==>tinkerpop.utilities[active]
==>tinkerpop.sugar
==>complexible.stardog
==>tinkerpop.tinkergraph[active]
```

5. Activate the `complexible.stardog` plugin in Gremlin Console

```
gremlin> :plugin use complexible.stardog
==>complexible.stardog activated
```

6. You're done installing the stardog-gremlin plugin for Gremlin Console. Now you can create a `StardogGraph` and start exploring the TinkerPop 3 API with Stardog.

Using a Stardog Graph

The following describes the process to create a `StardogGraph` and explore data in Stardog using the TinkerPop 3 API via the Gremlin Console.

The only requirement is that you have an existent database in Stardog as directed in Database Configuration (#_database_configuration), which could be in-memory or disk based. Assuming you already installed the Stardog plugin for the Gremlin Console and it is active, start the Gremlin Console.

```
gremlin-console$ bin/gremlin.sh
```

In the Gremlin Console, create the configuration settings for opening the `StardogGraph`. Assuming the Stardog server is running in `localhost:5820`, the user is `admin` and password `admin`.

```
gremlin> graphConf = StardogGraphConfiguration.builder()
...
gremlin> graphConf.connectionString("http://localhost:5820/mygraph").credentials("admin",
"admin").baseIRI("http://tinkerpop.incubator.apache.org/")
==>gremlin.graph=tag:stardog:api:context:default
stardog.computer.cache_size=5000
stardog.label_iri=http://www.w3.org/2000/01/rdf-schema#label
stardog.connection=http://localhost:5820/mygraph
stardog.user=admin
stardog.password=admin
stardog.base_iri=http://tinkerpop.incubator.apache.org/
gremlin> graph = StardogGraphFactory.open(graphConf.build())
==>cachedstardoggraph[cachedstardoggraph]
```

Named Graphs

The previous commands will create a Graph within the `default` graph of the Stardog database `mygraph`. A database can contain multiple graphs, which would be the equivalent to `named-graphs` in Stardog.

To create a `StardogGraph` over a specific named-graph, just set the named-graph URI in the Graph Configuration (#_graph_configuration) for the `StardogGraph` to create:

```
gremlin> graphConf.namedGraph("tag:graph1")
==>gremlin.graph=tag:stardog:api:context:default
...
stardog.named_graph=tag:graph1
...
```

NOTE by default, the property `gremlin.graph` is set to the default graph in a Stardog database; setting the `stardog.named_graph` configuration option will override the graph option.

Stardog & Gremlin Server

The TinkerPop 3 implementation for Stardog includes a plugin for Gremlin Server (http://tinkerpop.incubator.apache.org/docs/3.0.0.M9-incubating/#gremlin-server).

Installation

The following steps describe how to install the Stardog plugin into the gremlin server:

1. Create `stardog-gremlin/plugin` directory within the `ext/` directory in the gremlin server directory.

```
~/gremlin-server/ext/$ mkdir -p stardog-gremlin/plugin
```

2. Flat-copy all Stardog client jar files to the directory created in the previous step.

```
~/gremlin-server/ext/stardog-gremlin/plugin$ find ~/stardog/client -iname '*.jar' -exec cp \{\} . \;
```

3. Make sure the jar file `stardog-gremlin-X.X.X.jar` is contained in the `stardog-gremlin/plugin` directory along with all other Stardog jars; copy the jar if it doesn't exist.

Configure Stardog Graphs

To setup a graph for use with the Gremlin Server you need to create a configuration file in `conf/` with the Stardog graph properties. The following example file, `stardoggraph-mygraph.properties`, contains the required properties to use a Stardog graph, described in Graph Configuration (#_graph_configuration):

```
# Properties for creating a StardogGraph in Gremlin Server
gremlin.graph=com.complexible.stardog.gremlin.structure.StardogGraph
stardog.connection=http://localhost:5820/mygraph
stardog.user=admin
stardog.password=admin
stardog.named_graph=tag:stardog:api:graph:default
stardog.reasoning_enabled=false
```

In the previous example, `gremlin.graph` defines the TinkerPop Class implementation to use, in this case is the `StardogGraph`. The property `gremlin.stardog.named_graph` is required when configuring a graph in Gremlin Server, if the graph is contained in the Stardog DB's default graph, the value to use is: `tag:stardog:api:graph:default` as shown in the example; if other named-graph is used, just set the value to the named-graph's `URI`. The rest of the properties are just connection settings to the Stardog server.

Now you need to point to the Stardog graph properties file from the server configuration file, `conf/gremlin-server.yaml`, and enable the Stardog plugin. the following are the relevant parts of the configuration file that need to be set:

```
graphs: {
  graph: conf/stardoggraph-mygraph.properties      (1)
}
plugins:
  - complexible.stardog                            (2)
...
```

1. set the stardog graph properties
2. enable the stardog gremlin plugin

Running the Gremlin Server

Having a Stardog server running, at this point you're ready to start the Gremlin Server.

```
~/gremlin-server$ bin/gremlin-server.sh
```

You should see that the Gremlin Server creates an instance of the `StardogGraph`, named `graph`, based on the properties file configured.

```
[INFO] Graphs - Graph [graph] was successfully configured via [conf/stardoggraph-mygraph.properties].
```

SECURITY

Stardog's security model is based on standard role-based access control: users have **permissions** over **resources** during **sessions**; permissions can be grouped into **roles**; and roles can be assigned to **users**.

Stardog uses Apache Shiro (http://shiro.apache.org/) for authentication, authorization, and session management and jBCrypt (http://www.mindrot.org/projects/jBCrypt/) for password hashing.

Resources

A resource is some Stardog entity or service to which access is controlled. Resources are identified by their **type** and their **name**. A particular resource is denoted as `type_prefix:name`. The valid resource types with their prefixes are shown below.

8. Table of System Resources

Resource	Prefix	Description

Resource	Prefix	Description
User	`user`	A user (e.g., `user:admin`)
Role	`role`	A role assigned to a user (`role:reader`)
Database	`db`	A database (`db:myDB`)
Named Graph	`named-graph`	A named graph (graph subset) (`named-graph:myDb\named-graph-id`)
Database Metadata	`metadata`	Metadata of a database (`metadata:myDB`)
Database Admin	`admin`	Database admin tasks (e.g., `admin:myDB`)
Integrity Constraints	`icv-constraints`	Integrity constraints associated with a database (e.g., `icv-constraints:myDB`)

Permissions

Permissions are composed of a **permission subject**, an **action**, and a **permission object**, which is interpreted as **the subject resource can perform the specified action over the object resource**.

Permission subjects can be of type `user` or `role` only. Permission objects can be of any valid type.

NOTE `write` permission in Stardog refers to graph contents, including mutative operations performed via SPARQL Update (i.e., `INSERT`, `DELETE`, etc.). The other permissions, i.e., `create` and `delete`, apply to resources of the system itself, i.e., users, databases, database metadata, etc.

Valid actions include the following:

`read`
: Permits reading the resource properties

`write`
: Permits changing the resource properties

`create`
: Permits creating new resources

`delete`
: Permits deleting a resource

`grant`
: Permits granting permissions over a resource

`revoke`
: Permits revoking permissions over a resource

`execute`
: Permits executing administration actions over a database

`all`
: Special action type that permits all previous actions over a resource

Wildcards

Stardog understands the use of wildcards to represent sets of resources. A wildcard is denoted with the character `*`. Wildcards can be used to create complex permissions; for instance, we can give a user the ability to create any database by granting it a `create` permission over `db:*`. Similarly, wildcards can be used in order to revoke multiple permissions simultaneously.

Superusers

It is possible at user-creation time to specify that a given user is a superuser. Being a superuser is equivalent to having been granted an `all` permission over every resource, i.e., `*:*`. Therefore, as expected, superusers are allowed to perform any valid action over any existing (or future) resource.

Database Owner Default Permissions

When a user creates a resource, it is automatically granted `delete`, `write`, `read`, `grant`, and `revoke` permissions over the new resource. If the new resource is a database, then the user is additionally granted `write`, `read`, `grant`, and `revoke` permissions over `icv-constraints:theDatabase` and `execute` permission over `admin:theDatabase`. These latter two permissions give the owner of the database the ability to administer the ICV constraints for the database and to administer the database itself.

Default Security Configuration

WARNING Out of the box, the Stardog security setup is minimal and **insecure**: `user:admin` with password set to "admin" is a superuser; `user:anonymous` with password "anonymous" has the "reader" role; `role:reader` allows `read` of any resource.

Do not deploy Stardog in production or in hostile environments with the default security settings.

Setting Password Constraints

To setup the constraints used to validate passwords when adding new users, configure the following settings in the `stardog.properties` configuration file.

- `password.length.min`: Sets the password policy for the minimum length of user passwords, the value can’t be less than 1 or greater than `password.length.max`. Default: `4`.
- `password.length.max`: Sets the password policy for the maximum length of user passwords, the value can’t be greater than 1024 or less than 1. Default: `20`.
- `password.regex`: Sets the password policy of accepted chars in user passwords, via a Java regular expression. Default: `[\w@#$%!&]+`

Using a Password File

To avoid putting passwords into scripts or environment variables, you can put them into a suitably secured password file. If no credentials are passed explicitly in CLI invocations, or you do not ask Stardog to prompt you for credentials interactively, then it will look for credentials in a password file.

On a Unix system, Stardog will look for a file called `.sdpass` in the home directory of the user Stardog is running as; on a Windows system, it will look for `sdpass.conf` in `Application Data\stardog` in the home directory of the user Stardog is running as. If the file is not found in these locations, Stardog will look in the location provided by the `stardog.passwd.file` system property.

Password File Format

The format of the password file is as follows:

- any line that starts with a `#` is ignored
- each line contains a single password in the format: `hostname:port:database:username:password`.
- wildcards, `*`, are permitted for any field but the password field; colons and backslashes in fields are escaped with `\`.

For example,

```
#this is my password file; there are no others like it and this one is mine anyway...
*:*:*:flannery:aNahthu8
*:*:summercamp:jemima:foh9Moaz
```

Of course you should secure this file carefully, making sure that only the user that Stardog runs as can read it.

Named Graph Security

Stardog's security model is based on standard RBAC notions: users have permissions over resources during sessions; permissions can be grouped into roles; and roles can be assigned to users. Stardog defines a database resource type so that users and roles can be given read or write access to a database. With Named Graph Security added in Stardog 3.1, Stardog lets you specify which named graphs a user can read from or write to; that is, named graphs are now an explicit resource type in Stardog's security model.

Example

To grant a user permissions to a named graph,

```
$ stardog-admin user grant -a read -o named-graph:myDB\http://example.org/g1 myUser
$ stardog-admin user grant -a write -o named-graph:myDB\http://example.org/g2 myUser
```

Note the use of "\" to separate the name of the database ("myDB") from the named graph identifier ("http://example.org/g1").

IMPORTANT Named Graph Security is **disabled** by default (for backwards compatibility with the installed base). It can be enabled globally (or per database) by setting `security.named.graphs=true`, in `stardog.properties` globally, or per database.

Named Graph Operations

Stardog does not support the notion of an empty named graph; thus, there is no operation to create a named graph. Deleting a named graph is simply removing all the triples in that named graph; so it's also not a special operation. For this reason, only **read** and **write** permissions can be used with named graphs and **create** and **delete** permissions cannot be used with named graphs.

How Named Graph Permissions Work

The set of named graphs to which a user has read or write access is **the union of named graphs for which it has been given explicit access plus the named graphs for which the user's roles have been given access**.

Querying

An effect of named graph permissions is changing the RDF Dataset associated with a query. The default and named graphs specified for an RDF Dataset will be filtered to match the named graphs that a user has read access to.

NOTE A read query never triggers a security exception due to named graph permissions. The graphs that a user cannot read from would be **silently dropped from the RDF dataset for the query**, which may cause the query to return no answers, despite there being matching triples in the database.

The RDF dataset for SPARQL update queries will be modified similarly based on read permissions.

NOTE The RDF dataset for an update query affects only the `WHERE` clause.

Writing

Write permissions are enforced by throwing a security exception whenever a named graph is being updated by a user that does not have write access to the graph. Adding a triple to an unauthorized named graph will raise an exception even if that triple already exists in the named graph. Similarly trying to remove a non-existent triple from an unauthorized graph raises an error.

NOTE The unauthorized graph may not exist in the database; any graph that is not explicitly listed in a user's permissions is unauthorized.

Updates either succeed as a whole or fail. If an update request tries to modify both an authorized graph an unauthorized graph, it would fail without making any modifications.

Reasoning

Stardog allows a set of named graphs to be used as the schema for reasoning. The OWL axioms and rules defined in these graphs are extracted and used in the reasoning process. The schema graphs are specified in the database configuration and affect all users running reasoning queries.

Named graph permissions do **not** affect the schema axioms used in reasoning and every reasoning query will use the same schema axioms even though some users might **not** have been granted explicit read access to schema graphs. But non-schema axioms in those named graphs would **not** be visible to users without authorization.

Enterprise Authentication

Stardog can use an LDAP server to authenticate enterprise users. Stardog assumes the existence of two different groups to identify regular and superusers, respectively. Groups must be identified with the `cn` attribute and be instances of the `groupOfNames` object class. Users must be specified using the `member` attribute.

For example,

```
dn: cn=stardogSuperUsers,ou=group,dc=example,dc=com
cn: stardogSuperUsers
objectclass: groupOfNames
member: uid=superuser,ou=people,dc=example,dc=com

dn: cn=stardogUsers,ou=group,dc=example,dc=com
cn: stardogUsers
objectclass: groupOfNames
member: uid=regularuser,ou=people,dc=example,dc=com
member: uid=anotherregularuser,ou=people,dc=example,dc=com
```

Credentials and other user information are stored as usual:

```
dn: uid=superuser,ou=people,dc=example,dc=com
objectClass: inetOrgPerson
cn: superuser
sn: superuser
uid: superuser
userPassword: superpassword
```

Configuring Stardog

In order to enable LDAP authentication in Stardog, we need to include the following **mandatory** properties in `stardog.properties`:

- `security.realms`: with a value of `ldap`
- `ldap.provider.url`: The URL of the LDAP server
- `ldap.security.principal`: An LDAP user allowed to retrieve group members from the LDAP server
- `ldap.security.credentials`: The principal's password
- `ldap.user.dn.template`: A template to form LDAP names from Stardog usernames
- `ldap.group.lookup.string`: A string to lookup the Stardog user groups
- `ldap.users.cn`: The `cn` of the group identifying regular Stardog users

- `ldap.superusers.cn`: The `cn` of the group identifying Stardog super users
- `ldap.cache.invalidate.time`: The time duration to invalidate cache entries, default to `24h`.

Here's another example:

```
security.realms = ldap
ldap.provider.url = ldap://localhost:5860
ldap.security.principal = uid=admin,ou=people,dc=example,dc=com
ldap.security.credentials = secret
ldap.user.dn.template = uid={0},ou=people,dc=example,dc=com
ldap.group.lookup.string = ou=group,dc=example,dc=com
ldap.users.cn = stardogUsers
ldap.superusers.cn = stardogSuperUsers
ldap.cache.invalidate.time = 1h
```

User Management

Users can no longer be added/removed/modified via Stardog. User management is delegated to the LDAP server.

An LDAP Quirk

When Stardog manages users, instead of delegating to LDAP, when a user is created, they are assigned the permission `read:user:$NEW_USER`. But when user management is delegated to LDAP, this permission is not automatically created at new user creation time in Stardog and, therefore, it should be added manually to Stardog. If this doesn't happen, users won't be able to—among other things—log into the Web Console.

Authenticated User Cache

Stardog includes a time constrained cache with a configurable time for eviction, default to 24 hours. To disable the cache, the eviction time must be set to `0ms`.

Authorization

The LDAP server is used for **authentication only**. Permissions and roles are assigned in Stardog.

Stale Permissions/Roles

Permissions and roles in Stardog might refer to users that no long exist, i.e., those that were deleted from the LDAP server. This is safe as these users will not be able to authenticate (see above). It is possible to configure Stardog to periodically clean up the list of permissions and roles according to the latest users in the LDAP server. In order to do this, we pass a Quartz cron expression (http://www.quartz-scheduler.org/documentation/quartz-1.x/tutorials/crontrigger) using the `ldap.consistency.scheduler.expression` property:

```
## Execute the consistency cleanup at 6pm every day
ldap.consistency.scheduler.expression = 0 0 18 * * ?
```

Managing Stardog Securely

Stardog resources can be managed securely by using the tools included in the admin CLI or by programming against Stardog APIs. In this section we describe the permissions required to manage various Stardog resources either by CLI or API.

Users

Create a user

`create` permission over `user:*`. Only superusers can create other superusers.

Delete a user

`delete` permission over the user.

Enable/Disable a user

User must be a superuser.

Change password of a user

User must be a superuser or user must be trying to change its own password.

Check if a user is a superuser

`read` permission over the user or user must be trying to get its own info.

Check if a user is enabled

`read` permission over the user or user must be trying to get its own info.

List users

Superusers can see all users. Other users can see only users over which they have a permission.

Roles

Create a role

`create` permission over `role:*`.

Delete a role

`delete` permission over the role.

Assign a role to a user

`grant` permission over the role **and** user must have all the permissions associated to the role.

Unassign a role from a user

`revoke` permission over the role **and** user must have all the permissions associated to the role.

List roles

Superusers can see all roles. Other users can see only roles they have been assigned or over which they have a permission.

Databases

Create a database

`create` permission over `db:*`.

Delete a database

`delete` permission over `db:theDatabase`.

Add/Remove integrity constraints to a database

`write` permission over `icv-constraints:theDatabase`.

Verify a database is valid

`read` permission over `icv-constraints:theDatabase`.

Online/Offline a database

`execute` permission over `admin:theDatabase`.

Migrate a database

`execute` permission over `admin:theDatabase`.

Optimize a database

`execute` permission over `admin:theDatabase`.

List databases

Superusers can see all databases. Regular users can see only databases over which they have a permission.

Permissions

Grant a permission

`grant` permission over the permission object **and** user must have the permission that it is trying to grant.

Revoke a permission from a user or role over an object resource

`revoke` permission over the permission object **and** user must have the permission that it is trying to revoke.

List user permissions

User must be a superuser or user must be trying to get its own info.

List role permissions

User must be a superuser or user must have been assigned the role.

Deploying Stardog Securely

To ensure that Stardog's RBAC access control implementation will be effective, all non-administrator access to Stardog databases should occur over network (i.e., non-native) database connections.[35 (#_footnote_35)]

To ensure the confidentiality of user authentication credentials when using remote connections, the Stardog server should only accept connections that are encrypted with SSL.

Configuring Stardog to use SSL

Stardog HTTP server includes native support for SSL. To enable Stardog to optionally support SSL connections, just pass `--enable-ssl` to the server start command. If you want to require the server to use SSL only, that is, to reject any non-SSL connections, then use `--require-ssl`.

When starting from the command line, Stardog will use the standard Java properties for specifying keystore information:

- `javax.net.ssl.keyStorePassword` (the password)
- `javax.net.ssl.keyStore` (location of the keystore)
- `javax.net.ssl.keyStoreType` (type of keystore, defaults to JKS)

These properties are checked first in `stardog.properties`; then in JVM args passed in from the command line, e.g. `-Djavax.net.ssl.keyStorePassword=mypwd`. If you're creating a Server programmatically via `ServerBuilder`, you can specify values for these properties using the appropriate `ServerOptions` when creating the server. These values will override anything specified in `stardog.properties` or via normal JVM args.

Configuring Stardog Client to use SSL

Stardog HTTP client supports SSL when the `https:` scheme is used in the database connection string. For example, the following invocation of the Stardog command line utility will initiate an SSL connection to a remote database:

```
$ stardog query https://stardog.example.org/sp2b_10k "ask { ?s ?p ?o }"
```

If the client is unable to authenticate to the server, then the connection will fail and an error message like the following will be generated.

```
Error during connect.  Cause was SSLPeerUnverifiedException: peer not authenticated
```

The most common cause of this error is that the server presented a certificate that was not issued by an authority that the client trusts. The Stardog client uses standard Java security components to access a store of trusted certificates. By default, it trusts a list of certificates installed with the Java runtime environment, but it can be configured to use a custom trust store.[36 (#_footnote_36)]

The client can be directed to use a specific Java KeyStore file as a trust store by setting the `javax.net.ssl.trustStore` system property. To address the authentication error above, that trust store should contain the issuer of the server's certificate. Standard Java tools can create such a file. The following invocation of the `keytool` utility creates a new trust store named `my-truststore.jks` and initializes it with the certificate in `my-trusted-server.crt`. The tool will prompt for a passphrase to associate with the trust store. This is not used to encrypt its contents, but can be used to ensure its integrity.[37 (#_footnote_37)]

```
$ keytool -importcert  -keystore my-truststore.jks -alias stardog-server -file my-trusted-server.crt
```

The following Stardog command line invocation uses the newly created truststore.

```
$ STARDOG_SERVER_JAVA_ARGS="-Djavax.net.ssl.trustStore=my-truststore.jks"
$ stardog query https://stardog.example.org/sp2b_10k "ask { ?s ?p ?o }"
```

For custom Java applications that use the Stardog client, the system property can be set programmatically or when the JVM is initialized.

The most common deployment approach requiring a custom trust store is when a self-signed certificate is presented by the Stardog server. For connections to succeed, the Stardog client must trust the self-signed certificate. To accomplish this with the examples given above, the self-signed certificate should be in the `my-trusted-server.crt` file in the keytool invocation.

A client may also fail to authenticate to the server if the hostname in the Stardog database connection string does not match a name contained in the server certificate.[38 (#_footnote_38)]

This will cause an error message like the following

```
Error during connect.  Cause was SSLException: hostname in certificate didn't match
```

The client does not support connecting when there's a mismatch; therefore, the only workarounds are to replace the server's certificate or modify the connection string to use an alias for the same server that matches the certificate.

PROGRAMMING STARDOG

You can program Stardog in Java, over HTTP, JavaScript, Clojure, Groovy, Spring, and .Net.

Sample Code

There's a Github repo of example Java code (https://github.com/complexible/stardog-examples) that you can fork and use as the starting point for your Stardog projects. Feel free to add new examples using pull requests in Github.

JAVA PROGRAMMING

In the Network Programming (#_network_programming) section, we look at how to interact with Stardog over a network via HTTP. In this chapter we describe how to program Stardog from Java using SNARL Stardog Native API for the RDF Language, Sesame, and Jena. We prefer SNARL to Sesame to Jena and recommend them—all other things being equal—in that order.

If you're a Spring developer, you might want to read Spring Programming (#_spring_programming) or if you prefer a ORM-style approach, you might want to checkout Empire (https://github.com/mhgrove/Empire), an implementation of JPA (http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html) for RDF that works with Stardog.

Examples

The **best** way to learn to program Stardog with Java is to study the examples:

1. SNARL (https://gist.github.com/1045573)
2. Sesame bindings (https://gist.github.com/1045568)
3. Jena bindings (https://gist.github.com/1045572)
4. SNARL and OWL 2 reasoning (https://gist.github.com/1045578)

5. SNARL and Connection Pooling (https://gist.github.com/1070230)
6. SNARL and Searching (https://gist.github.com/1085116)

We offer some commentary on the interesting parts of these examples below.

Creating & Administering Databases

`AdminConnection` provides simple programmatic access to all administrative functions available in Stardog.

Creating a Database

You can create a basic temporary memory database with Stardog with one line of code:

```
Unresolved directive in java.ad -
include::https://gist.githubusercontent.com/mhgrove/1333782/raw/CreateTempMemDb.java[]
```

WARNING It's **crucially important to always clean up connections to the database** by calling `AdminConnection#close().

You can also use the `memory` (/docs/5.3.6/java/snarl/com/complexible/stardog/api/admin/adminconnection#memory) and `disk` (/docs/5.3.6/java/snarl/com/complexible/stardog/api/admin/adminconnection#disk) functions to configure and create a database in any way you prefer. These methods return `DatabaseBuilder` (/docs/5.3.6/java/snarl/com/complexible/stardog/api/admin/databasebuilder) objects which you can use to configure the options of the database you'd like to create. Finally, the `create` (/docs/5.3.6/java/snarl/com/complexible/stardog/api/admin/databasebuilder#create) method takes the list of files to bulk load into the database when you create it and returns a valid `ConnectionConfiguration` (/docs/5.3.6/java/snarl/com/complexible/stardog/api/connectionconfiguration) which can be used to create new `Connections` (/docs/5.3.6/java/snarl/com/complexible/stardog/api/connection) to your database.

WARNING It is important to note that you **must** take care to always log out of the server when you are done working with `AdminConnection`.

```
Unresolved directive in java.ad -
include::https://gist.githubusercontent.com/mhgrove/1333782/raw/CreateMemSearchDb.java[]
```

This illustrates how to create a temporary memory database named `test` which supports full text search via Searching (#_searching).

```
Unresolved directive in java.ad -
include::https://gist.githubusercontent.com/mhgrove/1333782/raw/CreateDiskAndICV.java[]
```

This illustrates how to create a persistent disk database with ICV guard mode and reasoning enabled. For more information on what the available options for `set` are and what they mean, see the Database Admin (#_database_admin) section. Also note, Stardog database administration can be performed from the CLI (#_command_line_interface).

Creating a Connection String

As you can see, the `ConnectionConfiguration` (/docs/5.3.6/java/snarl/com/complexible/stardog/api/connectionconfiguration) in `com.complexible.stardog.api` (/docs/5.3.6/java/snarl/com/complexible/stardog/api/package-summary) package class is where the initial action takes place:

```
Unresolved directive in java.ad -
include::https://gist.githubusercontent.com/mhgrove/1045578/raw/L4044.java[]
```

The `to` (/docs/5.3.6/java/snarl/com/complexible/stardog/api/connectionconfiguration#to) method takes a `Database Name` as a string; and then `connect` (/docs/5.3.6/java/snarl/com/complexible/stardog/api/connectionconfiguration#connect) connects to the database using all specified properties on the configuration. This class and its constructor methods are used for **all** of Stardog's Java APIs: SNARL native Stardog API, Sesame, Jena, as well as HTTP. In the latter cases, you must also call `server` (/docs/5.3.6/java/snarl/com/complexible/stardog/api/connectionconfiguration#serverjava.lang.String) and pass it a valid URL to the Stardog server using HTTP.

Without the call to `server`, `ConnectionConfiguration` will attempt to connect to a local, embedded version of the Stardog server. The `Connection` still operates in the standard client-server mode, the only difference is that the server is running in the **same** JVM as your application.

NOTE Whether using SNARL, Sesame, or Jena, most perhaps all Stardog Java code will use `ConnectionConfiguration` to get a handle on a Stardog database—whether embedded or remote—and, after getting that handle, can use the appropriate API.

See the `ConnectionConfiguration` (/docs/5.3.6/java/snarl/com/complexible/stardog/api/connectionconfiguration) API docs or How to Make a Connection String (#_how_to_make_a_connection_string) for more information.

Managing Security

We discuss the security system in Stardog in Security (#_security). When logged into the Stardog DBMS (/docs/5.3.6/java/snarl/com/complexible/stardog/api/admin/adminconnection) you can access all security related features detailed in the security section using any of the core security interfaces for managing users (/docs/5.3.6/java/snarl/com/complexible/stardog/security/usermanager), roles (/docs/5.3.6/java/snarl/com/complexible/stardog/security/rolemanager), and permissions (/docs/5.3.6/java/snarl/com/complexible/stardog/security/permissionmanager).

Using SNARL

In examples 1 and 4 above, you can see how to use SNARL in Java to interact with Stardog. *The SNARL API will give the best performance overall and is the native Stardog API*. It uses some Sesame domain classes but is otherwise a clean-sheet API and implementation.

The SNARL API is fluent with the aim of making code written for Stardog easier to write and easier to maintain. Most objects are easily re-used to make basic tasks with SNARL as simple as possible. We are always interested in feedback on the API, so if you have suggestions or comments, please send them to the mailing list.

Let's take a closer look at some of the interesting parts of SNARL.

Adding Data

```
Unresolved directive in java.ad -
include::https://gist.githubusercontent.com/mhgrove/1045573/raw/SNARLAddData.java[]
```

You must always enclose changes to a database within a transaction begin and commit or rollback. Changes are local until the transaction is committed or until you try and perform a query operation to inspect the state of the database within the transaction.

By default, RDF added will go into the default context unless specified otherwise. As shown, you can use Adder (/docs/5.3.6/java/snarl/com/complexible/stardog/api/adder) directly to add statements and graphs to the database; and if you want to add data from a file or input stream, you use the `io` (/docs/5.3.6/java/snarl/com/complexible/stardog/api/io), `format`, and `stream` chain of method invocations.

See the SNARL API (java/snarl) Javadocs for all the gory details.

Removing Data

```
Unresolved directive in java.ad -
include::https://gist.githubusercontent.com/mhgrove/1045573/raw/SNARLRemoveData.java[]
```

Let's look at removing (/docs/5.3.6/java/snarl/com/complexible/stardog/api/remover) data via SNARL; in the example above, you can see that file or stream-based removal is symmetric to file or stream-based addition, i.e., calling `remove` in an `io` chain with a file or stream call. See the SNARL API docs for more details about finer-grained deletes, etc.

Parameterized SPARQL Queries

```
Unresolved directive in java.ad -
include::https://gist.githubusercontent.com/mhgrove/1045573/raw/SNARLQuery.java[]
```

SNARL also lets us parameterize SPARQL queries. We can make a `Query` object by passing a SPARQL query in the constructor. Simple. Obvious.

Next, let's set a limit for the results: `aQuery.limit10`; or if we want no limit, `aQuery.limitQuery.NO_LIMIT`. By default, there is no limit imposed on the query object; we'll use whatever is specified in the query. But you can use limit to override any limit specified in the query, however specifying NO_LIMIT will not remove a limit specified in a query, it will only remove any limit override you've specified, restoring the state to the default of using whatever is in the query.

We can execute that query with `executeSelect` and iterate over the results. We can also rebind the "?s" variable easily: `aQuery.parameter"s", aURI`, which will work for all instances of "?s" in any BGP in the query, and you can specify `null` to remove the binding.

Query objects are re-usable, so you can create one from your original query string and alter bindings, limit, and offset in any way you see fit and re-execute the query to get the updated results.

We **strongly** recommend the use of SNARL's parameterized queries over concatenating strings together in order to build your SPARQL query. This latter approach opens up the possibility for SPARQL injection attacks unless you are very careful in scrubbing your input.[39 (#_footnote_39)]

Getter Interface

```
Unresolved directive in java.ad -
include::https://gist.githubusercontent.com/mhgrove/1045573/raw/SNARLGetter.java[]
```

SNARL also supports some sugar for the classic statement-level `getSPO` --scars, anyone?--interactions. We ask in the first line of the snippet above for an iterator over the Stardog connection, based on `aURI` in the subject position. Then a while-loop, as one might expect…You can also parameterize `` Getter`s by binding different positions of the `Getter `` which acts like a kind of RDF statement filter—and then iterating as usual.

NOTE the `aIter.close` which is important for Stardog databases to avoid memory leaks. If you need to materialize the iterator as a graph, you can do that by calling `graph`.

The snippet doesn't show `object` or `context` parameters on a `Getter`, but those work, too, in the obvious way.

Reasoning

Stardog supports query-time reasoning (#_owl_rule_reasoning) using a query rewriting technique. In short, when reasoning is requested, a query is automatically rewritten to **n** queries, which are then executed. As we discuss below in Connection Pooling, reasoning is enabled at the `Connection` layer and then any queries executed over that connection are executed with reasoning enabled; you don't need to do anything up front when you create your database if you want to use reasoning.

```
Unresolved directive in java.ad -
include::https://gist.githubusercontent.com/mhgrove/1045578/raw/CreateReasoningConn.java[]
```

In this code example, you can see that it's trivial to enable reasoning for a `Connection`: simply call `reasoning` with `true` passed in.

Search

Stardog's search (#_searching) system can be used from Java. The fluent Java API for searching in SNARL looks a lot like the other search interfaces: We create a `Searcher` instance with a fluent constructor: `limit` sets a limit on the results; `query` contains the search query, and `threshold` sets a minimum threshold for the results.

```
Unresolved directive in java.ad -
include::https://gist.githubusercontent.com/mhgrove/1085116/raw/SearcherUsage.java[]
```

Then we call the `search` method of our `Searcher` instance and iterate over the results i.e., `SearchResults`. Last, we can use `offset` on an existing `Searcher` to grab another page of results.

Stardog also supports performing searches over the full-text index **within** a SPARQL query via the LARQ SPARQL syntax (http://jena.apache.org/documentation/larq/). This provides a powerful mechanism for querying both your RDF index and full-text index at the same time while also giving you a more performant option to the SPARQL `regex` filter.

User-defined Lucene Analyzer

Stardog's Semantic Search (#_searching) capability uses Lucene's default text analyzer (https://lucene.apache.org/core/4_7_2/analyzers-common/org/apache/lucene/analysis/standard/StandardAnalyzer.html), which may not be ideal for your data or application. You can implement a custom analyzer that Stardog will use by implementing `org.apache.lucene.analysis.Analyzer`. That lets you customize Stardog to support different natural languages, domain-specific stop word lists, etc.

See Custom Analyzers (https://github.com/complexible/stardog-examples/tree/master/examples/analyzer) in the stardog-examples Github repo for a complete description of the API, registry, sample code, etc.

User-defined Functions and Aggregates

Stardog may be extended via Function and Aggregate extensibility APIs, which are fully documented, including sample code, in the stardog-examples Github repo (https://github.com/complexible/stardog-examples/blob/master/examples/function/readme.md) section about function extensibility.

In short you can extend Stardog's SPARQL query evaluation with custom functions and aggregates easily. Function extensibility corresponds to built-in expressions used in `FILTER`, `BIND` and `SELECT` expressions, as well as aggregate operators in a SPARQL query like `COUNT` or `SAMPLE`.

SNARL Connection Views

SNARL `Connections` (/docs/5.3.6/java/snarl/com/complexible/stardog/api/connection#) support obtaining a specified type of `Connection`. This lets you extend and enhance the features available to a `Connection` while maintaining the standard, simple Connection API. The Connection `as` (/docs/5.3.6/java/snarl/com/complexible/stardog/api/connection#as) method takes as a parameter the interface, which must be a sub-type of a `Connection`, that you would like to use. `as` will either return the `Connection` as the view you've specified, or it will throw an exception if the view could not be obtained for some reason.

An example of obtaining an instance of a `SearchConnection` (/docs/5.3.6/java/snarl/com/complexible/stardog/api/search/searchconnection) to use Stardog's full-text search support would look like this:

```
Unresolved directive in java.ad -
include::https://gist.githubusercontent.com/mhgrove/1085116/raw/SearchConnectionView.java[]
```

SNARL API Docs

Please see SNARL API (java/snarl/) docs for more information.

Using Sesame

Stardog supports the Sesame API; thus, for the most part, using Stardog and Sesame is not much different from using Sesame with other RDF databases. There are, however, at least two differences worth pointing out.

Wrapping connections with StardogRepository

```
Unresolved directive in java.ad -
include::https://gist.githubusercontent.com/mhgrove/1045568/raw/init.java[]
```

As you can see from the code snippet, once you've created a `ConnectionConfiguration` with all the details for connecting to a Stardog database, you can wrap that in a `StardogRepository` which is a Stardog-specific implementation of the Sesame `Repository` interface. At this point, you can use the resulting `Repository` like any other Sesame `Repository` implementation. Each time you call `Repository.getConnection`, your original `ConnectionConfiguration` will be used to spawn a new connection to the database.

Autocommit

Stardog's `RepositoryConnection` implementation will, by default, disable `autoCommit` status. When enabled, every single statement added or deleted via the `Connection` will incur the cost of a transaction, which is too heavyweight for most use cases. You can enable `autoCommit` and it will work as expected; but **we recommend leaving it disabled**.

Using RDF4J

Stardog also supports RDF4J (http://rdf4j.org), the follow-up to Sesame. Its use is nearly identical to the Stardog Sesame API, mostly with package name updates.

Wrapping connections with StardogRepository

The RDF4J API uses `com.complexible.stardog.rdf4j.StardogRepository`, which works the same way as the Sesame `StardogRepository` mentioned above. Its constructor will take either a `ConnectionConfiguration` like Sesame's or a Connection String (#_how_to_make_a_connection_string).

Autocommit

The major difference between the RDF4J and Sesame APIs is that the RDF4J one will leave the `autoCommit` mode ON by default, instead of disabling it. This is because as of RDF4J's 2.7.0 release, they have deprecated the `setAutoCommit` method in favor of assuming it to be always on unless `begin()/commit()` are used, which we still **VERY highly recommend**.

Using Jena

Stardog supports Jena via a Sesame-Jena bridge, so it's got more overhead than Sesame or SNARL. YMMV. There are two points in the Jena example to emphasize.

Init in Jena

```
Unresolved directive in java.ad -
include::https://gist.githubusercontent.com/mhgrove/1045572/raw/InitJena.java[]
```

The initialization in Jena is a bit different from either SNARL or Sesame; you can get a Jena `Model` instance by passing the `Connection` instance returned by `ConnectionConfiguration` to the Stardog factory, `SDJenaFactory`.

Add in Jena

```
Unresolved directive in java.ad -
include::https://gist.githubusercontent.com/mhgrove/1045572/raw/AddToJena.java[]
```

Jena also wants to add data to a `Model` one statement at a time, which can be less than ideal. To work around this restriction, we recommend adding data to a `Model` in a single Stardog transaction, which is initiated with `aModel.begin`. Then to read data into the model, we recommend using RDF/XML, since that triggers the `BulkUpdateHandler` in Jena or grab a `BulkUpdateHandler` directly from the underlying Jena graph.

The other options include using the Stardog CLI (#_command_line_interface) client to bulk load a Stardog database or to use SNARL for loading and then switch to Jena for other operations, processing, query, etc.

Client-Server Stardog

Using Stardog from Java in either embedded or client-server mode is **very similar**--the only visible difference is the use of `url` in a `ConnectionConfiguration`: when it's present, we're in client-server model; else, we're in embedded mode.

That's a good and a bad thing: it's good because the code is symmetric and uniform. It's bad because it can make reasoning about performance difficult, i.e., it's not entirely clear in client-server mode which operations trigger or don't trigger a round trip with the server and, thus, which may be more expensive than they are in embedded mode.

In client-server mode, **everything triggers a round trip** with these exceptions:

- closing a connection outside a transaction
- any parameterizations or other of a query or getter instance
- any database state mutations in a transaction that don't need to be immediately visible to the transaction; that is, changes are sent to the server only when they are required, on commit, or on any query or read operation that needs to have the accurate up-to-date state of the data within the transaction.

Stardog generally tries to be as lazy as possible; but in client-server mode, since state is maintained on the client, there are fewer chances to be lazy and more interactions with the server.

Connection Pooling

Stardog supports connection pools for SNARL `Connection` objects for efficiency and programmer sanity. Here's how they work:

```
Unresolved directive in java.ad -
include::https://gist.githubusercontent.com/mhgrove/1070230/raw/JustCode.java[]
```

Per standard practice, we first initialize security and grab a connection, in this case to the `testConnectionPool` database. Then we setup a `ConnectionPoolConfig`, using its fluent API, which establishes the parameters of the pool:

`using`	Sets which ConnectionConfiguration we want to pool; this is what is used to actually create the connections.
`minPool`, `maxPool`	Establishes min and max pooled objects; max pooled objects includes both leased and idled objects.
`expiration`	Sets the idle life of objects; in this case, the pool reclaims objects idled for 1 hour.
`blockAtCapacity`	Sets the max time in minutes that we'll block waiting for an object when there aren't any idle ones in the pool.

Whew! Next we can `create` the pool using this `ConnectionPoolConfig` thing.

Finally, we call `obtain` on the `ConnectionPool` when we need a new one. And when we're done with it, we return it to the pool so it can be re-used, by calling `release`. When we're done, we `shutdown` the pool.

Since reasoning (#_owl_rule_reasoning) in Stardog is enabled per `Connection`, you can create two pools: one with reasoning connections, one with non-reasoning connections; and then use the one you need to have reasoning **per query**; never pay for more than you need.

API Deprecation

Methods and classes in SNARL API that are marked with the `com.google.common.annotations.Beta` are subject to change or removal in any release. We are using this annotation to denote new or experimental features, the behavior or signature of which may change significantly before it's out of "beta".

We will otherwise attempt to keep the public APIs as stable as possible, and methods will be marked with the standard `@Deprecated` annotation for a least one full revision cycle before their removal from the SNARL API. See Compatibility Policies (#_compatibility_policies) for more information about API stability.

Anything marked `@VisibleForTesting` is just that, visible as a consequence of test case requirements; don't write any important code that depends on functions with this annotation.

Using Maven

As of Stardog 3.0, we support Maven for both client and server JARs. The following table summarizes the type of dependencies that you will have to include in your project, depending on whether the project is a Stardog client, or server, or both. Additionally, you can also include the Jena or Sesame bindings if you would like to use them in your project. The Stardog dependency list below follows the Gradle (http://www.gradle.org) convention and is of the form: `groupId:artifactId:VERSION`. Versions 3.0 and higher are supported.

9. Table of client type dependencies

Type	Stardog Dependency	Type
client	`com.complexible.stardog:client-http:VERSION`	pom
server	`com.complexible.stardog:server:VERSION`	pom
rdf4j	`com.complexible.stardog.rdf4j:stardog-rdf4j:VERSION`	jar
sesame	`com.complexible.stardog.sesame:stardog-sesame-core:VERSION`	jar
jena	`com.complexible.stardog.jena:stardog-jena:VERSION`	jar
gremlin	`com.complexible.stardog.gremlin:stardog-gremlin:VERSION`	jar

You can see an example of their usage in our examples repository on Github (https://github.com/complexible/stardog-examples/blob/628cf3dab2/examples/api/build.gradle#L3-L14).

WARNING If you're using Maven as your build tool, then `client-http` and `server` dependencies require that you specify the packaging type **as POM** (`pom`):

```
<dependency>
  <groupId>com.complexible.stardog</groupId>
  <artifactId>client-http</artifactId>
  <version>$VERSION</version>
  <type>pom</type> (1)
</dependency>
```

1. **The dependency type must be set to `pom`.**

Note: Though Gradle may still work without doing this, it is still best practice to specify the dependency type there as well:

```
compile "com.complexible.stardog:client-http:${VERSION}@pom"
```

Public Maven Repo

The public Maven repository for the current Stardog release is http://maven.stardog.com (http://maven.stardog.com). To get started, you need to add the following endpoint to your preferred build system, e.g. in your Gradle build script:

```
repositories {
  maven {
    url "http://maven.stardog.com"
  }
}
```

Similarly, if you’re using Maven you’ll need to add the following to your Maven `pom.xml`:

```
<repositories>
   <repository>
     <id>stardog-public</id>
     <url>http://maven.stardog.com</url>
   </repository>
</repositories>
```

Private Maven Repo

CUSTOMER ACCESS

This feature or service is available to Stardog customers. For information about licensing, please email (mailto:sales@stardog.com) us.

For access to nightly builds, priority bug fixes, priority feature access, hot fixes, etc. Enterprise Premium Support customers have access to their own private Maven repository that is linked to our internal development repository. We provide a private repository which you can either proxy from your preferred Maven repository manager—e.g. Artifactory or Nexus—or add the private endpoint to your build script.

Connecting to Your Private Maven Repo

Similar to our public Maven repo, we will provide you with a private URL and credentials to your private repo, which you will refer to in your Gradle build script like this:

```
repositories {
  maven {
    url $yourPrivateUrl
       credentials {
          username $yourUsername
          password $yourPassword
        }
    }
}
```

Or if you’re using Maven, add the following to your `pom.xml`:

```
<repositories>
   <repository>
      <id>stardog-private</id>
      <url>$yourPrivateUrl</url>
   </repository>
</repositories>
```

Then in your `~/.m2/settings.xml` add:

```
<settings>
  <servers>
    <server>
      <id>stardog-private</id>
      <username>$yourUsername</username>
      <password>$yourPassword</password>
    </server>
  </servers>
</settings>
```

NETWORK PROGRAMMING

In the Java Programming (#_java_programming) section, we consider interacting with Stardog programmatically from a Java program. In this section we consider interacting with Stardog over HTTP. In some use cases or deployment scenarios, it may be necessary to interact with or control Stardog remotely over an IP-based network.

Stardog supports SPARQL 1.0 HTTP Protocol (http://www.w3.org/TR/rdf-sparql-protocol/); the SPARQL 1.1 Graph Store HTTP Protocol (http://www.w3.org/TR/sparql11-http-rdf-update/); the Stardog HTTP Protocol; and SNARL, an RPC-style protocol based on Google Protocol Buffers (http://code.google.com/apis/protocolbuffers/).

SPARQL Protocol

Stardog supports the standard SPARQL Protocol HTTP bindings, as well as additional functionality via HTTP. Stardog also supports SPARQL 1.1's Service Description format. See the spec (https://www.w3.org/TR/sparql11-service-description/) if you want details.

Stardog HTTP Protocol

The Stardog HTTP Protocol supports SPARQL Protocol 1.1 and additional resource representations and capabilities. The Stardog HTTP API v4 is also available on Apiary: http://docs.stardog.apiary.io/ (http://docs.stardog.apiary.io/). The Stardog Linked Data API (aka "Annex") is also documented on Apiary: http://docs.annex.apiary.io/ (http://docs.annex.apiary.io/).

Generating URLs

If you are running the HTTP server at

```
http://localhost:12345/
```

To form the URI of a particular Stardog Database, the Database Short Name is the first URL path segment appended to the deployment URI. For example, for the Database called `cytwombly`, deployed in the above example HTTP server, the Database Network Name might be

```
http://localhost:12345/cytwombly
```

All the resources related to this database are identified by URL path segments relative to the Database Network Name; hence:

```
http://localhost:12345/cytwombly/size
```

In what follows, we use URI Template (http://code.google.com/p/uri-templates/)) notation to parameterize the actual request URLs, thus: `/{db}/size`.

We also abuse notation to show the permissible HTTP request types and default MIME types in the following way: `REQ | REQ /resource/identifier → mime_type | mime_type`. In a few cases, we use `void` as short hand for the case where there is a response code but the response body may be empty.

HTTP Headers: Content-Type & Accept

All HTTP requests that are mutative (add or remove) must include a valid `Content-Type` header set to the MIME type of the request body, where "valid" is a valid MIME type for N-Triples, Trig, Trix, Turtle, NQuads, JSON-LD, or RDF/XML:

RDF/XML	`application/rdf+xml`
Turtle	`application/x-turtle` or `text/turtle`
N-Triples	`application/n-triples`

TriG	`application/trig`
TriX	`application/trix`
N-Quads	`application/n-quads`
JSON-LD	`application/ld+json`

SPARQL `CONSTRUCT` queries must also include a `Accept` header set to one of these RDF serialization types.

When issuing a `SELECT` query the `Accept` header should be set to one of the valid MIME types for `SELECT` results:

SPARQL XML Results Format	`application/sparql-results+xml`
SPARQL JSON Results Format	`application/sparql-results+json`
SPARQL Boolean Results	`text/boolean`
SPARQL Binary Results	`application/x-binary-rdf-results-table`

Response Codes

Stardog uses the following HTTP response codes:

`200`	Operation has succeeded.
`202`	Operation was received successfully and will be processed shortly.
`400`	Indicates parse errors or that the transaction identifier specified for an operation is invalid or does not correspond to a known transaction.
`401`	Request is unauthorized.
`403`	User attempting to perform the operation does not exist, their username or password is invalid, or they do not have the proper credentials to perform the action.
`404`	A resource involved in the request—for example the database or transaction—does not exist.
`409`	A conflict for some database operations; for example, creating a database that already exists.
`500`	A unspecified failure in some internal operation…Call your office, Senator!

There are also Stardog-specific error codes in the `SD-Error-Code` header in the response from the server. These can be used to further clarify the reason for the failure on the server, especially in cases where it could be ambiguous. For example, if you received a `404` from the server trying to commit a transaction denoted by the path `/myDb/transaction/commit/293845klf9f934`…it's probably not clear what is missing: it's either the transaction or the database. In this case, the value of the `SD-Error-Code` header will clarify.

The enumeration of `SD-Error-Code` values and their meanings are as follows:

`0`	Authentication error
`1`	Authorization error
`2`	Query evaluation error
`3`	Query contained parse errors
`4`	Query is unknown
`5`	Transaction not found
`6`	Database not found
`7`	Database already exists
`8`	Database name is invalid
`9`	Resource (user, role, etc) already exists
`10`	Invalid connection parameter(s)
`11`	Invalid database state for the request
`12`	Resource in use
`13`	Resource not found
`14`	Operation not supported by the server
`15`	Password specified in the request was invalid

In cases of error, the message body of the result will include any error information provided by the server to indicate the cause of the error.

Stardog Resources

To interact with Stardog over HTTP, use the following resource representations, HTTP response codes, and resource identifiers.

A Stardog Database

```
GET /{db} → void
```

Returns a representation of the database. As of Stardog 5.3.6, this is merely a placeholder; in a later release, this resource will serve the web console where the database can be interacted with in a browser.

Database Size

```
GET /{db}/size → text/plain
```

Returns the number of RDF triples in the database.

Query Evaluation

```
GET | POST /{db}/query
```

The SPARQL endpoint for the database. The valid Accept types are listed above in the HTTP Headers (#_http_headers_content_type_accept) section.

To issue SPARQL queries with reasoning over HTTP, see Using Reasoning (#_using_reasoning).

SPARQL update

```
GET | POST /{db}/update → text/boolean
```

The SPARQL endpoint for updating the database with SPARQL Update. The valid Accept types are `application/sparql-update` `or application/x-www-form-urlencoded`. Response is the result of the update operation as text, eg `true` or `false`.

Query Plan

```
GET | POST /{db}/explain → text/plain
```

Returns the explanation for the execution of a query, i.e., a query plan. All the same arguments as for Query Evaluation are legal here; but the only MIME type for the Query Plan resource is `text/plain`.

Transaction Begin

```
POST /{db}/transaction/begin → text/plain
```

Returns a transaction identifier resource as `text/plain`, which is likely to be deprecated in a future release in favor of a hypertext format. `POST` to begin a transaction accepts neither body nor arguments.

Transaction Security Considerations

WARNING Stardog's implementation of transactions with HTTP is vulnerable to man-in-the-middle attacks, which could be used to violate Stardog's isolation guarantee (among other nasty side effects).

Stardog's transaction identifiers are 64-bit GUIDs and, thus, pretty hard to **guess**; but if you can grab a response in-flight, you can steal the transaction identifier if basic access auth or RFC 2069 digest auth is in use. **You've been warned.**

In a future release, Stardog will use RFC 2617 HTTP Digest Authentication (http://tools.ietf.org/html/rfc2617), which is less vulnerable to various attacks and will never ask a client to use a different authentication type, which should lessen the likelihood of MitM attacks for properly restricted Stardog clients—that is, a Stardog client that treats any request by a proxy server or origin server (i.e., Stardog) to use basic access auth or RFC 2069 digest auth as a MitM attack. See RFC 2617 (http://tools.ietf.org/html/rfc2617) for more information.

Transaction Commit

```
POST /{db}/transaction/commit/{txId} → void | text/plain
```

Returns a representation of the committed transaction; `200` means the commit was successful. Otherwise a `500` error indicates the commit failed and the text returned in the result is the failure message.

As you might expect, failed commits exit cleanly, rolling back any changes that were made to the database.

Transaction Rollback

```
POST /{db}/transaction/rollback/{txId} → void | text/plain
```

Returns a representation of the transaction after it's been rolled back. `200` means the rollback was successful, otherwise `500` indicates the rollback failed and the text returned in the result is the failure message.

Querying (Transactionally)

```
GET | POST /{db}/{txId}/query
```

Returns a representation of a query executed within the `txId` transaction. Queries within transactions will be slower as extra processing is required to make the changes visible to the query. Again, the valid Accept types are listed above in the `HTTP Headers` section.

```
GET | POST /{db}/{txId}/update → text/boolean
```

The SPARQL endpoint for updating the database with SPARQL Update. Update queries are executed within the specified transaction `txId` and are **not** atomic operations as with the normal SPARQL update endpoint. The updates are executed when the transaction is committed like any other change. The valid Accept types are `application/sparql-update` `or application/x-www-form-urlencoded`. Response is the result of the update operation as text, eg `true` or `false`.

Adding Data (Transactionally)

```
POST /{db}/{txId}/add → void | text/plain
```

Returns a representation of data added to the database of the specified transaction. Accepts an optional parameter, `graph-uri`, which specifies the named graph the data should be added to. If a named graph is not specified, the data is added to the default (i.e., unnamed) context. The response codes are `200` for success and `500` for failure.

Deleting Data (Transactionally)

```
POST /{db}/{txId}/remove → void | text/plain
```

Returns a representation of data removed from the database within the specified transaction. Also accepts `graph-uri` with the analogous meaning as above--Adding Data (Transactionally) (#_adding_data_transactionally). Response codes are also the same.

Clear Database

```
POST /{db}/{txId}/clear → void | text/plain
```

Removes all data from the database within the context of the transaction. `200` indicates success; `500` indicates an error. Also takes an optional parameter, `graph-uri`, which removes data from a named graph. To clear only the default graph, pass `DEFAULT` as the value of `graph-uri`.

Export Database

```
GET /{db}/export → RDF
```

Exports the default graph in the database in Turtle format. Also takes an optional parameter, `graph-uri`, which selects a named graph to export. The valid Accept types are the ones defined above in HTTP Headers (#_http_headers_content_type_accept) for RDF Formats.

Explanation of Inferences

```
POST /{db}/reasoning/explain → RDF
POST /{db}/reasoning/{txId}/explain → RDF
```

Returns the explanation of the axiom which is in the body of the `POST` request. The request takes the axioms in any supported RDF format and returns the explanation for why that axiom was inferred as Turtle.

Explanation of Inconsistency

```
GET | POST /{db}/reasoning/explain/inconsistency → RDF
```

If the database is logically inconsistent, this returns an explanation for the inconsistency.

Consistency

```
GET | POST /{db}/reasoning/consistency → text/boolean
```

Returns whether or not the database is consistent w.r.t to the TBox.

Listing Integrity Constraints

```
GET /{db}/icv → RDF
```

Returns the integrity constraints for the specified database serialized in any supported RDF format.

Adding Integrity Constraints

```
POST /{db}/icv/add
```

Accepts a set of valid Integrity constraints serialized in any RDF format supported by Stardog and adds them to the database in an atomic action. 200 return code indicates the constraints were added successfully, 500 indicates that the constraints were not valid or unable to be added.

Removing Integrity Constraints

```
POST /{db}/icv/remove
```

Accepts a set of valid Integrity constraints serialized in any RDF format supported by Stardog and removes them from the database in a single atomic action. `200` indicates the constraints were successfully remove; `500` indicates an error.

Clearing Integrity Constraints

```
POST /{db}/icv/clear
```

Drops **all** integrity constraints for a database. `200` indicates all constraints were successfully dropped; `500` indicates an error.

Converting Constraints to SPARQL Queries

```
POST /{db}/icv/convert
```

The body of the `POST` is a single integrity constraint, serialized in any supported RDF format, with `Content-type` set appropriately. Returns either a `text/plain` result containing a single SPARQL query; or it returns `400` if more than one constraint was included in the input.

Execute GraphQL Query

```
GET | POST /{db}/graphql → application/json
```

Executes a GraphQL query. The `GET` request accepts a `query` variable which should be a GraphQL query and an optional `variables` property that is a JSON document for representing input variable bindings. The body of the `POST` request should be a JSON document with `query` and (optionally) `variables` fields. Reasoning can be enabled by setting the `@reasoning` variable to `true` in the `variables`. A schema for the query can be used by setting the `@schema` variable to the name of the schema.

Adding GraphQL Schemas

```
PUT /{db}/graphql/schemas/{schema}
```

Adds a GraphQL schema to the database. The name of the schema is specified by the path variable `schema`.

Getting GraphQL Schemas

```
GET /{db}/graphql/schemas/{schema} → application/graphql
```

Returns the contents of the specified GraphQL schema. The name of the schema is specified by the path variable `schema`. The response is the GraphQL schema document.

Removing GraphQL Schemas

```
DELETE /{db}/graphql/schemas/{schema}
```

Removes a GraphQL schema from the database. The name of the schema is specified by the path variable `schema`.

Removing All GraphQL Schemas

```
DELETE /{db}/graphql/schemas
```

Removes all GraphQL schemas from the database.

List GraphQL Schemas

```
GET /{db}/graphql/schemas → application/json
```

Lists all the GraphQL schemas in the database. The response is a JSON document where the `schemas` field is a list of schema names.

Admin Resources

To administer Stardog over HTTP, use the following resource representations, HTTP response codes, and resource identifiers.

List databases

```
GET /admin/databases → application/json
```

Lists all the databases available.

Output JSON example:

```
{ "databases" : ["testdb", "exampledb"] }
```

Copy a database

```
PUT /admin/databases/{db}/copy?to={db_copy}
```

Copies a database `db` to another specified `db_copy`.

Create a new database

```
POST /admin/databases
```

Creates a new database; expects a multipart request with a JSON specifying database name, options and filenames followed by (optional) file contents as a multipart `POST` request.

Expected input (`application/json`):

```
{
  "dbname" : "testDb",
  "options" : {
    "icv.active.graphs" : "http://graph, http://another",
    "search.enabled" : true,
    ...
  },
  "files" : [{ "filename":"fileX.ttl", "context":"some:context" }, ...]
}
```

Drop an existing database

```
DELETE /admin/databases/{db}
```

Drops an existing database `db` and all the information that it contains. Goodbye Callahan!

Repair a database

```
PUT /admin/databases/{db}/repair
```

Repairs a corrupted Stardog database. This command needs a running Stardog server and the database to be offline.

Backup a database

```
PUT /admin/databases/{db}/backup[?to={backup_location}]
```

Creates a backup of a database. A backup is a physical copy of the database and preserves database metadata in addition to the database contents. By default, backups are stored in the '.backup' directory in your Stardog home (or the 'backup.dir' property specified in your 'stardog.configuration') but the `to` parameter can be used to specify a different location.

Restore a database from a backup

```
PUT /admin/databases?from={backup_location}[&name={new_name}][&force={true|false}]
```

Restores a database from its backup. The location of the backup should be the full path to the backup on the server side. If you wish to restore the backup to a different database, a new name can be provided. A backup will not be restored over an existing database of the same name; the force flag should be used to overwrite the database.

Optimize a database

```
PUT /admin/databases/{db}/optimize
```

Optimizes a database for query answering after a database has been heavily modified.

Sets an existing database online.

```
PUT /admin/databases/{db}/online
```

Request message to set an existing database online.

Sets an existing database offline.

```
PUT /admin/databases/{db}/offline
```

Request message to set an existing database offline; receives optionally a JSON input to specify a timeout for the offline operation. When not specified, defaults to 3 minutes as the timeout; the timeout should be provided in **milliseconds**. The timeout is the amount of time the database will wait for existing connections to complete before going offline. This will allow open transaction to commit/rollback, open queries to complete, etc. After the timeout has expired, all remaining open connections are closed and the database goes offline.

Optional input (`application/json`):

```
{ "timeout" : timeout_in_ms}
```

Set option values to an existing database.

```
POST /admin/databases/{kb}/options
```

Set options in the database passed through a JSON object specification, i.e. JSON Request for option values. Database options can be found here (#_configuring_a_database).

Expected input (`application/json`):

```
{
  "database.name" : "DB_NAME",
  "icv.enabled" : true | false,
  "search.enabled" : true | false,
  ...
}
```

Get option values of an existing database.

```
PUT /admin/databases/{kb}/options → application/json
```

Retrieves a set of options passed via a JSON object. The JSON input has empty values for each key, but will be filled with the option values in the database in the output.

Expected input:

```
{
  "database.name" : ...,
  "icv.enabled" : ...,
  "search.enabled" : ...,
  ...
}
```

Output JSON example:

```
{
  "database.name" : "testdb",
  "icv.enabled" : true,
  "search.enabled" : true,
  ...
}
```

Add a new user to the system.

```
POST /admin/users
```

Adds a new user to the system; allows a configuration option for superuser as a JSON object. Superuser configuration is set as default to false. The password **must** be provided for the user.

Expected input:

```
{
  "username"  : "bob",
  "superuser" : true | false
  "password"  : "passwd"
}
```

Change user password.

```
PUT /admin/users/{user}/pwd
```

Changes user's password in the system. Receives input of new password as a JSON Object.

Expected input:

```
{"password" : "xxxxx"}
```

Check if user is enabled.

```
GET /admin/users/{user}/enabled → application/json
```

Verifies if user is enabled in the system.

Output JSON example:

```
{
  "enabled": true
}
```

Check if user is superuser.

```
GET /admin/users/{user}/superuser → application/json
```

Verifies if the user is a superuser:

```
{
  "superuser": true
}
```

Listing users.

```
GET /admin/users → application/json
```

Retrieves a list of users.

Output JSON example:

```
{
  "users": ["anonymous", "admin"]
}
```

Listing user roles.

```
GET /admin/users/{user}/roles → application/json
```

Retrieves the list of the roles assigned to user.

Output JSON example:

```
{
  "roles": ["reader"]
}
```

Deleting users.

```
DELETE /admin/users/{user}
```

Removes a user from the system.

Enabling users.

```
PUT /admin/users/{user}/enabled
```

Enables a user in the system; expects a JSON object in the following format:

```
{
  "enabled" : true
}
```

Setting user roles.

```
PUT /admin/users/{user}/roles
```

Sets roles for a given user; expects a JSON object specifying the roles for the user in the following format:

```
{
  "roles" : ["reader","secTestDb-full"]
}
```

Adding new roles.

```
POST /admin/roles
```

Adds the new role to the system.

Expected input:

```
{
  "rolename" : ""
}
```

Listing roles.

```
GET /admin/roles → application/json
```

Retrieves the list of roles registered in the system.

Output JSON example:

```
{
  "roles": ["reader"]
}
```

Listing users with a specified role.

```
GET /admin/roles/{role}/users → application/json
```

Retrieves users that have the role assigned.

Output JSON example:

```
{
  "users": ["anonymous"]
}
```

Deleting roles.

```
DELETE /admin/roles/{role}?force={force}
```

Deletes an existing role from the system; the force parameter is a boolean flag which indicates if the delete call for the role must be forced.

Assigning permissions to roles.

```
PUT /admin/permissions/role/{role}
```

Creates a new permission for a given role over a specified resource; expects input JSON Object in the following format:

```
{
  "action" : "read" | "write" | "create" | "delete" | "revoke" | "execute" | "grant" | "*",
  "resource_type" : "user" | "role" | "db" | "named-graph" | "metadata" | "admin" | "icv-constraints" |
"*",
  "resource" : ""
}
```

Assigning permissions to users.

```
PUT /admin/permissions/user/{user}
```

Creates a new permission for a given user over a specified resource; expects input JSON Object in the following format:

```
{
  "action" : "read" | "write" | "create" | "delete" | "revoke" | "execute" | "grant" | "*",
  "resource_type" : "user" | "role" | "db" | "named-graph" | "metadata" | "admin" | "icv-constraints" |
"*",
  "resource" : ""
}
```

Deleting permissions from roles.

```
POST /admin/permissions/role/{role}/delete
```

Deletes a permission for a given role over a specified resource; expects input JSON Object in the following format:

```
{
  "action" : "read" | "write" | "create" | "delete" | "revoke" | "execute" | "grant" | "*",
  "resource_type" : "user" | "role" | "db" | "named-graph" | "metadata" | "admin" | "icv-constraints" |
"*",
  "resource" : ""
}
```

Deleting permissions from users.

```
POST /admin/permissions/user/{user}/delete
```

Deletes a permission for a given user over a specified resource; expects input JSON Object in the following format:

```
{
  "action" : "read" | "write" | "create" | "delete" | "revoke" | "execute" | "grant" | "*",
  "resource_type" : "user" | "role" | "db" | "named-graph" | "metadata" | "admin" | "icv-constraints" |
"*",
  "resource" : ""
}
```

Listing role permissions.

```
GET /admin/permissions/role/{role} → application/json
```

Retrieves permissions assigned to the role.

Output JSON example:

```
{
  "permissions": ["stardog:read:*"]
}
```

Listing user permissions.

```
GET /admin/permissions/user/{user} → application/json
```

Retrieves permissions assigned to the user.

Output JSON example:

```
{
  "permissions": ["stardog:read:*"]
}
```

Listing user effective permissions.

```
GET /admin/permissions/effective/user/{user} → application/json
```

Retrieves effective permissions assigned to the user.

Output JSON example:

```
{
  "permissions": ["stardog:*"]
}
```

Get nodes in cluster

```
GET /admin/cluster
```

Retrieves the list of nodes in the cluster; the elected cluster coordinator is the first element in the array. This route is only available when Stardog is running within a cluster setup.

Output JSON example:

```
{
  "nodes": [
    "192.168.69.1:5820 (Available)",
    "192.168.69.2:5820 (Available)",
    "192.168.69.3:5820 (Available)"
  ]
}
```

Shutdown server.

```
POST /admin/shutdown
```

Shuts down the Stardog Server. If successful, returns a `202` to indicate that the request was received and that the server will be shut down shortly.

Query Version Metadata

```
GET | POST /{db}/vcs/query
```

Issue a query over the version history metadata using SPARQL. Method has the same arguments and outputs as the normal query method of a database.

Versioned Commit

```
POST /{db}/vcs/{tid}/commit_msg
```

Input example:

```
This is the commit message
```

Accepts a commit message in the body of the request and performs a VCS commit of the specified transaction

Create Tag

```
POST /{db}/vcs/tags/create
```

Input example:

```
"f09c0e02350627480839da4661b8e9cbd70f6372", "This is the commit message"
```

Create a tag from the given revision id with the specified commit message.

Delete Tag

```
POST /{db}/vcs/tags/delete
```

Input example:

```
"f09c0e02350627480839da4661b8e9cbd70f6372"
```

Delete the tag with the given revision.

Revert to Tag

```
POST /{db}/vcs/revert
```

Input example:

```
"f09c0e02350627480839da4661b8e9cbd70f6372", "893220fba7910792084dd85207db94292886c4d7", "This is the revert
message"
```

Perform a revert of a revision to the specified revision with the given commit message.

EXTENDING STARDOG

In this chapter we discuss the various ways you can extend the Stardog Knowledge Graph Platform. Stardog's extension mechanisms utilize the JDK Service Loader (http://docs.oracle.com/javase/6/docs/api/java/util/ServiceLoader.html) to load new services at runtime and make them available to the various parts of the system.

To register an extension, a file should be placed in `META-INF/services` whose name is the **fully-qualified** of the extension type. This **must** be included in the jar file with the compile source of the extension. The jar then should be placed somewhere in Stardog's classpath, usually either `server/ext` or a folder specified by the environment variable `STARDOG_EXT`. Stardog will pick up the implementations on startup by using the JDK `ServiceLoader` framework.

HTTP Server

Your extension to the server should extend `com.stardog.http.server.undertow.HttpService`. You'll need a service definition in `META-INF/services` called `com.stardog.http.server.undertow.HttpService`, which contains the fully-qualified class name(s) of your extension(s). That service definition should be included in your jar file and dropped into the classpath. On (re)start, Stardog will find the service and auto load it into the server.

An `HttpService` uses a subset of the JAX-RS (https://jax-rs-spec.java.net/) specification for route definition. All `` HttpService`s are scanned when the server starts and the routes are extracted, compiled into lambdas to avoid the overhead of `java.lang.reflect `` and passed into the server for route handling.

For example, if you wanted to define a route that will accept only `POST` requests whose body is JSON and produces binary output, it would look like this:

```
@POST
@Path("/path/to/my/service")
@Consumes("application/json")
@Produces("application/octet-stream")
public void myMethod(final HttpServerExchange theExchange) {
    // implementation goes here
}
```

There's a couple things to note here. First, the path of the route is partially defined by the `@Path` annotation. That value is post-fixed to any root `@Path` specified on the service itself. That complete path is normally with respect to the root of the server. However, services can have sub-services, in which case it would be relative to the parent service, or you can simply override this altogether by overriding the `routes` method. Similarily, you can override the `routes` method if you would like to define child services for the route. Here's how the transaction service mounts the SPARQL protocol and SPARQL Update protocol as child services:

```
public Iterable<Route> routes(@Nonnull final HttpPath thePath) {
        List<HttpService> aServices = Lists.newArrayList(
                new SPARQLProtocol(mKernel),
            new SPARQLUpdate(mKernel)
        );

        final HttpPath aRoot = thePath.var(VAR_TX);

        return () -> Iterators.concat(aServices.stream()
                                               .flatMap(aService -> Streams.stream(aService.routes(aRoot)))
                                               .iterator(),
                                    HttpService.super.routes(thePath).iterator());
}
```

Further, note that the route method takes a *single* parameter: `HttpServerExchange`. This is the raw Undertow HttpServerExchange (http://undertow.io/javadoc/1.4.x/io/undertow/server/HttpServerExchange.html). No attempt is made to parse out arguments such as `javax.ws.rs.QueryParam`, that is left to the implementor. The only part of the exchange that is parsed is any path variables. If your path is `/myservice/{db}/myaction` then `db` is a variable and will match whatever is included in that segment of the path. This is included in `HttpServerExchange#getQueryParameters`.

An `HttpService` should either have a no-argument constructor, **or** a constructor that accepts a single argument of type `HttpServiceLoader.ServerContext`. The `ServerContext` argument will contain information about the server: initialization options such as port, or whether security is disabled, as well as a reference to Stardog itself in the event your service needs a handle to Stardog. If your service is extending Stardog in some way, for convenience, you can extend from `KernelHttpService`.

The implementation of the route itself should conform to Undertow's guide (http://undertow.io/undertow-docs/undertow-docs-1.4.0/index.html#undertow-handler-authors-guide).

Here's the implementation of `user list`:

```
@GET
@Produces("application/json")
public void listUsers(final HttpServerExchange theExchange) {
        JsonArray aUsers = new JsonArray();

        mKernel.get().getUserManager().getAllUsers().stream()
               .map(JsonPrimitive::new)
               .forEach(aUsers::add);

        JsonObject aObj = new JsonObject();
        aObj.add(HTTPAdminProtocolConsts.COLLECTION_USERS, aUsers);

        final String aJSON = aObj.toString();

        HttpServerExchanges.Responses.contentType(theExchange, "application/json");
        HttpServerExchanges.Responses.contentLength(theExchange, aJSON.length());

        theExchange.getResponseSender().send(aJSON);
}
```

Query Functions

The Stardog com.complexible.stardog.plan.filter.functions.Function (/docs/5.3.6/java/snarl/com/complexible/stardog/plan/filter/functions/function) interface is the extension point for section 17.6 (Extensible Value Testing) of the SPARQL spec (https://www.w3.org/TR/sparql11-query/#extensionFunctions).

`Function` corresponds to built-in expressions used in `FILTER`, `BIND` and `SELECT` expressions, as well as aggregate operators in a SPARQL query. Examples include `&&` and `||` and functions defined in the SPARQL spec like `sameTerm`, `str`, and `now`.

Implementing Custom Functions

The starting point for implementing your own custom function is to extend AbstractFunction (/docs/5.3.6/java/snarl/com/complexible/stardog/plan/filter/functions/abstractfunction). This class provides much of the basic scaffolding for implementing a new `Function` from scratch.

If your new function falls into one of the existing categories, it should implement the appropriate marker interface:

- `com.complexible.stardog.plan.filter.functions.cast.CastFunction`
- `com.complexible.stardog.plan.filter.functions.datetime.DateTimeFunction`
- `com.complexible.stardog.plan.filter.functions.hash.HashFunction`
- `com.complexible.stardog.plan.filter.functions.numeric.MathFunction`
- `com.complexible.stardog.plan.filter.functions.rdfterm.RDFTermFunction`
- `com.complexible.stardog.plan.filter.functions.string.StringFunction`

If not, then it **must** implement `com.complexible.stardog.plan.filter.functions.UserDefinedFunction`. Extending one of these marker interfaces is required for the `Function` to be traverseable via the visitor pattern.

A zero-argument constructor **must** be provided which delegates some initialization to `super`, providing first the `int` number of required arguments followed by one or more URIs which identify the function. *Any* these URIs can be used to identify the function in a SPARQL query. The URIs are typed as `String` but **should** be valid URIs.

For functions which take a range of arguments, for example a minimum of 2, but no more than 4 values, a Range (http://docs.guava-libraries.googlecode.com/git-history/release/javadoc/com/google/common/collect/Range.html) can be used as the first parameter passed to `super` rather than an `int`.

`Function` extends from `Copyable`, therefore implementations should also provide a "copy constructor" which can be called from the `copy` method:

```
private MyFunc(final MyFunc theFunc) {
    super(myFunc);
    // make copies of any local data structures
}

@Override
public MyFunc copy() {
        return new MyFunc(this);
}
```

Evaluating the function is handled by `Value internalEvaluate(final Value…)` The parameters of this method correspond to the arguments passed into the function; it's the values of the variables for each solution of the query. Here we can perform whatever actions are required for our function. `AbstractFunction` will have already taken care of validating that we're getting the correct *number* of arguments to the function, but we still have to validate the input. `AbstractFunction` provides some convenience methods to this end, for example `assertURI` and `assertNumericLiteral` for requiring that inputs are either a valid URI, or a literal with a numeric datatype respectively.

Errors that occur in the evaluation of the function should throw a `com.complexible.stardog.plan.filter.ExpressionEvaluationException`; this corresponds to the `ValueError` concept defined in the SPARQL specification.

Registering Custom Functions

Create a file called `com.complexible.stardog.plan.filter.functions.Function` in the `META-INF/services` directory with the name of your custom Function class.

Using Custom Functions

Functions are identified by their URI; you can reference them in a query using their fully-qualified URI, or specify prefixes for the namespaces and utilize only the qname. For this example, if the namespace `tag:stardog:api:` is associated with the prefix `stardog` and within that namespace we have our function `myFunc` we can invoke it from a SPARQL query as: `bind(stardog:myFunc(?var) as ?tc)`

Custom Aggregates

While the SPARQL specification has an extension point for value testing and allows for custom functions in `FILTER`/`BIND`/`SELECT` expressions, there is no similar mechanism for aggregates. The space of aggregates is closed by definition, all legal aggregates are enumerated in the spec itself.

However, as with custom functions, there are many use cases for creating and using custom aggregate functions. Stardog provides a mechanism for creating and using custom aggregates **without** requiring custom SPARQL syntax.

Implementing Custom Aggregates

To implement a custom aggregate, you should extend AbstractAggregate (java/snarl/com/complexible/stardog/plan/aggregates/AbstractAggregate.html).

The rules regarding constructor, "copy constructor" and the `copy` method for `Function` apply to `Aggregate` as well.

Two methods must be implemented for custom aggregates, `Value _getValue() throws ExpressionEvaluationException` and `void aggregate(final Value theValue, final long theMultiplicity) throws ExpressionEvaluationException`. `_getValue` returns the computed aggregate value while `aggregate` adds a Value to the current running aggregation. In terms of the `COUNT` aggregate, `aggregate` would increment the counter and `_getValue` would return the final count.

The multiplicity argument to `aggregate` corresponds to the fact that intermediate solution sets have a multiplicity associated with them. It's most often 1, but joins and choice of the indexes used for the scans internally can affect this. Rather than repeating the solution N times, we associate a multiplicity of N with the solution. Again, in terms of `COUNT`, this would mean that rather than incrementing the count by `1`, it would be incremented by the multiplicity.

Registering Custom Aggregates

Aggregates such as `COUNT` or `SAMPLE` are implementations of `Function` in the same way `sameTerm` or `str` are and are registered with Stardog in the exact same manner.

Using Custom Aggregates

You can use your custom aggregates just like any other aggregate function. Assuming we have a custom aggregate `gmean` defined in the `tag:stardog:api:` namespace, we can refer to it within a query as such:

```
PREFIX : <http://www.example.org>
PREFIX stardog: <tag:stardog:api:>

SELECT (stardog:gmean(?O) AS ?C)
WHERE { ?S ?P ?O }
```

Database Archetypes

The Stardog database archetypes (http://docs.stardog.com/#_database_archetypes) provide a simple way to associate one or more ontologies and optionally a set of constraints with a database. Stardog provides two built-in database archetypes out-of-the-box: PROV and SKOS.

Running FOAF Example

This example shows a user-define archetype for FOAF (http://xmlns.com/foaf/spec/).

First build the jar file for this example using gradle:

```
$ ./gradlew jar
```

Copy the jar file to your Stardog installation directory and (re)start the server:

```
$ cp examples/foaf/build/libs/foaf-*.jar $STARDOG/server/dbms/
$ $STARDOG/bin/stardog-admin server start
```

Create a new database using the FOAF archetype:

```
$ $STARDOG/bin/stardog-admin db create -o database.archetypes="foaf" -n foafDB
```

That's it. Even though you created a database without any data you will see that there is a default namespace, ontology and constraints associated with this database:

```
$ bin/stardog namespace list foafDB
+---------+---------------------------------------------+
| Prefix  |                  Namespace                  |
+---------+---------------------------------------------+
| foaf    | http://xmlns.com/foaf/0.1/                  |
| owl     | http://www.w3.org/2002/07/owl#              |
| rdf     | http://www.w3.org/1999/02/22-rdf-syntax-ns# |
| rdfs    | http://www.w3.org/2000/01/rdf-schema#       |
| stardog | tag:stardog:api:                            |
| xsd     | http://www.w3.org/2001/XMLSchema#           |
+---------+---------------------------------------------+
$ bin/stardog reasoning schema foafDB
foaf:publications a owl:ObjectProperty
foaf:jabberID a owl:InverseFunctionalProperty
foaf:jabberID a owl:DatatypeProperty
foaf:interest rdfs:domain foaf:Agent
foaf:workInfoHomepage a owl:ObjectProperty
foaf:schoolHomepage rdfs:range foaf:Document
foaf:status a owl:DatatypeProperty
foaf:currentProject rdfs:domain foaf:Person
...
$ bin/stardog icv export foafDB
AxiomConstraint{foaf:isPrimaryTopicOf a owl:InverseFunctionalProperty}
```

Registering Archetypes

User-defined archetypes are loaded to Stardog through JDK `ServiceLoader` framework. Create a file called `com.complexible.stardog.db.DatabaseArchetype` in the `META-INF/services` directory. The contents of this file should be all of the **fully-qualified** class names for your custom archetypes.

Search Analyzers

By default, the full-text index in Stardog uses Lucene's StandardAnalyzer (https://lucene.apache.org/core/4_7_2/analyzers-common/org/apache/lucene/analysis/standard/StandardAnalyzer.html).

However, any class implementing `org.apache.lucene.analysis.Analyzer` can be used in place of the default analyzer. To specify a different `Analyzer` a service named `com.complexible.stardog.search.AnalyzerFactory` should be registered. `AnalyzerFactory` returns the desired `Analyzer` implementation to be used when creating the Lucene index from the RDF contained in the database.

This is an example of an AnalyzerFactory which uses the built-in Lucene analyzer for the French language:

```
public final class FrenchAnalyzerFactory implements AnalyzerFactory {

        /**
         * {@inheritDoc}
         */
        @Override
        public Analyzer get() {
                return new FrenchAnalyzer(Version.LUCENE_47);
        }
}
```

Any of the common Lucene analyzers (https://lucene.apache.org/core/4_7_2/analyzers-common/index.html) can be used as well as any custom implementation of `Analyzer`. In the latter case, be sure your implementation is in Stardog's class path.

Create a file called `com.complexible.stardog.search.AnalyzerFactory` in the `META-INF/services` directory. The contents of this file should be the **fully-qualified** class name of your `AnalyzerFactory`.

Note, as of Stardog 3.0, only **one** `AnalyzerFactory` can be registered at a time, attempts to register more than one will yield errors on startup.

JAVASCRIPT PROGRAMMING

Source code and documentation for stardog.js are available available on Github (https://github.com/stardog-union/stardog.js) and npm (https://npmjs.org/package/stardog).

stardog.js

This framework wraps all the functionality of a client for the Stardog DBMS and provides access to a full set of functions such as executing SPARQL Queries, administration tasks on Stardog, and the use of the Reasoning API.

The implementation uses the HTTP protocol, since most of Stardog functionality is available using this protocol. For more information, see Network Programming (#_network_programming).

The framework is currently supported for node.js and the browser, including test cases for both environments. You'll need npm to run the test cases and install the dependencies.

CLOJURE PROGRAMMING

The stardog-clj source code (http://github.com/complexible/stardog-clj) is available as Apache 2.0 licensed code.

Installation

Stardog-clj is available from Clojars. To use, just include the following dependency:

```
[stardog-clj "2.2.2"]
```

Starting with Stardog 2.2.2, the stardog-clj version always matches the latest release of Stardog.

Overview

Stardog-clj provides a set of functions as API wrappers to the native SNARL API. These functions provide the basis for working with Stardog, starting with connection management, connection pooling, and the core parts of the API, such as executing a SPARQL query or adding and removing RDF from the Stardog database. Over time, other parts of the Stardog API will be appropriately wrapped with Clojure functions and idiomatic Clojure data structures.

Stardog-clj provides the following features:

1. Specification based descriptions for connections, and corresponding "connection" and "with-connection-pool" functions and macros
2. Functions for query, ask, graph, and update to execute `SELECT`, `ASK`, `CONSTRUCT`, and SPARQL Update queries respectively
3. Functions for insert and remove, for orchestrating the Adder and Remover APIs in SNARL
4. Macros for resource handling, including with-connection-tx, with-connection-pool, and with-transaction
5. Support for programming Stardog applications with either the connection pool or direct handling of the connection
6. Idiomatic clojure handling of data structures, with converters that can be passed to query functions

The API with source docs can be found in the `stardog.core` and `stardog.values` namespaces.

API Overview

The API provides a natural progression of functions for interacting with Stardog

```
(create-db-spec "testdb" "snarl://localhost:5820/" "admin" "admin" "none")
```

This creates a connection space for use in `connect` or `make-datasource` with the potential parameters:

```
{:url "snarl://localhost:5820/" :db "testdb" :pass "admin" :user "admin" :max-idle 100 :max-pool 200 :min-
pool 10 :reasoning false}
```

Create a single Connection using the database spec. Can be used with `with-open`, `with-transaction`, and `with-connection-tx` macros.

```
(connect db-spec)
```

Creates a data source, i.e. `ConnectionPool`, using the database spec. Best used within the `with-connection-pool` macro.

```
(make-datasource db-spec)
```

Executes the body with a transaction on each of the connections. Or establishes a connection and a transaction to execute the body within.

```
(with-transaction [connection...] body)
(with-connection-tx binding-forms body)
```

Evaluates body in the context of an active connection obtained from the connection pool.

```
(with-connection-pool [con pool] .. con, body ..)
```

Examples

Here are some examples of using stardog-clj

Create a connection and run a query

```
Unresolved directive in clojure.ad -
include::https://gist.githubusercontent.com/AlBaker/10039066/raw/Stardog.clj[]
```

Insert data

```
Unresolved directive in clojure.ad -
include::https://gist.githubusercontent.com/AlBaker/10039743/raw/StardogInsert.clj[]
```

Run a query with a connection pool

```
Unresolved directive in clojure.ad -
include::https://gist.githubusercontent.com/AlBaker/10041003/raw/StardogPool.clj[]
```

SPARQL Update

```
Unresolved directive in clojure.ad -
include::https://gist.githubusercontent.com/AlBaker/11130645/raw/StardogUpdate.clj[]
```

Graph function for Construct queries

```
Unresolved directive in clojure.ad -
include::https://gist.githubusercontent.com/AlBaker/11130603/raw/StardogGraph.clj[]
```

Ask function for ASK queries

```
Unresolved directive in clojure.ad -
include::https://gist.githubusercontent.com/AlBaker/11130618/raw/StardogAsk.clj[]
```

.NET PROGRAMMING

In the Network Programming (#_network_programming) section, we looked at how to interact with Stardog over a network via HTTP and SNARL protocols. In this chapter we describe how to program Stardog from .Net using http://www.dotnetrdf.org (http://www.dotnetrdf.org).

NOTE .dotNetRDF is an open source library developed and supported by third parties; questions or issues with the .Net API should be directed to http://www.dotnetrdf.org (http://www.dotnetrdf.org).

You should also be aware that dotNetRDF uses the HTTP API for all communication with Stardog so you **must** enable the HTTP server to use Stardog from .Net. It's enabled by default so most users should not need to do anything to fulfill this requirement.

dotNetRDF Documentation

See the documentation (https://bitbucket.org/dotnetrdf/dotnetrdf/wiki/Home) for using dotNetRDF with Stardog.

SPRING PROGRAMMING

The Spring for Stardog source code (http://github.com/complexible/stardog-spring) is available on Github. Binary releases are available on the Github release page (https://github.com/complexible/stardog-spring/releases).

As of 2.1.3, Stardog-Spring and Stardog-Spring-Batch can both be retrieved from Maven central:

- `com.complexible.stardog:stardog-spring:2.1.3`
- `com.complexible.stardog:stardog-spring-batch:2.1.3`

The corresponding Stardog Spring version will match the Stardog release, e.g. stardog-spring-2.2.2 for Stardog 2.2.2.

Overview

Spring for Stardog makes it possible to rapidly build Stardog-backed applications with the Spring Framework. As with many other parts of Spring, Stardog's Spring integration uses the template design pattern for abstracting standard boilerplate away from application developers.

Stardog Spring can be included via Maven with `com.complexible.stardog:stardog-spring:version` and `com.complexible.stardog:stardog-spring-batch` for Spring Batch support. Both of these dependencies require the public Stardog repository to be included in your build script, and the Stardog Spring packages installed in Maven. Embedded server is still supported, but via providing an implementatino of the Provider interface. This enables users of the embedded server to have full control over how to use the embedded server.

At the lowest level, Spring for Stardog includes

1. `DataSouce` and `DataSourceFactoryBean` for managing Stardog connections
2. `SnarlTemplate` for transaction- and connection-pool safe Stardog programming
3. `DataImporter` for easy bootstrapping of input data into Stardog

In addition to the core capabilities, Spring for Stardog also integrates with the Spring Batch framework. Spring Batch enables complex batch processing jobs to be created to accomplish tasks such as ETL or legacy data migration. The standard `ItemReader` and `ItemWriter` interfaces are implemented with a separate callback writing records using the SNARL Adder API.

Basic Spring

There are three Beans to add to a Spring application context:

- `DataSourceFactoryBean`: `com.complexible.stardog.ext.spring.DataSourceFactoryBean`
- `SnarlTemplate`: `com.complexible.stardog.ext.spring.SnarlTemplate`
- `DataImporter`: `com.complexible.stardog.ext.spring.DataImporter`

`DataSourceFactoryBean` is a Spring `FactoryBean` that configures and produces a `DataSource`. All of the Stardog `ConnectionConfiguration` and `ConnectionPoolConfig` methods are also property names of the `DataSourceFactoryBean`--for example, "to", "url", "createIfNotPresent". If you are interested in running an embedded server, use the `Provider` interface and inject it into the `DataSourceFactoryBean`. Note: all of the server jars must be added to your classpath for using the embedded server.

`javax.sql.DataSource`, that can be used to retrieve a `Connection` from the `ConnectionPool`. This additional abstraction serves as place to add Spring-specific capabilities (e.g. `spring-tx` support in the future) without directly requiring Spring in Stardog.

`SnarlTemplate` provides a template abstraction over much of Stardog's native API, SNARL (#_java_programming), and follows the same approach of other Spring template, i.e., `JdbcTemplate`, `JmsTemplate`, and so on.

Spring for Stardog also comes with convenience mappers, for automatically mapping result set bindings into common data types. The `SimpleRowMapper` projects the `BindingSet` as a `List>` and a `SingleMapper` that accepts a constructor parameter for binding a single parameter for a single result set.

The key methods on `SnarlTemplate` include the following:

```
query(String sparqlQuery, Map args, RowMapper)
```

`query()` executes the SELECT query with provided argument list, and invokes the mapper for result rows.

```
doWithAdder(AdderCallback)
```

`doWithAdder()` is a transaction- and connection-pool safe adder call.

```
doWithGetter(String subject, String predicate, GetterCallback)
```

`doWithGetter()` is the connection pool boilerplate method for the `Getter` interface, including the programmatic filters.

```
doWithRemover(RemoverCallback)
```

`doWithRemover()` As above, the remover method that is transaction and pool safe.

```
execute(ConnectionCallback)
```

`execute()` lets you work with a connection directly; again, transaction and pool safe.

```
construct(String constructSparql, Map args, GraphMapper)
```

`construct()` executes a SPARQL CONSTRUCT query with provided argument list, and invokes the `GraphMapper` for the result set.

`DataImporter` is a new class that automates the loading of RDF files into Stardog at initialization time.

It uses the Spring Resource API, so files can be loaded anywhere that is resolvable by the Resource API: classpath, file, url, etc. It has a single load method for further run-time loading and can load a list of files at initialization time. The list assumes a uniform set of file formats, so if there are many different types of files to load with different RDF formats, there would be different `DataImporter` beans configured in Spring.

Spring Batch

In addition to the base `DataSource` and `SnarlTemplate`, Spring Batch support adds the following:

- `SnarlItemReader`: `com.complexible.stardog.ext.spring.batch.SnarlItemReader`
- `SnarlItemWriter`: `com.complexible.stardog.ext.spring.batch.SnarlItemWriter`
- `BatchAdderCallback`: `com.complexible.stardog.ext.spring.batch.BatchAdderCallback`

GROOVY PROGRAMMING

Groovy (http://http://groovy.codehaus.org/) is an agile and dynamic programming language for the JVM, making popular programming features such as closures available to Java developers. Stardog's Groovy support makes life easier for developers who need to work with RDF, SPARQL, and OWL by way of Stardog.

The Groovy for Stardog source code (http://github.com/complexible/stardog-groovy) is available on Github.

Binary releases are available on the Github release page (https://github.com/complexible/stardog-groovy/releases) and via Maven central as of version 2.1.3 and beyond using the following dependency declaration (Gradle style) `com.complexible.stardog:stardog-groovy:2.1.3`.

As of version 2.1.3, Stardog-Groovy can be included via "com.complexible.stardog:stardog-groovy:2.1.3" from Maven central.

NOTE You must include our public repository in your build script to get the Stardog client dependencies into your local repository.

Using the embedded server with Stardog Groovy is not supported in 2.1.2, due to conflicts of the asm library for various third party dependencies. If you wish to use the embedded server with similar convenience APIs, please try Stardog with Spring (#_spring_programming). Also 2.1.3 and beyond of Stardog-Groovy no longer requires the use of the Spring framework.

The Stardog-Groovy version always matches the Stardog release, e.g. for Stardog 2.2.2 use stardog-groovy-2.2.2.

Overview

Groovy for Stardog provides a set of Groovy API wrappers for developers to build applications with Stardog and take advantage of native Groovy features. For example, you can create a Stardog connection pool in a single line, much like Groovy SQL support. In Groovy for Stardog, queries can be iterated over using closures and transaction safe closures can be executed over a connection.

For the first release, Groovy for Stardog includes `com.complexible.stardog.ext.groovy.Stardog` with the following methods:

1. `Stardog(map)` constructor for managing Stardog connection pools
2. `each(String, Closure)` for executing a closure over a query's results, including projecting SPARQL result variables into the closure.
3. `query(String, Closure)` for executing a closure over a query's results, passing the BindingSet to the closure
4. `insert(List)` for inserting a list of vars as a triple, or a list of list of triples for insertion
5. `remove(List)` for removing a triple from the database
6. `withConnection` for executing a closure with a transaction safe instance of `Connection` (/docs/5.3.6/java/snarl/com/complexible/stardog/api/connection)

Examples

Here are some examples of the more interesting parts of Stardog Groovy.

Create a Connection

```
Unresolved directive in groovy.ad -
include::https://gist.githubusercontent.com/AlBaker/4652565/raw/StardogConnection.groovy[]
```

SPARQL Vars Projected into Groovy Closures

```
Unresolved directive in groovy.ad -
include::https://gist.githubusercontent.com/AlBaker/4652590/raw/Projection.groovy[]
```

Add & Remove Triples

```
Unresolved directive in groovy.ad -
include::https://gist.githubusercontent.com/AlBaker/4652608/raw/StardogAddRemove.groovy[]
```

withConnection Closure

```
Unresolved directive in groovy.ad -
include::https://gist.githubusercontent.com/AlBaker/4652621/raw/StardogWithConnection.groovy[]
```

SPARQL Update Support

```
Unresolved directive in groovy.ad -
include::https://gist.githubusercontent.com/AlBaker/7862684/raw/StardogGroovyUpdate.groovy[]
```

MIGRATION GUIDE

Given the impact of the new storage engine in Stardog 6, we are departing from our ordinary practice and running a closed alpha release program for customers and then an open beta release program for anyone. The following migration guidance is intended to smooth the migration path, both in alpha and beta release programs.

WARNING You should not migrate mission-critical data or any data that you cannot afford to lose. This is presently an **alpha release** so plan accordingly.

Migrating to Stardog 6

Stardog 6 introduces a new storage engine and snapshot isolation (https://en.wikipedia.org/wiki/Snapshot_isolation) for concurrent transactions. This section provides an overview of those changes and how they affect users and programs written against previous versions.

Stardog 6 introduces a completely new disk index format and databases created by previous versions of Stardog must be migrated in order to work with Stardog 6. There is a dedicated CLI command for migrating the contents of an existing Stardog home directory (i.e., all of the databases in a multi-tenant system).

Functionality Unavailable During Alpha

The following functionality of Stardog 5 is unavailable during the Stardog 6 alpha release and will be available in Stardog 6 beta releases.

- Versioning (#_versioning)
- Machine Learning (#_machine_learning)

NOTE The following instructions are for migrating all the databases in an existing STARDOG_HOME directory. Instead of migrating all the databases you can start with a new empty home directory and restore select databases using backups created by Stardog versions 4 or 5. If you use the following instructions with very large databases then you should increase the memory settings by setting the environment variable STARDOG_JAVA_ARGS. The memory settings you use for Stardog 5 would work well with the migrate command.

Migrating Stardog

The steps for a single server migration:

- Stop the existing Stardog server; do **not** start Stardog 6 or have either server running
- Create a new empty Stardog home folder (we'll call it `NEW_HOME`)
- Copy your license file to `NEW_HOME`
- Unzip the Stardog 6 distribution
- In Stardog 6 distribution, run the following command:

```
$ stardog-admin server migrate OLD_HOME NEW_HOME
```

The command will migrate the contents of the each database along with the system database that contains users, roles, permissions, and other metadata. Progress for the migration will be printed to `STDOUT` and can take a significant amount of time if you have large databases. The `stardog.properties` (if it exists) file will **not** be copied automatically. See the next section for changes to the configuration options.

Migrating Docker-hosted Stardog

The migration process for Stardog running in Docker is effectively the same with a couple of Docker-specific differences.

- Stop your Docker container.
- Create a new directory on the Docker host machine (we'll call it `NEW_HOME`).
- Copy your license file to `NEW_HOME`
- Run the Stardog 6 Docker container in the following way, which will bring you to a command prompt within the container:

```
$ docker run -v <path to NEW_HOME>:/var/opt/stardog -v <path to OLD_HOME>:/old_stardog \
--entrypoint /bin/bash -it complexible-eps-docker.jfrog.io/stardog:6.0.0-alpha
```

- Run the Stardog 6 migration tool in the following way:

```
$ /opt/stardog/bin/stardog-admin server migrate /old_stardog /var/opt/stardog
```

Migrating Stardog Cluster

The migration steps for the cluster:

- Stop all of the cluster nodes, but not the ZK cluster
- Follow the above steps for single server migration on any **one** cluster node
- Run the command `stardog-admin zk clear`
- Start the node where migration completed with Stardog 6
- On the other cluster nodes, create empty home folders
- Start another node, wait for the node to join the cluster, and then repeat for each cluster node

Disk Usage and Layout

The layout of data in Stardog 6 home directory is different than in all previous versions. Previously the data stored in a database was stored under a directory with the name of the database. In Stardog 6 the data for all databases is stored in a directory named `data` in the home directory. The database directories still exist but they contain only index metadata along with search and spatial index if those features are enabled.

The disk usage requirements for Stardog 6 are higher than Stardog 5. The actual difference will depend on the characteristics of your data, but you should expect to see 20% to 30% increase in disk usage. Similar to Stardog 5, the disk usage of bulk loaded databases, e.g. when data is loaded by the `stardog-admin db create` command, will be lower than the disk usage when the same data is added incrementally, that is, in smaller transactions over time.

Memory Databases

Stardog 6 no longer supports in-memory databases. If keeping all data in memory is desired, we recommend placing the home directory on a RAM disk and create databases in the usual way.

Database Optimization & Compaction

Similar to Stardog 5, Stardog 6 performance degrades over time as the database is updated with transactions. The disk usage will continue to increase and data deleted by transactions will not be removed from disk. The existing `db optimize` command can be used to perform index compaction on disk to improve the performance of reads and writes.

Database Configuration

Most server and database options and their semantics are unchanged in Stardog 6, with the following exceptions:

- Options starting with `index.differential.`. Stardog 6 has a new mechanism which replaces the previous implementation of differential index (#_differential_indexes).
- `transaction.isolation` needs to be set to `SERIALIZABLE` for ICV guard mode (#ICV Guard Mode) in order to ensure data integrity w.r.t. the constraints.

Snapshot Isolation

Stardog 6 uses a multi-versioned concurrency control (MVCC) model providing lock-free transactions with snapshot isolation guarantees. Stardog 5 provided a weaker snapshot isolation mechanism that required writers to acquire locks that sometimes blocked other transactions for a very long time, which is no longer the case. As a result, the performance of concurrent updates is greatly improved in Stardog 6, especially in the cluster setting.

An implication of the new MVCC approach is that if two concurrent transactions try to add or remove the same triple, one of the transactions will be aborted with a transaction conflict. In Stardog 5, such transactions were allowed. This caused anomalies when transactions updated the same triples value, e.g. updating a counter. In Stardog 6, the client should decide if conflicted transactions should be retried or aborted.

Migrating to Stardog 5

Stardog 5 introduces significant changes; this section provides an overview of those changes and how they affect users.

Disk Indexes

Stardog 5 does not change the format of disk indexes but uses new algorithms and data structures for computing an storing statistics for improved query planning. Migration of statistics is performed automatically the first time Stardog is started even if `server start` is executed without the `--upgrade` option. This migration might take a while based on your databases size and a progress of the update will be printed on the console.

The new statistics is not backward compatible and old versions of Stardog cannot be started with the same home directory after statistics has been migrated. If you want to revert back to an older version of Stardog you should manually delete all the `statistics.*` directories and Stardog 4 will recompute the statistics on start up. Again, this might take considerable time for large databases. If you want to switch between Stardog 4 and Stardog 5 quickly you should have two copies of your home directory.

Network Protocol

SNARL protocol was deprecated in Stardog 4 and is completely removed in Stardog 5. Note that, this does not affect the SNARL Java API which continues to be the preferred API to work with Stardog 5. If you have been using SNARL protocol, i.e. connection strings that begin with `snarl://` (or `snarls://`), then you should change your connection strings to begin with `http://` (or `https://`). If you are using the SNARL Java API you might need to update your library dependencies to use the HTTP client dependency. See the Using Maven (#_using_maven) section for details.

Embedded Mode

Stardog 5 no longer requires a running server for use of Stardog in an embedded manner. To use an embedded version of Stardog, you simply start Stardog:

```
Stardog aStardog = Stardog.builder().create();
try {
    // use stardog
}
finally {
    aStardog.shutdown();
}
```

Then use the existing SNARL API methods for connecting to an embedded server. See our examples (https://github.com/stardog-union/stardog-examples/blob/develop/examples/api/main/src/com/complexible/stardog/examples/api/ConnectionAPIExample.java) for a complete demonstration of using Stardog.

UNDERSTANDING STARDOG

Background information on performance, testing, terminology, known issues, compatibility policies, etc.

FAQ

Some frequently asked questions for which we have answers.

How do I report a bug? What information should I include?

Question

Something isn't working and I don't know what to do…

Answer

A bug report seems prudent in this case; customers should use their dedicated support channel. Others may use the support forum (https://community.stardog.com). You should include, at a minimum:

1. which release and version of Stardog you are using 4.x? 5.x? Community? Enterprise?
2. which JVM you are using
3. anything from `stardog.log` (in `STARDOG_HOME`) that seems relevant

Why can't I load Dbpedia (or other RDF) data?

Question

I get a parsing error when loading Dbpedia or some other RDF. What can I do?

Answer

First, it's not a bad thing to expect data providers to publish valid data. Second, it is, apparently, a very naive thing to expect data providers to publish valid data…

Stardog supports a loose parsing mode which will ignore certain kinds of data invalidity and may allow you to load invalid data. See `strict.parsing` in Configuration Options (#_configuration_options).

Why doesn't search work?

Question

I created a database but search doesn’t work.

Answer

Search is disabled by default; you can enable it at database creation time, or at any subsequent time, using the Web Console or by using `metadata set` (/man/metadata-set.html) CLI. It can be enabled using `db create` (/man/db-create.html) too.

Why don’t my queries work?!

Question

I’ve got some named graphs and blah blah my queries don’t work blah blah.

Answer

Queries with FROM NAMED with a named graph that is not **in** Stardog will not cause Stardog to download the data from an arbitrary HTTP URL and include it in the query. *Stardog will only evaluate queries over data that has been loaded into it*.

SPARQL queries without a context or named graph are executed against **the default, unnamed graph**. In Stardog, **the default graph is not the union of all the named graphs and the default graph**. This behavior is configurable via the `query.all.graphs` configuration parameter.

Why is Stardog Cluster acting weird or running slowly?

Question

Should I put Stardog HA and Zookeeper on the same hard drives?

Answer

Never do this! Zookeeper is disk-intensive and displays bad I/O contention with Stardog query evaluation. Running both Zk and Stardog on the same disks will result in bad performance and, in some cases, intermittent failures.

SPARQL 1.1

Question

Does Stardog support SPARQL 1.1?

Answer

Yes.

Deadlocks and Slowdowns

Question

Stardog slows down or deadlocks?! I don’t understand why, I’m just trying to send some queries and do something with the results…in a tight inner loop of doom!

Answer

Make sure you are closing result sets (`TupleQueryResult` and `GraphQueryResult`; or the Jena equivalents) when you are done with them. These hold open resources both on the client and on the server and failing to close them when you are done will cause files, streams, lions, tigers, and bears to be held open. If you do that enough, then you’ll eventually exhaust all of the resources in their respective pools, which can cause slowness or, in some cases, deadlocks waiting for resources to be returned.

Similarly close your connections when you are done with them. Failing to close `Connections`, `Iterations`, `QueryResults`, and other **closeable** objects will lead to undesirable behavior.

Update Performance

Question

I’m adding one triple at a time, in a tight loop, to Stardog; is this the ideal strategy with respect to performance?

Answer

The answer is "not really"…Update performance is best if there are fewer transactions that modify larger number of triples. If you are using the Stardog Java API, the client will buffer changes in large transactions and flush the buffer periodically so you don't need to worry about memory problems. If you need transactions with small number of triples then you may need to experiment to find the sweet spot with respect to your data, database size, the size of the differential index, and update frequency.

Public Endpoint

Question

I want to use Stardog to serve a public SPARQL endpoint; is there some way I can do this without publishing user account information?

Answer

We don't necessarily recommend this, but it's possible. Simply pass `--disable-security` to `stardog-admin` when you start the Stardog Server. This **completely** disables security in Stardog which will let users access the SPARQL endpoint, and all other functionality, without needing authorization.

Remote Bulk Loading

Question

I'm trying to create a database and bulk load files from my machine to the server and it's not working, the files don't seem to load, what gives?

Answer

Stardog does not transfer files during database creation to the server, sending big files over a network kind of defeats the purpose of blazing fast bulk loading. If you want to bulk load files from your machine to a remote server, copy them to the server and bulk load them.

Canonicalized Literals

Question

Why doesn't my literal look the same as I when I added it to Stardog?

Answer

Stardog performs literal canonicalization (/docs/5.3.6/java/snarl/com/complexible/stardog/index/indexoptions#CANONICAL_LITERALS) by default. This can be turned off by setting `index.literals.canonical` to `false`. See Configuration Options (#_configuration_options) for the details.

Cluster Isn't Working

Question

I've setup Stardog Cluster, but it isn't working and I have `NoRouteToHostException` exceptions all over my Zookeeper log.

Answer

Typically—but especially on Red Hat Linux and its variants—this means that `iptables` is blocking one, some, or all of the ports that the Cluster is trying to use. You can disable `iptables` or, better yet, configure it to unblock the ports Cluster is using.

Client Connection Isn't Working

Question

I'm getting a `ServiceConfigurationError` saying that `SNARLDriver` could not be instantiated.

Answer

Make sure that your classpath includes all Stardog JARs and that the user executing your code has access to them.

Logging

Question

Why doesn't Stardog implement our (byzantine and proprietary!) corporate logging scheme?

Answer

Stardog 4 will log to `$STARDOG_HOME/stardog.log` by default (#_logging), but you can use a log4j 2 config file in `$STARDOG_HOME` so that Stardog will log wherever & however you want.

Loading Compressed Data

Question

How can I load data from a compressed format that Stardog doesn't support without decompressing the file?

Answer

Stardog supports several compression formats by default (zip, gzip, bzip2) so files compressed with those formats can be passed as input directly without decompression. Files compressed with other formats can also be loaded to Stardog by decompressing them on-the-fly using named pipes (http://en.wikipedia.org/wiki/Named_pipe) in Unix-like systems. The following example shows using a named pipe where the decompressed data is sent directly to Stardog without being writing to disk.

```
$ mkfifo some-data.rdf
$ xz -dc some-data.rdf.xz > some-data.rdf &
$ stardog-admin db create -n test some-data.rdf
```

SNARL Protocol

Question

I'm using Stardog and I'm seeing messages about SNARL deprecation, what's that about?

Answer

As of Stardog 4.2, the SNARL protocol is no longer the default protocol for Stardog and is deprecated. Support for the SNARL protocol will be removed in the Stardog 5 release. The HTTP protocol is now the default protocol and is recommended for all users. If you're seeing the deprecation warnings it's because you're explicitly using the SNARL protocol somewhere in your application.

All you have to do to migrate these usages is change `snarl`/`snarls` to `http`/`https` and you're done. You can no longer disable support for HTTP when starting the server; the `--no-http` option does noting, and the SNARL protocol is not enabled by default anymore and has to be explicitly enabled by using `--snarl`.

Please note, the SNARL *protocol* has been deprecated. The SNARL API is still supported and the recommended Java API for Stardog.

Working with RDF Files

Question

I have some RDF files that I need to process without loading into Stardog. What can I do?

Answer

As of Stardog 5.0, Stardog provides some CLI commands that work directly over files. These commands exist under the `stardog file` command. For example, you can use the `file cat` (https://stardog.com/docs/man/file-cat) command to concatenate multiple RDF files into a single file and `file split` (https://stardog.com/docs/man/file-split) command to split a single RDF file into multiple RDF files. These commands are similar to their *nix counterparts but can handle RDF formats and perform compression/decompression on-the-fly. There is also the `file obfuscate` (https://stardog.com/docs/man/file-obfuscate) command that can create an obfuscated version of the input RDF files similar to `data obfuscate` (https://stardog.com/docs/man/data-obfuscate) command.

Virtual Graph JDBC Driver Requirements

Question

What JDBC driver do I need for a virtual graph connection?

Answer

Virtual graph connections require a JDBC driver supporting JDBC 4.1 / Java 7. For Oracle, this means `ojdbc7.jar` or later.

Virtual Graph Identifier Quoting

Question

How do I quote field and table names in mappings and when should I do it?

Answer

Interpretation of identifiers follows that of the database system backing the virtual graph. For example, Oracle, interprets nonquoted identifiers as uppercase. PostgreSQL interprets unquoted identifiers as lowercase. In general, if you need to quote the identifier in a query, then you should quote it in a mapping.

Quoting is done using the native quoting character of the database. This means double quote for Oracle, PostgreSQL and other SQL standard-compatible systems. MySQL uses a backquote and SQL Server uses square brackets. This setting can be overridden by adding `parser.sql.quoting=ANSI` to your virtual graph properties file. This will allow the use of double quotes to quote identifiers. This is commonly done to write mappings using the R2RML convention of using double quotes and supporting mappings generated by other systems.

Virtual Graph Table not Found

Question

Why am I getting an error when I try to create a virtual graph? `Unable to parse logical table : From line 1, column 15 to line 1, column 18: Object 'SOME_TABLE' not found`

Answer

The virtual graph subsystem maintains a set of metadata including a list of tables and the types of their fields. If a table is not found, it's likely that it either needs to be quoted or the schema needs to be added to the search path by adding `sql.schemas` to your virtual graph properties file. This setting enables Stardog to see the table metadata in the named schemas. The table/query still needs to be qualified with the schema name when referring to it.

Benchmark Results

Live, dynamically updated performance data (https://docs.google.com/spreadsheets/d/1oHSWX_0ChZ61ofipZ1CMsW7OhyujioR28AfHzU9d56k/pubhtml) from BSBM, SP2B, LUBM benchmarks against the latest Stardog release.

Compatibility Policies

The Stardog 5.x release ("Stardog" for short) is a major milestone in the development of the system. Stardog is a stable platform for the growth of projects and programs written for Stardog.

Stardog provides (and defines) several user-visible things:

1. SNARL API
2. Stardog HTTP Protocol
3. a command-line interface

It is intended that programs—as well as SPARQL queries—written to Stardog APIs, protocols, and interfaces will continue to run correctly, unchanged, over the lifetime of Stardog. That is, over all releases identified by version `5.x.y`. At some indefinite point, Stardog 6.x will be released; but, until that time, and likely even after it, Stardog programs that work today should continue to work even as future releases of Stardog occur. APIs, protocols, and interfaces may grow, acquiring new parts and features, but not in a way that breaks existing Stardog programs.

Expectations

Although we expect that nearly all Stardog programs will maintain this compatibility over time, it is impossible to guarantee that no future change will break any program. This document sets expectations for the compatibility of Stardog programs in the future. The main, foreseeable reasons for which this compatibility may be broken in the future include:

1. **Security**: We reserve the right to break compatibility if doing so is required to address a security problem in Stardog.
2. **Unspecified behavior**: Programs that depend on unspecified[40 (#_footnote_40)] behaviors may not work in the future if those behaviors are modified.
3. **3rd Party Specification Errors**: It may become necessary to break compatibility of Stardog programs in order to address problems in some 3rd party specification.
4. **Bugs**: It will not always be possible to fix bugs found in Stardog—or in its 3rd party dependencies—while also preserving compatibility. With that proviso, we will endeavor to only break compatibility when repairing critical bugs.

It is always possible that the performance of a Stardog program may be (adversely) affected by changes in the implementation of Stardog. No guarantee can be made about the performance of a given program between releases, except to say that our expectation is that performance will generally trend in the appropriate direction.

Data Migration & Safety

We expect that data safety will always be given greater weight than any other consideration. But since Stardog stores a user's data differently from the form in which data is input to Stardog, we may from time to time change the way it is stored such that explicit data migration will be necessary.

Stardog provides for two data migration strategies:

1. Command-line migration tool(s)
2. Dump and reload

We expect that explicit migrations may be required from time to time between different releases of Stardog. We will endeavor to minimize the need for such migrations. We will only require the "dump and reload" strategy between **major** releases of Stardog (that is, from 1.x to 2.x, etc.), unless that strategy of migration is required to repair a security or other data safety bug.

Known Issues

The known issues in Stardog 5.3.6:

1. Our implementation of `CONSTRUCT` slightly deviates from the SPARQL 1.1 specification (http://www.w3.org/TR/sparql11-query/#construct): it does not implicitly `DISTINCT` query results; rather, it implicitly applies `REDUCED` semantics to `CONSTRUCT` query results.[41 (#_footnote_41)]
2. Asking for all individuals with reasoning via the query `{?s a owl:Thing}` might also retrieve some classes and properties. **WILLFIX**
3. Schema queries do not bind graph variables.
4. Dropping a database deletes all of the data files in Stardog Home associated with that database. If you want to keep the data files and remove the database from the system catalog, then you need to manually copy these files to another location before dropping the database.
5. If relative URIs exist in the data files passed to create, add, or remove commands, then they will be resolved using the constant base URI `http://api.stardog.com/ (http://api.stardog.com/)` if, but only if, the format of the file allows base URIs. Turtle and RDF/XML formats allows base URIs but N-Triples format doesn't allow base URIs and relative URIs in N-Triples data will cause errors.
6. Queries with `FROM NAMED` with a named graph that is **not** in Stardog will **not** cause Stardog to download the data from an arbitrary HTTP URL and include it in the query.
7. SPARQL queries without a context or named graph are executed against the default, unnamed graph. In Stardog, the default graph is **not** the union of all the named graphs and the default graph. Note: this behavior is configurable via the `query.all.graphs` configuration parameter.
8. RDF literals are limited to 8MB (after compression) in Stardog. Input data with literals larger than 8MB (after compression) will raise an exception.

Glossary

In the Stardog documentation, the following terms have a specific technical meaning.

Stardog Database Management System, aka Stardog Server	An instance of Stardog; only one Stardog Server may run per JVM. A computer may run multiple Stardog Servers by running one per multiple JVMs.
Stardog Home, aka `STARDOG_HOME`	A directory in a filesystem in which Stardog stores files and other information; established either in a Stardog configuration file or by environment variable. Only one Stardog Server may run simultaneously from a `STARDOG_HOME`.
Stardog Network Home	A URL (HTTP or SNARL) which identifies a Stardog Server running on the network.
Database	A Stardog database is a graph of RDF data under management of a Stardog Server. It may contain zero or more RDF Named Graphs. A Stardog Server may manage more than one Database; there is no hard limit, and the practical limit is disk space.
Database Short Name, aka Database Name	An identifier used to name a database, provided as input when a database is created.
Database Network Name	A Database Short Name is part of the URI of a Database addressed over some network protocol.
Index	The unit of persistence for a Database. We sometimes (sloppily) use Database and Index interchangeably in the manual.
Memory Database	A Database may be stored in-memory or on disk; a Memory Database is read entirely into system memory but can be (optionally) persisted to disk.
Disk Database	A Disk Database is only paged into system memory as needed and is persisted using one or more indexes.
Connection String	An identifier (a restricted subset of legal URLs, actually) that is used to connect to a Stardog database to send queries or perform other operations.
Named Graph	A Named Graph is an explicitly named unit of data within a Database. Named Graphs are queries explicitly by specifying them in SPARQL queries. There is no practical limit on the number of Named Graphs in a Database.
Default Graph	The Default Graph in a Database is the context into which RDF triples are stored when a Named Graph is not explicitly specified. A SPARQL query executed by Stardog that does not contain any Named Graph statements is executed against the data in the Default Graph only.
Security Realm	A Security Realm defines the users and their permissions for each Database in an Stardog Server. There is only one Security Realm per Stardog Server.

APPENDIX

Just move it to the Appendix for a great good!

SPARQL Query Functions

Stardog supports all of the functions in SPARQL, as well as some others from XPath and SWRL. Any of these functions can be used in queries or rules. Function names don’t require namespace prefixes in general unless ambiguity is present. XPath functions take precedence when resolving functions without namespace prefixes. Some functions appear in multiple namespaces, but all of the namespaces will work:

10. Table of Stardog Function Namespaces

Prefix	Namespace
stardog	`tag:stardog:api:functions:`
fn	`http://www.w3.org/2005/xpath-functions# (http://www.w3.org/2005/xpath-functions#)`
math	`http://www.w3.org/2005/xpath-functions/math# (http://www.w3.org/2005/xpath-functions/math#)`
swrlb	`http://www.w3.org/2003/11/swrlb# (http://www.w3.org/2003/11/swrlb#)`
leviathan	`http://www.dotnetrdf.org/leviathan# (http://www.dotnetrdf.org/leviathan#)`
afn	`http://jena.hpl.hp.com/ARQ/function# (http://jena.hpl.hp.com/ARQ/function#)`

The function names and URIs supported by Stardog are included below. Some of these functions exist in SPARQL natively, which just means they can be used without an explicit namespace. Some of the functions have a URI that can be used but they are also overloaded arithmetic operators. For example, if you want to add two day time durations you can simply use the expression `?duration1 + ?duration2` instead of `swrlb:addDayTimeDurations(?duration1 + ?duration2)`.

11. Table of Stardog Function Names & URIs

Function name	Recognized URIs and Symbols
abs	ABS (https://www.w3.org/TR/sparql11-query/#func-abs), fn:numeric-abs (http://www.w3.org/2005/xpath-functions#numeric-abs), swrlb:abs (http://www.w3.org/2003/11/swrlb#abs)
acos	math:acos (http://www.w3.org/2005/xpath-functions/math#acos), leviathan:cos-1 (http://www.dotnetrdf.org/leviathan#cos-1)
addDayTimeDurations	+ , swrlb:addDayTimeDurations (http://www.w3.org/2003/11/swrlb#addDayTimeDurations)
addDayTimeDurationToDate	+ , swrlb:addDayTimeDurationToDate (http://www.w3.org/2003/11/swrlb#addDayTimeDurationToDate)
addDayTimeDurationToDateTime	+ , swrlb:addDayTimeDurationToDateTime (http://www.w3.org/2003/11/swrlb#addDayTimeDurationToDateTime)
addDayTimeDurationToTime	+ , swrlb:addDayTimeDurationToTime (http://www.w3.org/2003/11/swrlb#addDayTimeDurationToTime)
addYearMonthDurations	+ , swrlb:addYearMonthDurations (http://www.w3.org/2003/11/swrlb#addYearMonthDurations)
addYearMonthDurationToDate	+ , swrlb:addYearMonthDurationToDate (http://www.w3.org/2003/11/swrlb#addYearMonthDurationToDate)

addYearMonthDurationToDateTime	`+`, swrlb:addYearMonthDurationToDateTime (http://www.w3.org/2003/11/swrlb#addYearMonthDurationToDateTime)
asin	math:asin (http://www.w3.org/2005/xpath-functions/math#asin), leviathan:sin-1 (http://www.dotnetrdf.org/leviathan#sin-1)
atan	math:atan (http://www.w3.org/2005/xpath-functions/math#atan)
bnode	BNODE (https://www.w3.org/TR/sparql11-query/#func-bnode)
boolean	xsd:boolean (http://www.w3.org/2001/XMLSchema#boolean)
bound	BOUND (https://www.w3.org/TR/sparql11-query/#func-bound)
cartesian	leviathan:cartesian (http://www.dotnetrdf.org/leviathan#cartesian)
ceil	CEIL (https://www.w3.org/TR/sparql11-query/#func-ceil), fn:numeric-ceil (http://www.w3.org/2005/xpath-functions#numeric-ceil), swrlb:ceiling (http://www.w3.org/2003/11/swrlb#ceiling)
coalesce	COALESCE (https://www.w3.org/TR/sparql11-query/#func-coalesce)
concat	CONCAT (https://www.w3.org/TR/sparql11-query/#func-concat), fn:concat (http://www.w3.org/2005/xpath-functions#concat), swrlb:stringConcat (http://www.w3.org/2003/11/swrlb#stringConcat)
contains	CONTAINS (https://www.w3.org/TR/sparql11-query/#func-contains), fn:contains (http://www.w3.org/2005/xpath-functions#contains), swrlb:contains (http://www.w3.org/2003/11/swrlb#contains)
containsIgnoreCase	swrlb:containsIgnoreCase (http://www.w3.org/2003/11/swrlb#containsIgnoreCase)
cos	math:cos (http://www.w3.org/2005/xpath-functions/math#cos), swrlb:cos (http://www.w3.org/2003/11/swrlb#cos), leviathan:cos (http://www.dotnetrdf.org/leviathan#cos)
cosec	leviathan:cosec (http://www.dotnetrdf.org/leviathan#cosec)
cosec-1	leviathan:cosec-1 (http://www.dotnetrdf.org/leviathan#cosec-1)
cosh	`stardog:cosh`
cotan	leviathan:cotan (http://www.dotnetrdf.org/leviathan#cotan)
cotan-1	leviathan:cotan-1 (http://www.dotnetrdf.org/leviathan#cotan-1)
cube	leviathan:cube (http://www.dotnetrdf.org/leviathan#cube)
datatype	Datatype (https://www.w3.org/TR/sparql11-query/#func-datatype)
date	swrlb:date (http://www.w3.org/2003/11/swrlb#date)
dateTime	xsd:dateTime (http://www.w3.org/2001/XMLSchema#dateTime)
day	DAY (https://www.w3.org/TR/sparql11-query/#func-day), fn:day-from-dateTime (http://www.w3.org/2005/xpath-functions#day-from-dateTime), fn:day-from-date (http://www.w3.org/2005/xpath-functions#day-from-date), fn:days-from-duration (http://www.w3.org/2005/xpath-functions#days-from-duration)
dayTimeDuration	swrlb:dayTimeDuration (http://www.w3.org/2003/11/swrlb#dayTimeDuration)
decimal	xsd:decimal (http://www.w3.org/2001/XMLSchema#decimal)
divideDayTimeDuration	`/`, swrlb:divideDayTimeDuration (http://www.w3.org/2003/11/swrlb#divideDayTimeDuration)

divideYearMonthDuration	`/`, swrlb:divideYearMonthDuration (http://www.w3.org/2003/11/swrlb#divideYearMonthDuration)
double	xsd:double (http://www.w3.org/2001/XMLSchema#double)
e	leviathan:e (http://www.dotnetrdf.org/leviathan#e), math:exp (http://www.w3.org/2005/xpath-functions/math#exp), afn:e (http://jena.hpl.hp.com/ARQ/function#e)
encode_for_uri	ENCODE_FOR_URI (https://www.w3.org/TR/sparql11-query/#func-encode_for_uri), fn:encode-for-uri (http://www.w3.org/2005/xpath-functions#encode-for-uri)
factorial	leviathan:factorial (http://www.dotnetrdf.org/leviathan#factorial)
float	xsd:float (http://www.w3.org/2001/XMLSchema#float)
floor	FLOOR (https://www.w3.org/TR/sparql11-query/#func-floor), fn:numeric-floor (http://www.w3.org/2005/xpath-functions#numeric-floor), swrlb:floor (http://www.w3.org/2003/11/swrlb#floor)
gmean	`tag:stardog:api:gmean`
hours	HOURS (https://www.w3.org/TR/sparql11-query/#func-hours), fn:hours-from-dateTime (http://www.w3.org/2005/xpath-functions#hours-from-dateTime), fn:hours-from-time (http://www.w3.org/2005/xpath-functions#hours-from-time), fn:hours-from-duration (http://www.w3.org/2005/xpath-functions#hours-from-duration)
identifier	`tag:stardog:api:identifier`
if	IF (https://www.w3.org/TR/sparql11-query/#func-if)
integer	xsd:integer (http://www.w3.org/2001/XMLSchema#integer)
iri	IRI (https://www.w3.org/TR/sparql11-query/#func-iri), URI (https://www.w3.org/TR/sparql11-query/#func-uri)
isbnode	IsBNode (https://www.w3.org/TR/sparql11-query/#func-isbnode)
isiri	IsIRI (https://www.w3.org/TR/sparql11-query/#func-isiri), IsURI (https://www.w3.org/TR/sparql11-query/#func-isuri)
isliteral	IsLiteral (https://www.w3.org/TR/sparql11-query/#func-isliteral)
isnumeric	IsNumeric (https://www.w3.org/TR/sparql11-query/#func-isnumeric)
isresource	IsResource (https://www.w3.org/TR/sparql11-query/#func-isresource)
lang	Lang (https://www.w3.org/TR/sparql11-query/#func-lang)
langmatches	LangMatches (https://www.w3.org/TR/sparql11-query/#func-langmatches)
lcase	LCASE (https://www.w3.org/TR/sparql11-query/#func-lcase), fn:lower-case (http://www.w3.org/2005/xpath-functions#lower-case), swrlb:lowerCase (http://www.w3.org/2003/11/swrlb#lowerCase)
localname	`stardog:localname`, afn:localname (http://jena.hpl.hp.com/ARQ/function#localname)
log	math:log (http://www.w3.org/2005/xpath-functions/math#log), leviathan:ln (http://www.dotnetrdf.org/leviathan#ln)
log10	math:log10 (http://www.w3.org/2005/xpath-functions/math#log10), leviathan:log (http://www.dotnetrdf.org/leviathan#log)

max	fn:max (http://www.w3.org/2005/xpath-functions#max), afn:max (http://jena.hpl.hp.com/ARQ/function#max)
md5	MD5 (https://www.w3.org/TR/sparql11-query/#func-md5), leviathan:md5hash (http://www.dotnetrdf.org/leviathan#md5hash)
min	fn:min (http://www.w3.org/2005/xpath-functions#min), afn:min (http://jena.hpl.hp.com/ARQ/function#min)
minutes	MINUTES (https://www.w3.org/TR/sparql11-query/#func-minutes), fn:minutes-from-dateTime (http://www.w3.org/2005/xpath-functions#minutes-from-dateTime), fn:minutes-from-time (http://www.w3.org/2005/xpath-functions#minutes-from-time), fn:minutes-from-duration (http://www.w3.org/2005/xpath-functions#minutes-from-duration)
mod	swrlb:mod (http://www.w3.org/2003/11/swrlb#mod)
month	MONTH (https://www.w3.org/TR/sparql11-query/#func-month), fn:month-from-dateTime (http://www.w3.org/2005/xpath-functions#month-from-dateTime), fn:month-from-date (http://www.w3.org/2005/xpath-functions#month-from-date), fn:months-from-duration (http://www.w3.org/2005/xpath-functions#months-from-duration)
multiplyDayTimeDuration	`*`, swrlb:multiplyDayTimeDuration (http://www.w3.org/2003/11/swrlb#multiplyDayTimeDuration)
multiplyYearMonthDuration	`*`, swrlb:multiplyYearMonthDuration (http://www.w3.org/2003/11/swrlb#multiplyYearMonthDuration)
namespace	`stardog:namespace`, afn:namespace (http://jena.hpl.hp.com/ARQ/function#namespace)
normalizeSpace	normalizeSpace (https://www.w3.org/TR/sparql11-query/#func-normalizeSpace), fn:normalize-space (http://www.w3.org/2005/xpath-functions#normalize-space), swrlb:normalizeSpace (http://www.w3.org/2003/11/swrlb#normalizeSpace)
now	NOW (https://www.w3.org/TR/sparql11-query/#func-now)
numeric-add	fn:numeric-add (http://www.w3.org/2005/xpath-functions#numeric-add)
numeric-divide	fn:numeric-divide (http://www.w3.org/2005/xpath-functions#numeric-divide), swrlb:divide (http://www.w3.org/2003/11/swrlb#divide)
numeric-integer-divide	fn:numeric-integer-divide (http://www.w3.org/2005/xpath-functions#numeric-integer-divide), swrlb:integerDivide (http://www.w3.org/2003/11/swrlb#integerDivide)
numeric-multiply	fn:numeric-multiply (http://www.w3.org/2005/xpath-functions#numeric-multiply)
numeric-round-half-to-even	fn:numeric-round-half-to-even (http://www.w3.org/2005/xpath-functions#numeric-round-half-to-even), swrlb:roundHalfToEven (http://www.w3.org/2003/11/swrlb#roundHalfToEven)
numeric-subtract	fn:numeric-subtract (http://www.w3.org/2005/xpath-functions#numeric-subtract), swrlb:subtract (http://www.w3.org/2003/11/swrlb#subtract)
numeric-unary-minus	fn:numeric-unary-minus (http://www.w3.org/2005/xpath-functions#numeric-unary-minus), swrlb:unaryMinus (http://www.w3.org/2003/11/swrlb#unaryMinus)
numeric-unary-plus	fn:numeric-unary-plus (http://www.w3.org/2005/xpath-functions#numeric-unary-plus), swrlb:unaryPlus (http://www.w3.org/2003/11/swrlb#unaryPlus)

pi	math:pi (http://www.w3.org/2005/xpath-functions/math#pi), afn:pi (http://jena.hpl.hp.com/ARQ/function#pi)
pow	math:pow (http://www.w3.org/2005/xpath-functions/math#pow), swrlb:pow (http://www.w3.org/2003/11/swrlb#pow), leviathan:pow (http://www.dotnetrdf.org/leviathan#pow)
pythagoras	leviathan:pythagoras (http://www.dotnetrdf.org/leviathan#pythagoras)
rand	RAND (https://www.w3.org/TR/sparql11-query/#func-rand), leviathan:rnd (http://www.dotnetrdf.org/leviathan#rnd)
reciprocal	leviathan:reciprocal (http://www.dotnetrdf.org/leviathan#reciprocal)
regex	Regex (https://www.w3.org/TR/sparql11-query/#func-regex), fn:matches (http://www.w3.org/2005/xpath-functions#matches), swrlb:matches (http://www.w3.org/2003/11/swrlb#matches)
replace	REPLACE (https://www.w3.org/TR/sparql11-query/#func-replace), fn:replace (http://www.w3.org/2005/xpath-functions#replace), swrlb:replace (http://www.w3.org/2003/11/swrlb#replace)
root	leviathan:root (http://www.dotnetrdf.org/leviathan#root)
round	ROUND (https://www.w3.org/TR/sparql11-query/#func-round), fn:numeric-round (http://www.w3.org/2005/xpath-functions#numeric-round), swrlb:round (http://www.w3.org/2003/11/swrlb#round)
sameTerm	sameTerm (https://www.w3.org/TR/sparql11-query/#func-sameTerm)
sec	leviathan:sec (http://www.dotnetrdf.org/leviathan#sec)
sec-1	leviathan:sec-1 (http://www.dotnetrdf.org/leviathan#sec-1)
seconds	SECONDS (https://www.w3.org/TR/sparql11-query/#func-seconds), fn:seconds-from-dateTime (http://www.w3.org/2005/xpath-functions#seconds-from-dateTime), fn:seconds-from-time (http://www.w3.org/2005/xpath-functions#seconds-from-time), fn:seconds-from-duration (http://www.w3.org/2005/xpath-functions#seconds-from-duration)
sha1	SHA1 (https://www.w3.org/TR/sparql11-query/#func-sha1)
sha256	SHA256 (https://www.w3.org/TR/sparql11-query/#func-sha256), leviathan:sha256hash (http://www.dotnetrdf.org/leviathan#sha256hash)
sha384	SHA384 (https://www.w3.org/TR/sparql11-query/#func-sha384)
sha512	SHA512 (https://www.w3.org/TR/sparql11-query/#func-sha512)
sin	math:sin (http://www.w3.org/2005/xpath-functions/math#sin), swrlb:sin (http://www.w3.org/2003/11/swrlb#sin), leviathan:sin (http://www.dotnetrdf.org/leviathan#sin)
sinh	`stardog:sinh`
sq	leviathan:sq (http://www.dotnetrdf.org/leviathan#sq)
sqrt	math:sqrt (http://www.w3.org/2005/xpath-functions/math#sqrt), afn:sqrt (http://jena.hpl.hp.com/ARQ/function#sqrt), leviathan:sqrt (http://www.dotnetrdf.org/leviathan#sqrt)
str	Str (https://www.w3.org/TR/sparql11-query/#func-str)

strafter	STRAFTER (https://www.w3.org/TR/sparql11-query/#func-strafter), fn:substring-after (http://www.w3.org/2005/xpath-functions#substring-after), swrlb:substringAfter (http://www.w3.org/2003/11/swrlb#substringAfter)
strbefore	STRBEFORE (https://www.w3.org/TR/sparql11-query/#func-strbefore), fn:substring-before (http://www.w3.org/2005/xpath-functions#substring-before), swrlb:substringBefore (http://www.w3.org/2003/11/swrlb#substringBefore)
strdt	STRDT (https://www.w3.org/TR/sparql11-query/#func-strdt)
strends	STRENDS (https://www.w3.org/TR/sparql11-query/#func-strends), fn:ends-with (http://www.w3.org/2005/xpath-functions#ends-with), swrlb:endsWith (http://www.w3.org/2003/11/swrlb#endsWith)
string	xsd:string (http://www.w3.org/2001/XMLSchema#string)
stringEqualIgnoreCase	swrlb:stringEqualIgnoreCase (http://www.w3.org/2003/11/swrlb#stringEqualIgnoreCase)
strlang	STRLANG (https://www.w3.org/TR/sparql11-query/#func-strlang)
strlen	STRLEN (https://www.w3.org/TR/sparql11-query/#func-strlen), fn:string-length (http://www.w3.org/2005/xpath-functions#string-length), swrlb:stringLength (http://www.w3.org/2003/11/swrlb#stringLength)
strstarts	STRSTARTS (https://www.w3.org/TR/sparql11-query/#func-strstarts), fn:starts-with (http://www.w3.org/2005/xpath-functions#starts-with), swrlb:startsWith (http://www.w3.org/2003/11/swrlb#startsWith)
struuid	STRUUID (https://www.w3.org/TR/sparql11-query/#func-struuid)
substr	SUBSTR (https://www.w3.org/TR/sparql11-query/#func-substr), fn:substring (http://www.w3.org/2005/xpath-functions#substring), swrlb:substring (http://www.w3.org/2003/11/swrlb#substring)
subtractDates	– , swrlb:subtractDates (http://www.w3.org/2003/11/swrlb#subtractDates)
subtractDayTimeDurationFromDate	– , swrlb:subtractDayTimeDurationFromDate (http://www.w3.org/2003/11/swrlb#subtractDayTimeDurationFromDate)
subtractDayTimeDurationFromDateTime	– , swrlb:subtractDayTimeDurationFromDateTime (http://www.w3.org/2003/11/swrlb#subtractDayTimeDurationFromDateTime)
subtractDayTimeDurationFromTime	– , swrlb:subtractDayTimeDurationFromTime (http://www.w3.org/2003/11/swrlb#subtractDayTimeDurationFromTime)
subtractDayTimeDurations	– , swrlb:subtractDayTimeDurations (http://www.w3.org/2003/11/swrlb#subtractDayTimeDurations)
subtractTimes	– , swrlb:subtractTimes (http://www.w3.org/2003/11/swrlb#subtractTimes)
subtractYearMonthDurationFromDate	– , swrlb:subtractYearMonthDurationFromDate (http://www.w3.org/2003/11/swrlb#subtractYearMonthDurationFromDate)
subtractYearMonthDurationFromDateTime	– , swrlb:subtractYearMonthDurationFromDateTime (http://www.w3.org/2003/11/swrlb#subtractYearMonthDurationFromDateTime)
subtractYearMonthDurations	– , swrlb:subtractYearMonthDurations (http://www.w3.org/2003/11/swrlb#subtractYearMonthDurations)
tan	math:tan (http://www.w3.org/2005/xpath-functions/math#tan), swrlb:tan (http://www.w3.org/2003/11/swrlb#tan)
tanh	`stardog:tanh`

ten	leviathan:ten (http://www.dotnetrdf.org/leviathan#ten)
time	swrlb:time (http://www.w3.org/2003/11/swrlb#time)
timezone	TIMEZONE (https://www.w3.org/TR/sparql11-query/#func-timezone), fn:timezone-from-dateTime (http://www.w3.org/2005/xpath-functions#timezone-from-dateTime), fn:timezone-from-date (http://www.w3.org/2005/xpath-functions#timezone-from-date), fn:timezone-from-time (http://www.w3.org/2005/xpath-functions#timezone-from-time)
toDegrees	`stardog:toDegrees` , leviathan:radians-to-degrees (http://www.dotnetrdf.org/leviathan#radians-to-degrees)
toRadians	`stardog:toRadians` , leviathan:degrees-to-radians (http://www.dotnetrdf.org/leviathan#degrees-to-radians)
translate	TRANSLATE (https://www.w3.org/TR/sparql11-query/#func-translate), fn:translate (http://www.w3.org/2005/xpath-functions#translate), swrlb:translate (http://www.w3.org/2003/11/swrlb#translate)
tz	TZ (https://www.w3.org/TR/sparql11-query/#func-tz)
ucase	UCASE (https://www.w3.org/TR/sparql11-query/#func-ucase), fn:upper-case (http://www.w3.org/2005/xpath-functions#upper-case), swrlb:upperCase (http://www.w3.org/2003/11/swrlb#upperCase)
uuid	UUID (https://www.w3.org/TR/sparql11-query/#func-uuid)
year	YEAR (https://www.w3.org/TR/sparql11-query/#func-year), fn:year-from-dateTime (http://www.w3.org/2005/xpath-functions#year-from-dateTime), fn:year-from-date (http://www.w3.org/2005/xpath-functions#year-from-date), fn:years-from-duration (http://www.w3.org/2005/xpath-functions#years-from-duration)
yearMonthDuration	swrlb:yearMonthDuration (http://www.w3.org/2003/11/swrlb#yearMonthDuration)

Milestones

This timeline describes major features and other notable changes to Stardog starting at 1.0; it will be updated for each notable new release. For a complete list of changes, including notable bug fixes, see the release notes (/docs/release-notes/).

4.1

- Multi-coordinator Cluster
- Cluster stability & performance improvements
- Query graph & revision history simultaneously
- New, faster SPARQL parser

4.0

- TinkerPop3 and property graphs
- virtual graphs
- RDF 1.1
- Java 8
- Geospatial query answering

3.1	▪ Named Graph security ▪ BOSH-based cluster management tool ▪ proper logshipping in the Cluster
3.0	▪ Equality reasoning via hybrid materialization ▪ Improved incremental write performance ▪ HA Cluster production ready ▪ Integrity constraint violation repair plans ▪ Improved query performance
2.2.1	▪ Stardog HA Cluster (beta)
2.2	▪ Support for RDF versioning ▪ Admin support for Web Console
2.1	▪ Database repair, backup & restore utilities ▪ Improved query scalability by flowing intermediate results off-heap or onto disk; requires a JDK that supports `sun.misc.Unsafe` ▪ Performance (#Performance Benchmark Results): significant improvement in performance of bulk loading and total scalability of a database ▪ Generation of multiple proofs for inferences & inconsistencies; proofs for integrity constraint violations ▪ Reduced memory footprint of queries while being executed
2.0	▪ SPARQL 1.1 Update (#_using_stardog): the most requested feature ever! ▪ Stardog Web Console (#_browsing): a Stardog Web app for managing Stardog Databases; includes Linked Data Server, etc. ▪ JMX monitoring (#Server Monitoring with Watchdog & JMX): includes graphical monitoring via Web Console ▪ HTTP & SNARL servers unified into a single server (default port 5820) ▪ Database Archetypes (#_database_archetypes) for PROV, SKOS; extensible for user-defined ontologies, schemas, etc. ▪ Stardog Rules Syntax (#_stardog_rules_syntax): new syntax for user-defined rules ▪ Performance improvements for SPARQL query evaluation ▪ Hierarchical explanations (#_proof_trees) of inferences using proof trees ▪ SL (#Reasoning Levels) reasoning profile ▪ Client and server dependencies cleanly separated ▪ Evaluation of non-recursive datalog queries to improve reasoning performance
1.2	▪ Query management: slow query log, kill-able queries, etc. ▪ new CLI ▪ new transaction layer ▪ SPARQL Service Description ▪ new security layer ▪ Query rewrite cache ▪ Removed Stardog Shell

1.1.2	▪ New optimizer for subqueries
1.1	▪ SPARQL 1.1 Query ▪ Transitive reasoning ▪ User-defined rules in SWRL ▪ new SWRL builtins and syntactic sugar for schema-queries ▪ Improved performance of reasoning queries involving rdf:type ▪ Improved performance of search indexing ▪ Deprecated Stardog Shell
1.0.4	▪ Convert ICVs to SPARQL queries in the CLI or Java API ▪ Running as a Windows Service ▪ Parametric queries in CLI
1.0.2	▪ Stardog Community edition introduced ▪ ICV in SNARL and HTTP ▪ HTTP Admin protocol extensions ▪ SPARQL 1.1 Graph Store Protocol
1.0.1	▪ Self-hosting Stardog documentation ▪ Prefix mappings per database ▪ Access and audit logging
1.0	▪ Execute `DESCRIBE` queries against multiple resources ▪ Database consistency checking from CLI ▪ Inference explanations from CLI

Previous Versions of Docs

12. Table of Historical Docs

Stardog Version	HTML	PDF
5.3.6	✓ (/docs/#)	✓ (/docs/stardog-manual-5.3.6.pdf)
4.2	✓ (/docs/4.2/#)	✓ (/docs/stardog-manual-4.2.pdf)
4.1.3	✓ (/docs/4.1.3/#)	✓ (/docs/stardog-manual-4.1.3.pdf)
4.1.2	✓ (/docs/4.1.2/#)	✓ (/docs/stardog-manual-4.1.2.pdf)
4.1.1	✓ (/docs/4.1.1/#)	✓ (/docs/stardog-manual-4.1.1.pdf)
4.1	✓ (/docs/4.1#)	✓ (/docs/stardog-manual-4.1.pdf)
4.0.5	✓ (/docs/4.0.5#)	✓ (/docs/stardog-manual-4.0.5.pdf)
4.0.3	✓ (/docs/4.0.3#)	✓ (/docs/stardog-manual-4.0.3.pdf)
4.0.2	✓ (/docs/4.0.2#)	✓ (/docs/stardog-manual-4.0.2.pdf)

4.0.1	✓ (/docs/4.0.1#)	✓ (/docs/stardog-manual-4.0.1.pdf)
4.0	✓ (/docs/4.0#)	✓ (/docs/stardog-manual-4.0.pdf)

---

1. Robert Butler, Marko A. Rodriguez, Brian Sletten, Alin Dreghiciu, Rob Vesse, Stephane Fallah, John Goodwin, José Devezas, Chris Halaschek-Wiener, Gavin Carothers, Brian Panulla, Ryan Kohl, Morton Swimmer, Quentin Reul, Paul Dlug, James Leigh, Alex Tucker, Ron Zettlemoyer, Jim Rhyne, Andrea Westerinen, Huy Phan, Zach Whitley, Maurice Rabb, Grant Pax, Conrad Leonard, John Shearer, Ryan Hohimer, and the crew at XSB.
2. If the same name is used for different functions in different namespaces then the precedence is given to the standard functions. It is best practice to use the explicit namespace for such functions to avoid ambiguity.
3. The last SPARQL 1.1 feature that we didn't support.
4. See the R2RML spec (http://www.w3.org/TR/r2rml/) for more information about R2RML.
5. User feedback about this limitation is welcomed.
6. Find another database that can do that!
7. In other words, if there is a conflict between this documentation and the output of the CLI tools' `help` command, the CLI output is correct.
8. We're big fans of /opt/stardog/{$version} and setting `STARDOG_HOME` to /var/stardog` but YMMV.
9. This is equally true, when using Stardog HA Cluster, of Zookeeper's access to free disk space. Bad things happen to the Stardog Cluster if Zookeeper cannot write to disk.
10. For more details about configuring these values, see https://github.com/Complexible/stardog-examples/blob/master/config/stardog.properties (https://github.com/Complexible/stardog-examples/blob/master/config/stardog.properties).
11. However, there may be some delay since Stardog only periodically checks the `query.timeout` value against internal query evaluation timers.
12. A good general purpose discussion of these issues in context of J2EE is this beginner's guide (http://vladmihalcea.com/2014/12/23/a-beginners-guide-to-transaction-isolation-levels-in-enterprise-java/).
13. As discussed in SPARQL Update, since Update queries are implicitly atomic transactional operations, which means you shouldn't issue an Update query within an open transaction.
14. The probability of recovering from a catastrophic transaction failure is inversely proportional to the number of subsequent write attempts; hence, Stardog offlines the database to prevent subsequent write attempts and to increase the likelihood of recovery.
15. Stardog also uses file handles and sockets, but we don't discuss those here.
16. These are conservative values and are dataset specific. Your data may require less memory…or more!
17. For more details about configuring these values, see https://github.com/Complexible/stardog-examples/blob/master/config/stardog.properties (https://github.com/Complexible/stardog-examples/blob/master/config/stardog.properties).
18. Blob Indexing and Text Enrichment with Semantics
19. "Client" here means the client of Stardog APIs.
20. "Because ZooKeeper requires a majority, it is best to use an odd number of machines. For example, with four machines ZooKeeper can only handle the failure of a single machine; if two machines fail, the remaining two machines do not constitute a majority. However, with five machines ZooKeeper can handle the failure of two machines." See Zk Admin (https://zookeeper.apache.org/doc/r3.1.2/zookeeperAdmin.html) for more.
21. Based on customer feedback we may relax these consistency guarantees in some future release. Please get in touch if you think an eventually consistent approach is more appropriate for your use of Stardog.
22. This point is especially true of Cluster but may be relevant for some workloads on a single Stardog database, that is, non-Cluster configuration, too.
23. You only pay for the reasoning that you use; no more and no less. Eager materialization is mostly a great strategy for hard disk manufacturers.
24. Sometimes called a "TBox".
25. Find another database, any other database anywhere, that can do that! We'll wait…
26. Triggered using the `--format tree` option of the `reasoning explain` CLI command.
27. Quick refresher: the `IF` clause defines the conditions to match in the data; if they match, then the contents of the `THEN` clause "fire", that is, they are inferred and, thus, available for other queries, rules, or axioms, etc.
28. Of course if you've tweaked `reasoning.schema.graphs`, then you should put the rules into the named graph(s) that are specified in that configuration parameter.
29. Built-in URIs such as `rdfs:subClassOf` or `owl:TransitiveProperty` are not allowed in rules
30. This is effectively the only setting for Stardog prior to 3.0.
31. These are harmless and won't otherwise affect query evaluation; they can also be added to the data, instead of to queries, if that fits your use case better.
32. The standard inference semantics of OWL 2 do not adopt the unique name assumption because, in information integration scenarios, things often have more than one name but that doesn't mean they are different things. For example, when several databases or other data sources all contain some partial information about, say, an employee, but they each name or identify the employee in different ways. OWL 2 won't assume these are different employees just because there are several names.
33. Strictly speaking, this is a bit misleading. Stardog ICV uses both open and closed world semantics: since inferences can violate or satisfy constraints, and Stardog uses open world semantics to calculate inferences, then the ICV process is compatible with open world reasoning, to which it then applies a form of closed world validation, as described in this chapter.
34. This is a good example of open world and closed world reasoning interacting for the win.
35. In other words, embedded Stardog access is inherently **insecure** and should be used accordingly.
36. The Stardog client uses an `X509TrustManager`. The details of how a trust store is selected to initialize the trust manager are http://docs.oracle.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html#X509TrustManager (http://docs.oracle.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html#X509TrustManager).
37. See the `javax.net.ssl.trustStorePassword` system property docs: http://docs.oracle.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html#X509TrustManager (http://docs.oracle.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html#X509TrustManager).
38. The matching algorithm used is described—http://hc.apache.org/httpcomponents-client-ga/tutorial/html/connmgmt.html-- (http://hc.apache.org/httpcomponents-client-ga/tutorial/html/connmgmt.html--) in the Apache docs about `BrowserCompatHostnameVerifier`.

39. You won’t be careful enough.

40. The relevant specs include the Stardog-specific specifications documented on this site, but also W3C (and other) specifications of various languages, including SPARQL, RDF, RDFS, OWL 2, HTTP, Google Protocol Buffers, as well as others.

41. Strictly speaking, this is a Sesame parser deviation from the SPARQL 1.1 spec with which we happen to agree.

Version 5.3.6