MarkLogic サーバー

Semantics Developer’s Guide

MarkLogic 9
2017 年 5 月

最終更新：9.0-2、2017 年 7 月

目次

Semantics Developer's Guide

1.0 MarkLogic におけるセマンティックの概要

セマンティックを使用すると、データ内のファクトおよび関係性を発見できるようになり、そのようなファクトのコンテキストが提供されます。セマンティック技術では、一連の固有の W3C 標準を参照して、データ内の関係性に関する情報をマシンで認識可能な形式で交換できるようにします。情報が Web 上にあるのか組織内にあるのかは問いません。MarkLogic のセマンティックでは RDF (Resource Description Framework) を採用しています。SPARQL クエリ、SPARQL Update、JavaScript、XQuery、REST を使用した RDF トリプル（RDF triple）のネイティブな格納、検索、および管理が可能です。

セマンティックには、柔軟なデータモデル（RDF）、クエリツール（SPARQL）、グラフおよびトリプルデータ管理ツール（SPARQL Update）、そして共通のマークアップ言語（RDFa、Turtle、N-Triples など）が必要です。MarkLogic では、SPARQL および SPARQL Update を使用してトリプルをネイティブに格納、管理、および検索できます。

RDF は、リンクトオープンデータの核となる技術の 1 つです。このフレームワークは、データからあいまいさを排除したり、まったく異なるソースに由来するデータを統合したり操作したりするための標準を提供します。マシンが認識可能な形式と人間が認識可能な形式のどちらにも対応しています。データをセマンティック Web でパブリッシュしたり共有したりする場合は、W3C 勧告および公式に定義された語彙を利用します。

RDF シリアライゼーションでデータをクエリするときは、SPARQL（SPARQL プロトコルおよび RDF クエリ言語）を使用します。また、トリプルデータおよびグラフを作成、削除、更新（削除 / 挿入）するときは、SPARQL Update を使用します。

推論（inference）を利用して、追加のセマンティック情報をデータから生成できます。また、LOD（リンクトオープンデータ）を使用して、データをエンリッチすることもできます。LOD は、データに埋め込まれた追加のセマンティックメタデータから作成される World Wide Web の拡張機能です。

> **注：** セマンティックは、個別のライセンス製品になります。SPARQL 機能を使用するには、Semantics Option を含むライセンスが必須です。SPARQL を使わずセマンティックを活用する API（オプティック API や SQL API など）を使用する場合は、Semantics Option ライセンスは不要です。

詳細については、次のリソースを参照してください。

- http://www.w3.org/standards/semanticweb
- http://www.w3.org/RDF
- http://www.w3.org/TR/rdf-sparql-query
- http://www.w3.org/TR/sparql11-update

このドキュメントでは、MarkLogic サーバーのセマンティックデータを読み込み、クエリ、および操作する方法について説明します。また、この章では、MarkLogic サーバーにおけるセマンティックの概要について説明します。この章は、次のセクションで構成されています。

1.1 用語

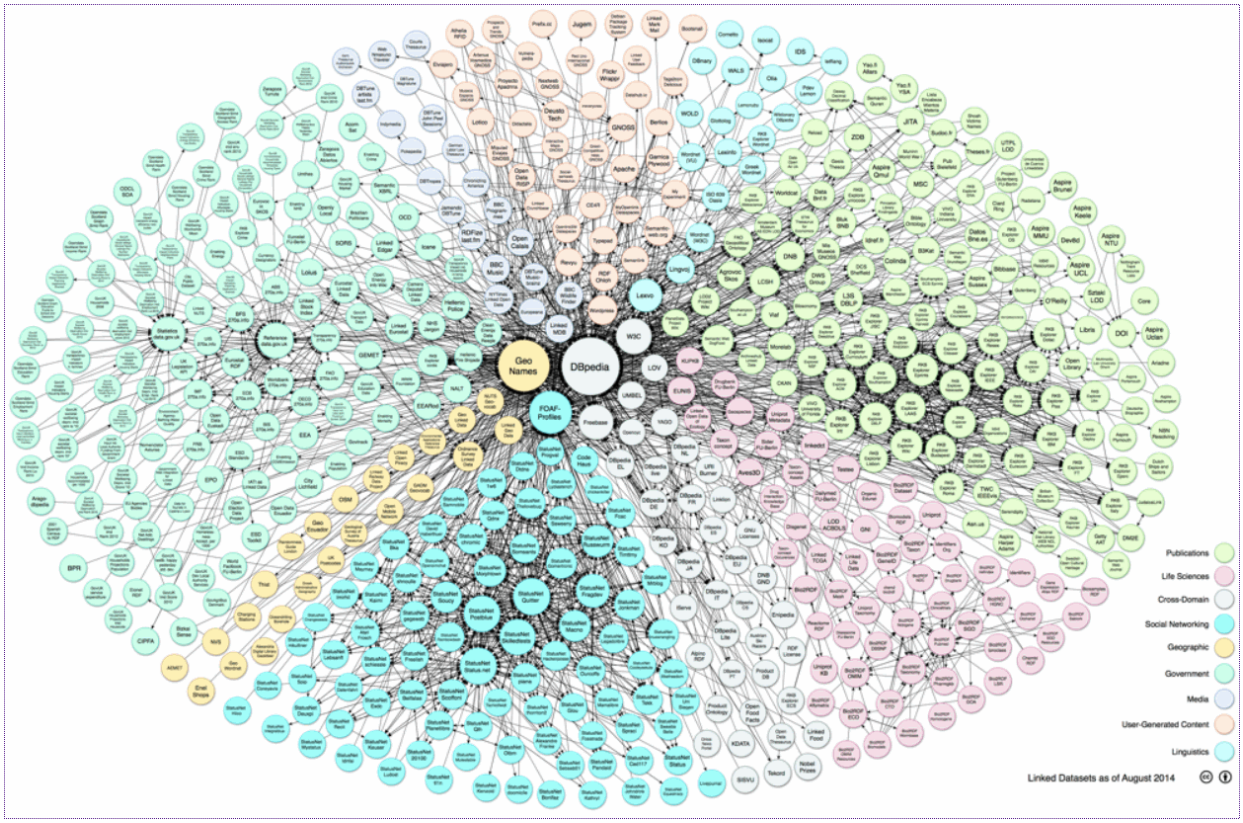
このガイドで使用する用語は、次のとおりです。

用語	定義
RDF	RDF (Resource Description Framework) は、主語、述語、および目的語で構成されたトリプルとしてファクトを表現するために使用するデータモデルです。このフレームワークは定義済みの語彙を持つ W3C 規格です。
RDF トリプル	主語、述語、目的語、およびグラフ（オプション）を表現するアトミック値が含まれた RDF ステートメント。各トリプルが 1 つのファクトを表現します。
主語	人物やエンティティといったリソースの表現。グラフまたはトリプル内のノード。
述語	主語のプロパティや特徴を表現したもの。あるいは主語と目的語の関係を表現したもの。述語は、アークまたはエッジとも呼ばれます。
目的語	プロパティ値を表現するノード。他のトリプルやグラフの主語になることもあります。また、目的語は型付きリテラルになることがあります。「RDF データ型」（13 ページ）を参照してください。
グラフ	RDF トリプルのステートメントまたはパターンの集まり。グラフベースの RDF モデルでは、ノードは主語または目的語リソースを表現し、述語はこのようなノード間の接続を提供します。名前が割り当てられたグラフは、名前付きグラフとも呼ばれます。
クアッド	主語、述語、目的語、およびトリプルのコンテキストを表す追加のリソースノードから構成された表現。

用語	定義
語彙	タームを分類する標準的な形式。FOAF（Friend of a Friend）および DC（ダブリンコア）などの語彙は、ファクトを記述および表現するために使用する概念や関係性を定義します。例えば OWL は、World Wide Web 上でオントロジーをパブリッシュおよび共有するための Web オントロジー言語です。
トリプルインデックス	SPARQL クエリの実行を容易にするために、MarkLogic に読み込まれるトリプルに付けられるインデックス。
RDF トリプルストア	RDF グラフに対する永続的な格納、インデックス付け、およびクエリアクセスに対応したストレージツール。
IRI	IRI (Internationalized Resource Identifier) は、RDF トリプル内のリソースを一意に識別する目的で使用する短縮文字列です。IRI には、中国語や日本語の漢字、韓国語、キリル文字など、国際符号化文字集合（Unicode/ISO 10646）の文字を含めることができます。
CURIE	短縮 URI 表現（Compact URI Expression）。
SPARQL	SPARQL プロトコルおよび RDF クエリ言語（SPARQL Protocol and RDF Query Language）の再帰的な頭字語。RDF シリアライゼーションでデータをクエリするために設計されたクエリ言語です。MarkLogic では、SPARQL 1.1 のシンタックスおよび関数を使用できます。
SPARQL プロトコル	クエリクライアントからクエリプロセッサへ SPARQL クエリを伝達する手段。抽象インターフェイスと、HTTP（Hypertext Transfer Protocol）および SOAP（Simple Object Access Protocol）に対するバインドで構成されます。
SPARQL Update	SPARQL クエリ言語から派生したシンタックスを使用する、RDF グラフの更新言語。
RDFa	RDFa（Resource Description Framework in Attributes）は、リッチなメタデータを Web ドキュメント内に埋め込むために属性レベルの拡張機能の集まりを HTML、XHTML、および各種の XML ベースのドキュメントタイプに追加する W3C 勧告です。
空白ノード	IRI またはリテラルが提供されないリソースを表現する、RDF グラフ内のノード。「b ノード」と呼ばれることもあります。

1.2 リンクトオープンデータ

リンクトオープンデータを使用すると、メタデータおよびデータを Web 上で共有できます。World Wide Web では、人間が認識できる Web ページおよびハイパーリンクとして構造化および非構造化データのリソースにアクセスできます。リンクトオープンデータは、ページおよびページ間の関係性についてマシンが認識できるメタデータを挿入することでこれを拡張し、セマンティック的に構造化された知識を表現します。リンクトオープンデータクラウドでは、Web 上で使用可能なオープンデータセットの多様性についての知見を得られます。



リンクトオープンデータの詳細については、http://linkeddata.org/ を参照してください。

1.3 MarkLogic における RDF 実装

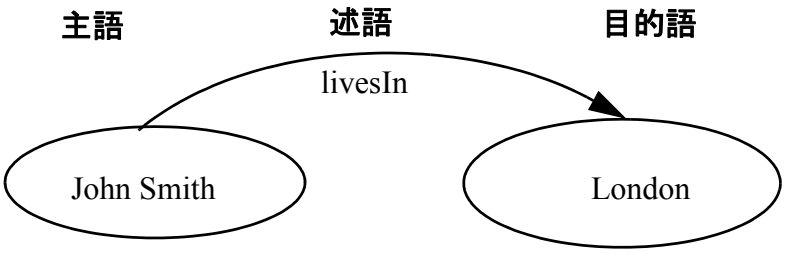
このセクションでは、MarkLogic サーバーに実装されている RDF を使用したセマンティック技術について説明します。このセクションは、次の概念で構成されています。

- MarkLogic での RDF の使用
- RDF データモデル
- RDF データ型
- RDF 語彙

1.3.1 MarkLogic での RDF の使用

RDF は、RDF トリプルを格納および検索する目的で MarkLogic に実装されています。具体的には、各トリプルは主語、述語、目的語、およびグラフ（オプション）が含まれた RDF トリプルステートメントです。

例えば以下のようになります。



主語ノードは John Smith という名前のリソースで、オブジェクトノードは London です。述語は 2 つのノードをリンクするエッジとして示され、関係性を表します。この例からは、John Smith lives in London（ジョン・スミスはロンドンに住んでいる）というステートメントを生成できます。

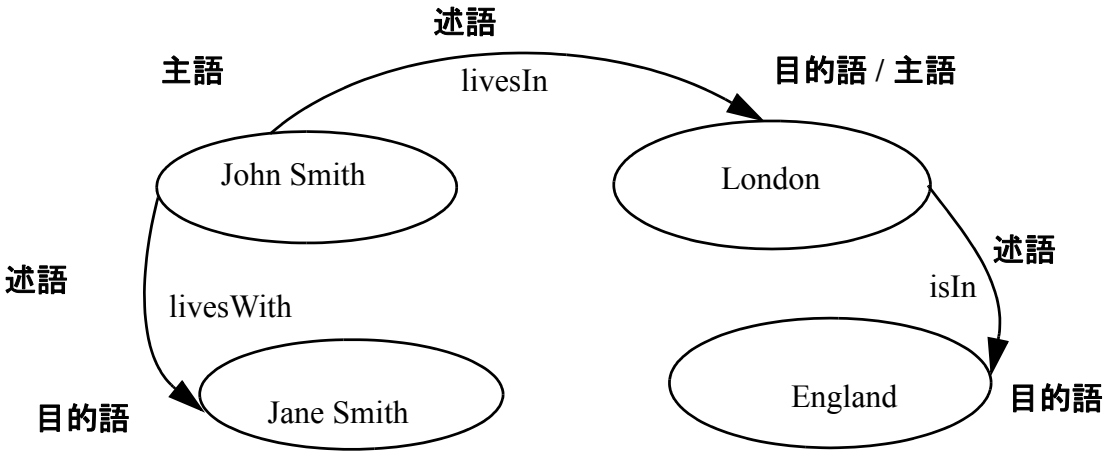
このトリプルを XML で表現すると次のようになります（2 つ目のトリプルが追加されています：

```
<sem:triples xmlns:sem="http://marklogic.com/semantics">
  <sem:triple>
    <sem:subject> http://xmlns.com/foaf/0.1/name/"John
Smith"</sem:subject>
    <sem:predicate> http://example.org/livesIn</sem:
    predicate>
    <sem:object
datatype="http://www.w3.org/2001/XMLSchema#string">
"London"</sem:object>
  </sem:triple>
</sem:triples>
```

JSON では、この同じトリプルは次のようになります。

```
{
  "my" : "data",
    "triple" : {
       "subject": "http://xmlns.com/foaf/0.1/name/
       John Smith",
       "predicate": "http://example.org/livesIn",
      "object": { "value": "London", "datatype": "xs:string" }
  }
}
```

トリプルの集まりは、RDF グラフとして格納されます。MarkLogic では、グラフはコレクションとして格納されます。次の図は、3 つのトリプルが含まれたシンプルな RDF グラフモデルの例です。グラフの詳細については、「RDF データモデル」（11 ページ）を参照してください。



トリプルの目的語ノードは、今度は別のトリプルの主語ノードになることができます。この例では、「John Smith lives with Jane Smith」（ジョン・スミスはジェーン・スミスと一緒に生活している）、「John Smith lives in London」（ジョン・スミスはロンドンに住んでいる）、「London is in England」（ロンドンはイングランドにある）というファクトを表現しています。

グラフは表形式で表現できます。

主語	述語	目的語
John Smith	livesIn	London
London	isIn	England
John Smith	livesWith	Jane Smith

これらのトリプルを JSON で表現すると、次のようになります。

```
{
  "my" : "data",
    "triples" : [{
      "subject": "http://xmlns.com/foaf/0.1/name/John Smith",
      "predicate": "http://example.org/livesIn",
      "object": { "value": "London", "datatype": "xs:string"
}
  },{
```

```
        "subject": "http://xmlns.com/foaf/0.1/name/London",
        "predicate": "http://example.org/isIn",
        "object": { "value": "England", "datatype": "xs:string"
}
  },{
        "subject": "http://xmlns.com/foaf/0.1/name/John Smith",
        "predicate": "http://example.org/livesWith",
        "object": { "value": "Jane Smith", "datatype":
        "xs:string" }
}
]}
```

1.3.1.1 MarkLogic での RDF トリプルの格納

RDF トリプルを MarkLogic にロードすると、そのトリプルは MarkLogic が管理する XML ドキュメントに格納されます。トリプルは、Turtle や N-Triples などの RDF シリアライゼーションを使用して、ドキュメントからロードできます。例えば以下のようになります。

```
<http://example.org/dir/js>
<http://xmlns.com/foaf/0.1/firstname> "John" .
<http://example.org/dir/js>
<http://xmlns.com/foaf/0.1/lastname> "Smith" .
<http://example.org/dir/js>
<http://xmlns.com/foaf/0.1/knows> "Jane Smith" .
```

RDF 形式の詳細な例については、「RDF 形式の例」（32 ページ）を参照してください。

この例のトリプルは、XML ドキュメントとして MarkLogic に格納され、ドキュメントルートは `sem:triples` になります。これらのトリプルには、ドキュメントルート要素 `sem:triples` があるので、managed triples だとわかります。

```
<?xml version="1.0" encoding="UTF-8"?>
<sem:triples xmlns:sem="http://marklogic.com/semantics">
 <sem:triple>
  <sem:subject>http://example.org/dir/js</sem:subject>
  <sem:predicate>http://xmlns.com/foaf/0.1/firstname</sem:
  predicate>
  <sem:object datatype="http://www.w3.org/2001/
  XMLSchema#string">John
  </sem:object>
 </sem:triple>
 <sem:triple>
```

```
    <sem:subject>http://example.org/dir/js</sem:subject>
    <sem:predicate>http://xmlns.com/foaf/0.1/lastname</
    sem:predicate>
    <sem:object datatype="http://www.w3.org/2001/
    XMLSchema#string">
        Smith</sem:object>
  </sem:triple>
  <sem:triple>
    <sem:subject>http://example.org/dir/js</sem:subject>
    <sem:predicate>http://xmlns.com/foaf/0.1/knows</
    sem:predicate>
    <sem:object datatype="http://www.w3.org/2001/
    XMLSchema#string">
        Jane Smith</sem:object>
  </sem:triple>
</sem:triples>
```

また、トリプルを XML ドキュメントに埋め込み、MarkLogic にそのままロードすることもできます。この場合は unmanaged triples になり、要素ノード `sem:triple` を持ちます。非管理対象トリプルには外側の `sem:triples` 要素は必要ありませんが、主語、述語、目的語要素が `sem:triple` 要素内に必要になります。

XML ドキュメントに含まれる埋め込みトリプルの例を次に示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<article>
  <info>
    <title>News for April 9, 2013</title>
     <sem:triples xmlns:sem="http://marklogic.com/
     semantics">
       <sem:triple>
        <sem:subject>http://example.org/article</
        sem:subject>
        <sem:predicate>http://example.org/mentions</
        sem:predicate>
        <sem:object>http://example.org/London</sem:object>
       <sem:triple>
     </sem:triples>
...
  </info>
</article>
```

ロードしたトリプルには、トリプルインデックス（triple index）と呼ばれる専用のインデックスが自動的に付けられます。トリプルインデックスにより、必要な権限を保持しているRDF データについては、ただちに検索できます。

1.3.1.2 トリプルのクエリ

Query Console では、MarkLogic 内またはインメモリに格納された RDF トリプルから情報を取得するネイティブの SPARQL クエリを記述できます。SPARQL を使用してクエリすると、「誰がイングランドに住んでいるか」という質問に対して「ジョンおよびジェーン・スミス」という答えが返されます。これは、前述のグラフモデルからのファクトの表明に基づきます。シンプルな SPARQL の `SELECT` クエリの例を次に示します。

```
SELECT ?person ?place
WHERE
{
  ?person <http://example.org/LivesIn> ?place .
  ?place <http://example.org/IsIn> "England".
}
```

SPARQL クエリを実行するために `sem:sparql` を利用した XQuery を使用することもできます。例えば以下のようになります。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/
semantics" at
  "/MarkLogic/semantics.xqy";

sem:sparql("
PREFIX kennedy:<http://example.org/kennedy>
SELECT *
WHERE
{
?s ?p ?o .
FILTER (regex(?o, 'Joseph', 'i'))
}
")
```

トリプルをクエリするために SPARQL および `sem:sparql` を使用する方法の詳細については、「セマンティッククエリ」（71 ページ）を参照してください。

XQuery では、`cts:triples` や `cts:triple-range-query` を指定すると、トリプル、ドキュメント、および値全体にクエリできます。

cts:triples クエリを使用する例を次に示します。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/
  semantics" at "/MarkLogic/semantics.xqy";

let $r :=
  cts:triples(sem:iri("http://example.org/people/dir"),
    sem:iri("http://xmlns.com/foaf/0.1/knows"),
    sem:iri("person1"))

return <result>{$r}</result>
```

cts:triple-range-query を使用するクエリの例を次に示します。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/
  semantics" "at /MarkLogic/semantics.xqy";

let $query := cts:triple-range-query(
  sem:iri("http://example.org/people/dir"),
  sem:iri("http://xmlns.com/foaf/0.1/knows"), ("person2"),
  "sameTerm")

return cts:search(fn:collection()//sem:triple, $query)
```

cts:query 関数（cts:or-query、cts:and-query など）を使用して、複合クエリを作成できます。

例えば以下のようになります。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/
  semantics" at "MarkLogic/semantics.xqy";

declare namespace dc = "http://purl.org/dc/elements/1.1/";

  cts:search(collection()//sem:triple, cts:or-query((
    cts:triple-range-query((),
    sem:curie-expand("foaf:name"),
      "Lamar Alexander", "="),
   cts:triple-range-query(sem:iri("http://www.rdfabout.com/
     rdf/usgov/congress/people/A000360"), sem:curie-expand
     ("foaf:img"), (),
  "="))))
```

`cts:triples` および `cts:triple-range-query` クエリの詳細については、「セマンティッククエリ」（71 ページ）を参照してください。

SPARQL クエリの結果と XQuery 検索を組み合わせて、複合クエリを作成することもできます。

例えば以下のようになります。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/
  semantics" at "/MarkLogic/semantics.xqy";
import module namespace semi = "http://marklogic.com/
  semantics/impl" at "/MarkLogic/semantics/sem-impl.xqy";

declare namespace sr = "http://www.w3.org/2005/
  sparql-results";

let $results := sem:sparql( "prefix k: <http://example.org/
  kennedy>
select * { ?s k:latitude ?lat .?s k:longitude ?lon }" )
let $xml := semi:sparql-results-xml($results)

return
for $sol in $xml/sr:results/sr:result
let $point := cts:point(xs:float($sol/sr:binding[@name eq
'lat']/sr:literal), xs:float($sol/sr:binding[@name eq
'lon']/sr:literal))
return <place name="{$sol/sr:binding[@name eq 's']/*}"
point="{$point}"/>
```

複合クエリの詳細については、「XQuery または JavaScript によるトリプルのクエリ」（125 ページ）を参照してください。

1.3.2 RDF データモデル

RDF トリプルを使用すると、ファクト（世界に関するファクト、ドメインに関するファクト、ドキュメントに関するファクト）を簡単に表現できます。各 RDF トリプルは、主語、述語、および目的語で表現されるファクト（表明）であり、「John livesIn London」（ジョンはロンドンに住んでいる）などになります。トリプルの主語と述語は、IRI（Internationalized Resource Identifier）である必要があります。IRI はリソースを一意に識別するために使用される短縮文字列です。目的語は、IRI またはリテラル（数値や文字列など）のいずれかになります。

- 主語と述語は IRI 参照であり、オプションでフラグメント識別子が付きます。例えば以下のようになります。

```
<http://xmlns.com/foaf/0.1/Person>
foaf:person
```

- リテラルとは、文字列（オプションで言語タグが付く）または数値です。RDF トリプルでは目的語として使用されます。例えば以下のようになります。

```
"Bob"
"chat" @fr
```

- 型付きリテラルは、文字列、整数、日付など、データ型に割り当てられたリテラルです。これらのリテラルは、「^^」演算子で型付けされます。例えば以下のようになります。

```
"Bob"^^xs:string
"3"^^xs:integer
"26.2"^^xs:decimal
```

主語または目的語は、空白ノード（b ノードまたは無名ノード）にすることもできます。これは、グラフでは名前のないノードになります。空白ノードはアンダースコアで表現され、その後にコロン（:）と識別子が続きます。例えば以下のようになります。

```
_:a
_:jane
```

IRI の詳細については、「IRI とプレフィックス」（13 ページ）を参照してください。

あるトリプルの目的語は別のトリプルの主語になることが多いため、トリプルのコレクションはグラフを形成します。このドキュメントでは、次の規則を使用してグラフを表現します。

- 主語と目的語は、楕円で表す。
- 述語は、エッジ（ラベル付き矢印）で表す。
- 型付きリテラルは、四角形で表す。

1.3.3 空白ノードの識別子

MarkLogic では、空白ノードには空白ノードの識別子が割り当てられます。この内部識別子は、複数の呼び出しの間で維持されます。トリプルでは、空白ノードは主語または目的語に対して使用でき、アンダースコア（_）で指定します。例えば以下のようになります。

```
_:jane <http://xmlns.com/foaf/0.1/name> "Jane Doe".
<http://example.org/people/about> <http://xmlns.com/foaf/
  0.1/knows>
_:jane
```

2 つの空白ノードがある場合、同一のものかどうかを判断できます。1 つ目のノード「`_:jane`」は、やはり「`_:jane`」に言及している 2 つ目の呼び出しと同じノードを参照します。空白ノードは スコーレム化された IRI として表現されます。これは、実存する変数が一意の定数で置き換えられる空白ノードです。各空白ノードのプレフィックスは「`http://marklogic.com/semantics/blank`」です。

1.3.4 RDF データ型

RDF では、XML スキーマデータ型を使用します。例えば `xs:string`、`xs:float`, `xs:double`、`xs:integer`、`xs:date` などが挙げられます。これについては仕様書である『*XML Schema Part 2: Datatypes Second Edition*』を参照してください。

http://www.w3.org/TR/xmlschema-2

XML スキーマのすべての単純型と単純型から生成されるすべての型がサポートされています（ただし `xs:QName` および `xs:NOTATION` を除く）。

RDF には、IRI によって命名されたカスタムのデータ型を含めることができます。例えば、サポートされる MarkLogic 固有のデータ型としては `cts:point` が挙げられます。

注： サポートされていないデータ型（`xs:QName`、`xs:NOTATION`、またはこれらから派生した型など）を使用すると、XDMP-BADRDFVAL 例外が生成されます。

データ型の宣言を省略すると、`xs:string` 型であるとみなされます。型付きリテラルを表すには、`datatype` 属性を指定するか、または `xml:lang` 属性を使用してリテラルの言語エンコーディングを指定します。例えば、「en」は英語を表します。

MarkLogic のセマンティックデータモデルのデータ型では、スキーマのないデータ型を持つ値を使用できます。このような値は、`xs:untypedAtomic` として識別されます。

1.3.5 IRI とプレフィックス

このセクションでは、IRI とプレフィックスの意味やロールについて説明します。このセクションは、次の概念で構成されています。

- IRI
- プレフィックス

1.3.5.1 IRI

IRI（Internationalized Resource Identifier）は、URI（Uniform Resource Identifier）の国際化バージョンです。URI は ASCII 文字のサブセットを使用し、このセットに限定されています。IRI は ASCII 以外の文字も使用しているため、国際的なコンテキストではより有用なものとなっています。IRI（および URI）は、リソースをフェッチできるようにする一意のリソース識別子です。また URN（Uniform Resource Name）も、リソースを一意に識別するために使用できます。

IRI は URL に似ている場合もあり、実際の Web サイトであることもそうではないこともあります。例えば以下のようになります。

```
<http://example.org/addressbook/d>
```

IRI は、リソースを識別するために URI の代わりに適宜使用されます。SPARQL では特に IRI を参照するため、このガイドの後の章では、URI ではなく IRI を参照します。

IRI はファクトのあいまいさを排除するために必須であり、特にデータがさまざまなデータソースに由来する場合に必要です。例えば、さまざまなソースから書籍に関する情報を受け取る場合、ある出版社は書籍名を「title」（タイトル）として記述し、別の出版社は著者の地位を「title」（肩書）として記述している可能性があります。同様に、あるドメインでは書籍の執筆者を「author」（著者）として記述し、別のドメインでは「creator」（作成者）として記述している可能性があります。

IRI（および URN）を使用して情報を提示することで、ファクトの意味をより明確に表現できます。次の例は、3 つの N-Triples を示したものです。

```
<http://example.org/people/title/sh1999>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#label>
"Lucasian Professor of Mathematics"

<urn:isbn:9780553380163>
<http://purl.org/dc/elements/1.1/title>
"A Brief History of Time"

<urn:isbn:9780553380163>
<http://purl.org/dc/elements/1.1/creator>
"Stephen Hawking"
```

注：書式を整えるために改行が挿入されています。そのため、この RDF の N-Triples シンタックスは無効です。通常、各トリプルは 1 行にします（Turtle のシンタックスでは、1 つのトリプルを複数の行に折り返すことが許可されています）。

IRI は RDF の重要なコンポーネントですが、通常は長くなるため管理が困難です。
IRI を短縮するメカニズムとして、CURIE（短縮 URI 表現）がサポートされています。
CURIE は、「CURIE Syntax Definition」で指定されています。

http://www.w3.org/TR/rdfa-syntax/#s_curies

1.3.5.2 プレフィックス

プレフィックスは IRI によって識別され、多くの場合は組織や会社の名前で始まります。例えば以下のようになります。

```
PREFIX js: <http://example.org/people/about/js>
```

プレフィックスは名前を識別するために使用する省略用の文字列です。指定されたプレフィックスにより、プレフィックス IRI が指定した文字列にバインドされます。その後 IRI を参照するときは、完全な IRI を記述する代わりにプレフィックスを使用できます。プレフィックスを使用して RDF を記述するときは、プレフィックスの後にコロンを続けます。定義するリソースには任意のプレフィックスを選択できます。例えば、これは SPARQL 宣言の 1 つです。

```
PREFIX dir: <http://example.org/people/about>
```

また、標準のプレフィックスや仕様の一部として合意の成されているプレフィックスを使用することもできます。RDF に対する SPARQL 宣言は次のとおりです。

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
```

プレフィックスは、使用するシリアライゼーションに依存します。Turtle のプレフィックス宣言は次のようになります。

```
@prefix dir: <http://example.org/people/about> .
```

1.3.6 RDF 語彙

RDF 語彙は、タームを分類するための標準的なシリアライゼーションを提供するために RDFS（RDF スキーマ）または OWL（Web オントロジー言語）を使用して定義されています。基本的に語彙とは、RDF グラフを形成するアークの IRI の集まりです。
例えば、FOAF 語彙は、人々ならびに関係性を記述したものです。

共有されている標準的な語彙が存在すると非常に便利ですが、語彙は結合したり新しく作成したりできるため、そのような語彙は必須ではありません。使用する語彙は、次のプレフィックス参照を使用して判断できます。

http://prefix.cc/about

語彙は膨大な数が存在し、増え続けています。広く使用され合意が成されている共通の RDF プレフィックスとしては、次のものが挙げられます。

プレフィックス	プレフィックス IRI	
cc	http://web.resource.org/cc#ns	クリエイティブ・コモンズ
dc	http://purl.org/dc/elements/1.1/	ダブリンコア語彙
dcterms	http://purl.org/dc/terms	ダブリンコアターム
rdfs	http://www.w3.org/2000/01/rdf-schema#	RDF スキーマ
rdf	http://www.w3.org/1999/02/22-rdf-syntax-ns#	RDF 語彙
owl	http://www.w3.org/2002/07/owl#	Web オントロジー言語
foaf	http://xmlns.com/foaf/0.1/	FOAF（Friend of a Friend）
skos	http://www.w3.org/2004/02/skos/core	SKOS（Simple Knowledge Organization System）
vcard	http://www.w3.org/2001/vcard-rdf/3.0	VCard 語彙
void	http://rdfs.org/ns/void	Vocabulary of Interlinked Datasets
xml	http://www.w3.org/XML/1998/namespace	XML 名前空間
xhtml	http://www.w3.org/1999/xhtml	XHTML 名前空間
xs	http://www.w3.org/2001/XMLSchema#	XML スキーマ
fn	http://www.w3.org/2005/xpath-functions	XQuery 関数および演算子

注： これらの語彙では、IRI は URL でもあります。

1.4 データセットの例

政府および政府機関、医療機関、金融機関、ソーシャルメディアなどのドメインから生まれるデータは増え続けています。このようなデータはトリプルとして使用可能であり、多くの場合は次の目的で SPARQL 経由でアクセスできます。

- セマンティック検索
- ダイナミックセマンティックパブリッシング
- 多様なデータセットの集約

一般の人々が使用できるデータセットは、多数存在します。

例えば以下のようになります。

- FOAF：http://www.foaf-project.org：人々、人々の行動、および他の人々やエンティティとの関係性について記述する標準的な RDF 語彙を提供するためのプロジェクト。
- DBPedia：http://dbpedia.org/Datasets：RDF データセットへの多数の外部リンクを持つ、Wikipedia から生成されたデータ。
- セマンティック Web：http://data.semanticweb.org：カンファレンスデータに関する数千もの一意のトリプルのデータベース。

2.0 MarkLogic でセマンティックを使ってみる

この章は、次のセクションで構成されています。

2.1 MarkLogic サーバーの設定

MarkLogic サーバーをインストールすると、データベース、REST インスタンス、および XDBC サーバー（コンテンツのロード用）が自動的に作成されます。デフォルトの Documents データベースは、MarkLogic サーバーをインストールして REST インスタンスをアタッチすると、その直後からポート 8000 で使用できるようになります。

このガイドの例では、この事前設定されたデータベース、XDBC サーバー、およびポート 8000 の REST API インスタンスを使用します。このセクションでは、トリプルを格納するための MarkLogic サーバーの設定を取り上げます。それには、トリプルを格納、検索、および管理するようにデータベースを設定する必要があります。

> **注：** このセクションで説明する手順を完了するには、MarkLogic サーバーの管理者権限が必要です。

『*Installation Guide for All Platforms*』の説明に従って MarkLogic サーバーをデータベースサーバーにインストールしてから、次の作業を行います。

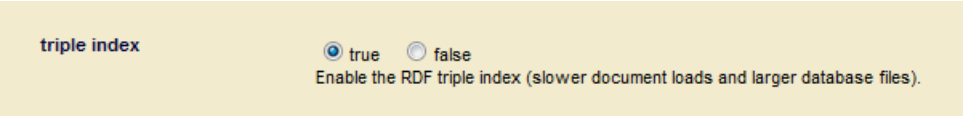
2.1.1 トリプルを使用するためのデータベースの設定

MarkLogic 9 以降では、Documents データベースのトリプルインデックスおよびコレクションレキシコンはデフォルトでオンになっています。また、これらのオプションは、作成する新しいデータベースでもすべて、デフォルトでオンになります。MarkLogic 9 の新しいインストールを使用する場合は、次の手順は省略して構いません。

既存のデータベースをトリプルに使用する場合は、トリプルインデックスおよびコレクションレキシコンを有効にする必要があります。次の手順は、データベースが正しくセットアップされているか確認する場合にも利用できます。

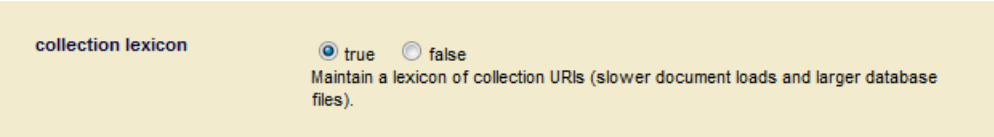
1. トリプル用にデータベースを設定するには、管理画面（`localhost:8001`）に移動します。Documents データベースをクリックし、[Configure] タブをクリックします。

2. 管理画面の［Configure］ページをスクロールダウンして、トリプルインデックスのステータスを確認します。



［true］に設定します（設定されていなかった場合）。トリプルインデックスがセマンティックに使用されるようになります。

3. さらに下にスクロールし、［collection lexicon］を［true］に設定します。



コレクションレキシコンインデックスは REST エンドポイントでは必須であり、必ず使用されます。名前付きグラフをクエリする場合は、コレクションレキシコンのみが必要です。

4. 完了したら［ok］をクリックします。

これで、トリプルをデフォルトデータベース（Documents データベース）にロードする前に必要な作業は完了です。

注： MarkLogic 9 以降の新しいインストールはすべて、トリプルインデックスおよびコレクションレキシコンがデフォルトでオンになります。また、新しいデータベースもすべて、トリプルインデックスおよびコレクションレキシコンがオンになります。

2.1.2 追加サーバーの設定

実稼動環境では、追加のアプリケーションサーバー、REST インスタンス、および XDBC サーバーを作成することになります。詳細については、次のリンクを参照してください。

- アプリケーションサーバー：追加のアプリケーションサーバーを作成するプロセスについては、『*Administrator's Guide*』の「Creating and Configuring App Servers」で説明しています。
- REST インスタンス：別のポートに別の REST インスタンスを作成するには、『*REST Application Developer's Guide*』の「Administering REST Client API Instances」を参照してください。
- XDBC サーバー：XDBC サーバーを作成するプロセスの詳細については、『*Administrator's Guide*』の「Creating a New XDBC Server」で説明しています。

2.2 トリプルのロード

このセクションでは、トリプルデータベースへのトリプルのロードについて取り上げます。次のトピックから構成されます。

- データセットのダウンロード
- mlcp を使用したトリプルのインポート
- インポートの検証

2.2.1 データセットのダウンロード

例では、DBPedia の Persondata（英語。Turtle シリアライゼーションされています）のサンプル全体を使用します。Persondata の別のサブセットを使用しても構いません。

1. DB Pedia から、次の Persondata サンプルデータセットをダウンロードします（downloads.dbpedia.org/3.8/en/persondata_en.ttl.bz2）。この章の残りの手順では、このデータセットを使用します。このデータセットは http://wiki.dbpedia.org/Downloads で入手できます。手動で選択する場合は、ページを下にスクロールダウンして Persondata を探し、「en」（英語）の「ttl」（Turtle）バージョン（`persondata_en.ttl.b2zip`）を選択します。

   注： DBPedia データセットは、クリエイティブ・コモンズの表示 - 継承ライセンスおよび GNU Free Documentation License の条件の下でライセンスされます。このデータは、ローカライズされた言語や N-Triples および N-Quad シリアライズ形式のものが用意されています。

2. 圧縮ファイルからローカルディレクトリにデータを抽出します。例えば `C:\space` にします。

2.2.2 mlcp を使用したトリプルのインポート

トリプルを MarkLogic にロードするには、mlcp（MarkLogic Content Pump）、REST エンドポイント、XQuery など複数の方法があります。トリプルの一括読み込みで推奨される方法は、mlcp です。ここで説明する各例では、スタンドアローンの Windows 環境で mlcp を使用します。

1. 『*mlcp User Guide*』の「Installation and Configuration」の説明に従って、MarkLogic Pump をインストールおよび設定します。

2. Windows コマンドインタプリタである `cmd.exe` で、mlcp インストール環境の mlcp の bin ディレクトリに移動します。例えば以下のようになります。

```
cd C:\space\marklogic-contentpump-1.3-1\bin
```

3. Persondata が C:\space にローカル保存されている場合は、プロンプトに次のコマンドを 1 行で入力します。

```
mlcp.bat import -host localhost -port 8000 -username admin ^
-password password -input_file_path c:\space\persondata_en.ttl ^
-mode local -input_file_type RDF -output_uri_prefix /people/
```

わかりやすくするため、この長いコマンドラインを Windows の行継続文字「^」で複数の行に分割しています。このコマンドを使用するときは、行継続文字を削除してください。

- UNIX 用に修正したコマンドは次のようになります。

```
mlcp.sh import -host localhost -port 8000 -username admin
-password\
password -input_file_path /space/persondata_en.ttl -mode
local\
-input_file_type RDF -output_uri_prefix /people/
```

わかりやすくするため、この長いコマンドラインを UNIX の行継続文字「\」で複数の行に分割しています。このコマンドを使用するときは、行継続文字を削除してください。

トリプルはデフォルトの Documents データベースにインポートおよび格納されます。

4. トリプルのデータインポートに成功すると、完了時の表示は次のようになります（UNIX の場合の出力）。

```
14/09/15 14:35:38 INFO contentpump.ContentPump: Hadoop
library version:2.0.0-alpha
14/09/15 14:35:38 INFO contentpump.LocalJobRunner: Content
type: XML
14/09/15 14:35:38 INFO input.FileInputFormat: Total input
paths to process : 1
O:file:///home/persondata_en.ttl : persondata_en.ttl
14/09/15 14:35:40 INFO contentpump.LocalJobRunner:
completed 0%
14/09/15 14:40:27 INFO contentpump.LocalJobRunner:
completed 100%
14/09/15 14:40:28 INFO contentpump.LocalJobRunner:
com.marklogic.contentpump.ContentPumpStats:
```

```
14/09/15 14:40:28 INFO contentpump.LocalJobRunner:
ATTEMPTED_INPUT_RECORD_COUNT: 59595
14/09/15 14:40:28 INFO contentpump.LocalJobRunner:
SKIPPED_INPUT_RECORD_COUNT: 0
14/09/15 14:40:28 INFO contentpump.LocalJobRunner: Total
execution time: 289 sec
```

2.2.3 インポートの検証

RDF トリプルがトリプルデータベースに正常にロードされたことを検証するには、次の手順を実行します。

1. Web ブラウザで、REST サーバーに移動します。

   `http://`*`hostname`*`:8000`

   「hostname」は MarkLogic サーバーのホストマシンの名前で、`8000` は MarkLogic サーバーをインストールしたときに作成された REST インスタンスのデフォルトのポート番号です。

2. Web アドレスの URL の末尾に「`/v1/graphs/things`」を付加します。

   例えば以下のようになります。

   ```
   http://hostname:8000/v1/graphs/things
   ```

   最初の 1000 件の主語が表示されます。

3. 主語のリンクをクリックすると、トリプルが表示されます。主語および目的語の IRI は、リンクとして表示されます。

8 triples

<http://dbpedia.org/resource/%C3%80lex_Corretja> <http://dbpedia.org/ontology/birthPlace> <http://dbpedia.org/resource/Barcelona> .
<http://dbpedia.org/resource/%C3%80lex_Corretja> <http://dbpedia.org/ontology/birthPlace> <http://dbpedia.org/resource/Spain> .
<http://dbpedia.org/resource/%C3%80lex_Corretja> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> .
<http://dbpedia.org/resource/%C3%80lex_Corretja> <http://xmlns.com/foaf/0.1/givenName> "Alex" .
<http://dbpedia.org/resource/%C3%80lex_Corretja> <http://xmlns.com/foaf/0.1/name> "Alex Corretja" .
<http://dbpedia.org/resource/%C3%80lex_Corretja> <http://xmlns.com/foaf/0.1/surname> "Corretja" .
<http://dbpedia.org/resource/%C3%80lex_Corretja> <http://purl.org/dc/elements/1.1/description> "Tennis player" .
<http://dbpedia.org/resource/%C3%80lex_Corretja> <http://dbpedia.org/ontology/birthDate> "1974-04-11"^^xs:date .

2.3 トリプルのクエリ

SPARQL クエリは、Query Console で実行するか、`/v1/graphs/sparql` エンドポイントを使用した HTTP エンドポイント経由（`GET:/v1/graphs/sparql` および `POST:/v1/graphs/sparql`）で実行できます。このセクションでは、次の内容を取り上げます。

- ネイティブ SPARQL によるクエリ
- sem:sparql 関数によるクエリ

注： このセクションでは、「データセットのダウンロード」（20 ページ）の説明に従ってサンプルデータセットをロード済みであることを前提にしています。

2.3.1 ネイティブ SPARQL によるクエリ

Query Console では、ネイティブ SPARQL またはビルトイン関数 `sem:sparql` を使用してクエリを実行できます。

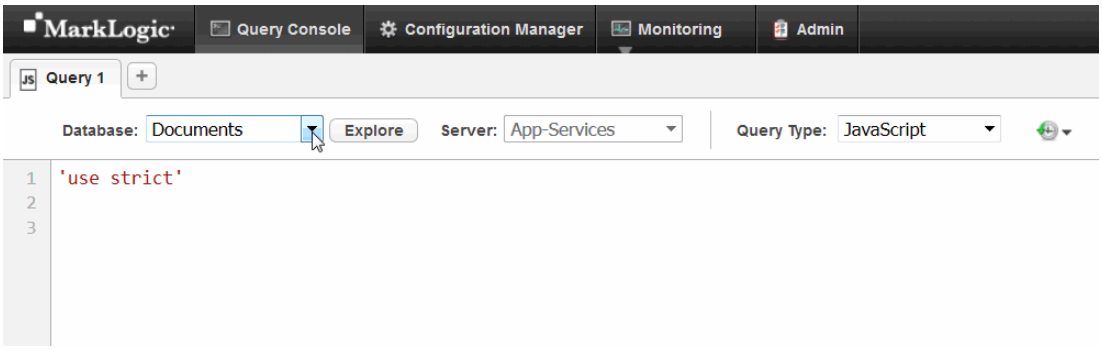
クエリを実行するには、次の手順に従います。

1. Web ブラウザで、Query Console に移動します。

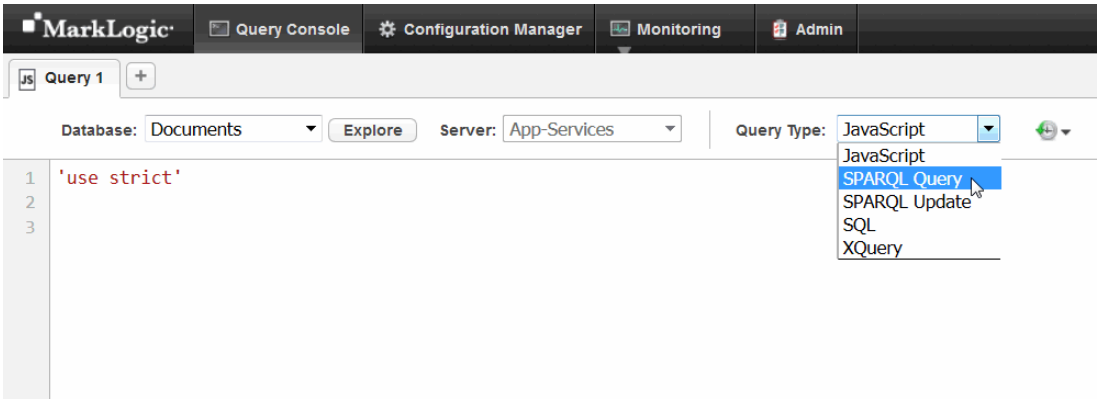
   `http://`***`hostname`***`:8000/qconsole`

   「hostname」は MarkLogic サーバーの名前です。

2. ［Database］ドロップダウンリストから、Documents データベースを選択します。



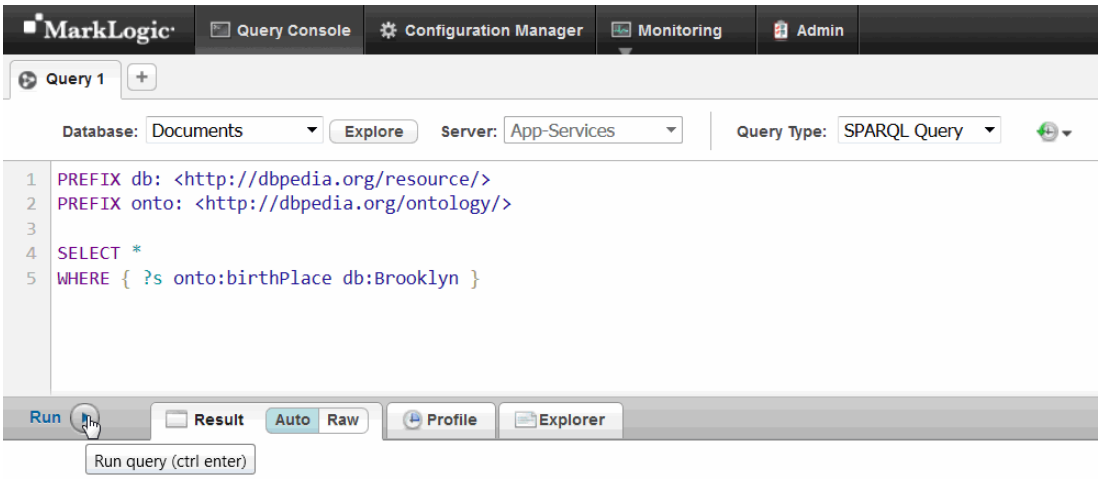
3. ［Query Type］ドロップダウンリストから、［SPARQL Query］を選択します。



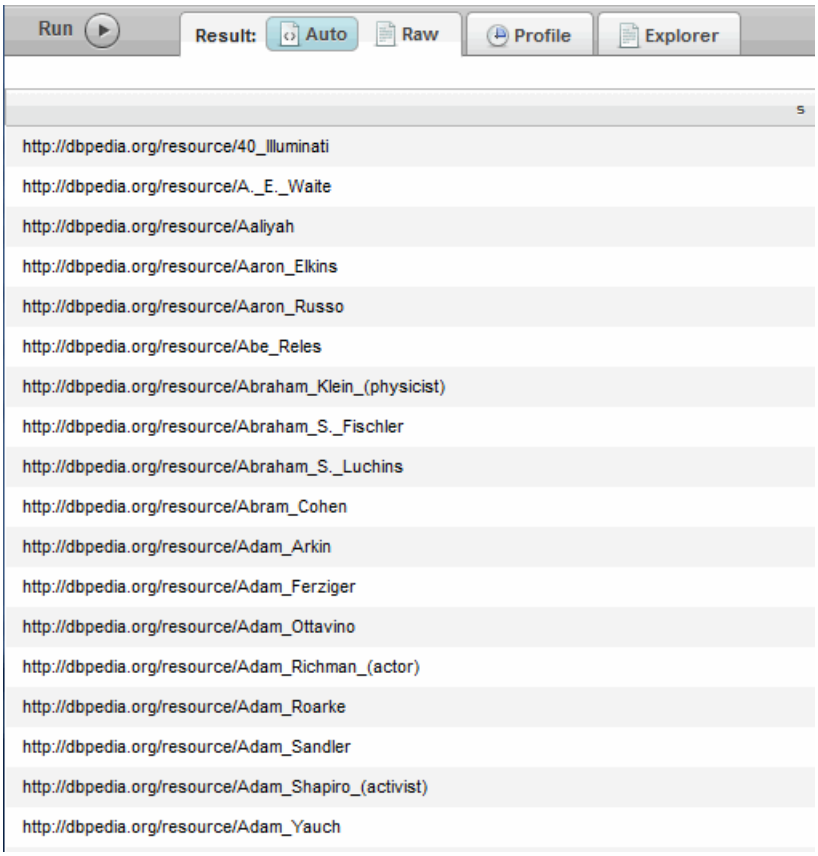
4. クエリウィンドウで、デフォルトのクエリテキストを以下の SPARQL クエリで置換します。

```
PREFIX db: <http://dbpedia.org/resource/>
PREFIX onto: <http://dbpedia.org/ontology/>

SELECT *
WHERE { ?s onto:birthPlace db:Brooklyn }
```

5. ［Run］をクリックします。



結果には、ブルックリン（Brooklyn）で誕生した人々が IRI として表示されます。



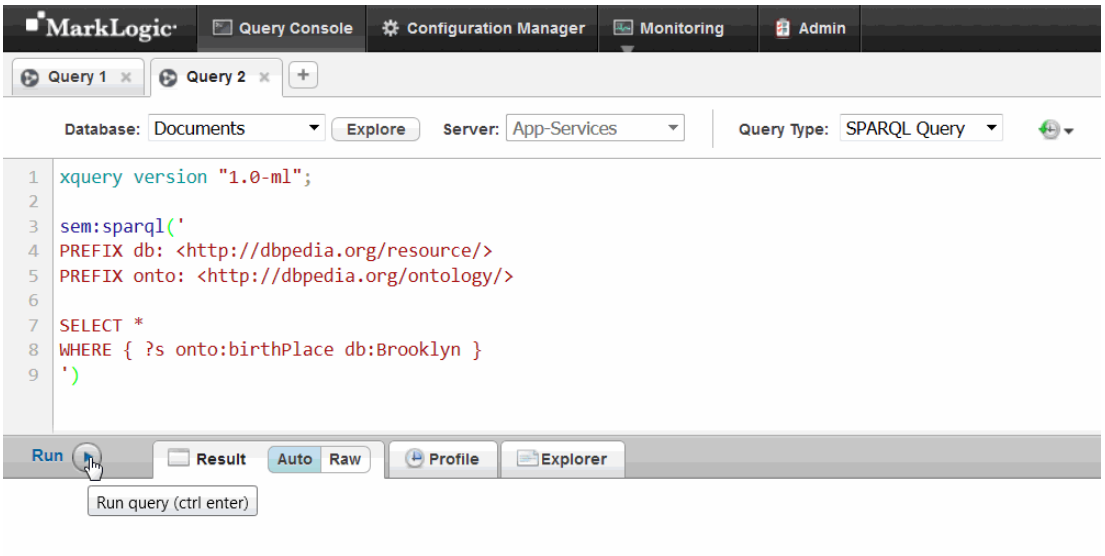
2.3.2 sem:sparql 関数によるクエリ

Query Console でビルトインの XQuery 関数 `sem:sparql` を使用して、同じクエリを実行します。

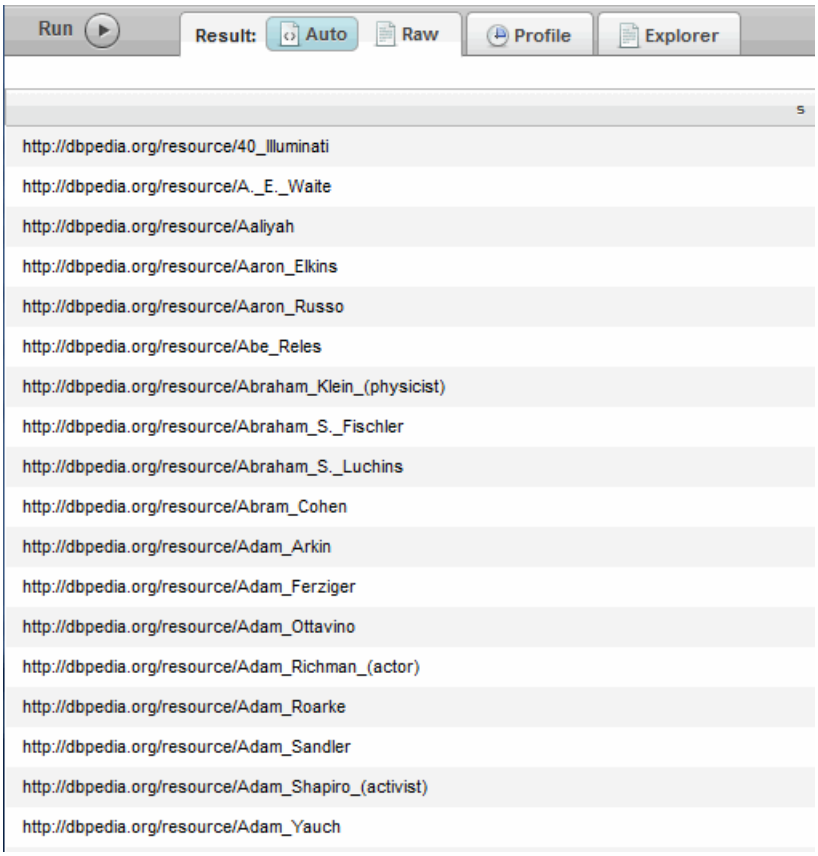
1. [Database] ドロップダウンリストから、Documents データベースを選択します。
2. [Query Type] ドロップダウンリストから、[XQuery] を選択します。
3. クエリウィンドウに、このクエリを入力します。

```
xquery version "1.0-ml";

sem:sparql('
PREFIX db: <http://dbpedia.org/resource/>
PREFIX onto: <http://dbpedia.org/ontology/>

SELECT *
WHERE { ?s onto:birthPlace db:Brooklyn }
')
```

4. [Run] をクリックします。



5. 結果には、「ネイティブ SPARQL によるクエリ」（24 ページ）の場合と同様に、ブルックリン（Brooklyn）で誕生した人々が IRI として表示されます。



SPARQL クエリの詳細および例については、「セマンティッククエリ」（71 ページ）を参照してください。

3.0 セマンティックトリプルの読み込み

MarkLogic データベースには、2 つのソースからトリプルをロードできます。埋め込みトリプル要素が含まれる XML ドキュメントからと、シリアライズされた RDF データが含まれるトリプルファイルからです。この章は、次のセクションで構成されています。

- 埋め込み RDF トリプルのロード
- トリプルのロード

トリプルは、SPARQL Update を使用してロードすることもできます。詳細については、「SPARQL Update」（172 ページ）を参照してください。

3.1 埋め込み RDF トリプルのロード

XML にトリプルが組み込まれたドキュメントは、『*Loading Content Into MarkLogic Server Guide*』の「Available Content Loading Interfaces」で説明するいずれかの読み込みツールを使用して読み込みます。

> **注：** 埋め込みトリプルは、`sem:triple` のスキーマで定義された MarkLogic XML 形式（`semantics.xsd`）にする必要があります。

MarkLogic データベースに読み込まれたトリプルは、SPARQL によるアクセスおよびクエリを目的として、トリプルインデックスによってインデックス付けされ、格納されます。詳細については、「MarkLogic での RDF トリプルの格納」（7 ページ）を参照してください。

3.2 トリプルのロード

サポートされている RDF シリアライゼーションでシリアライズされたトリプルが含まれるドキュメントは、複数の方法で MarkLogic に読み込むことができます。「サポートされている RDF トリプルの形式」（31 ページ）では、そのような RDF 形式について説明します。

1 つあるいは複数のトリプルグループをロードすると、パースされて XML ドキュメントが生成されます。XML ドキュメントごとに一意の IRI が生成されます。各ドキュメントには複数のトリプルを含めることができます。

> **注：** ドキュメント内に格納されるトリプル数の設定は MarkLogic サーバーによって定義されており、ユーザーは設定できません。

読み込まれるトリプルはトリプルインデックスでインデックス付けされて、SPARQL、XQuery、または両者の組み合わせでトリプルにアクセスしたりクエリしたりできるようになります。また、REST エンドポイントを使用して、SPARQL クエリを実行したり RDF データを返したりすることもできます。

トリプルのグラフを指定しなかった場合、トリプルはデフォルトグラフに格納されます。このデフォルトグラフは、collection と呼ばれる、MarkLogic サーバーの機能を使用します。MarkLogic サーバーは、コレクション IRI `http://marklogic.com/semantics#default-graph` を使用してデフォルトグラフをトラッキングします。

ロードプロセス中に別のコレクションを指定して、名前付きグラフにトリプルをロードできます。コレクションの詳細については、『*Search Developer's Guide*』の「コレクション」を参照してください。

> **注：** グラフ名を指定せずにトリプルをデータベースに挿入すると、そのトリプルはデフォルトグラフ `http://marklogic.com/semantics#default-graph` に挿入されます。トリプルをスーパーデータベースに挿入し、そのスーパーデータベースで `fn:count(fn:collection())` を実行すると、重複 URI に関する DUPURI 例外が発生します。

トリプルデータが含まれた XML ドキュメントが生成されると、`/triplestore` という名前のデフォルトディレクトリにロードされます。ロード用のツールによっては、別のディレクトリを指定できます。例えば、mlcp を使用してトリプルをロードする場合は、グラフとディレクトリをインポートオプションの一部として指定できます。詳細については、「mlcp を使用したトリプルのロード」（37 ページ）を参照してください。

このセクションでは、次の内容を取り上げます。

- サポートされている RDF トリプルの形式
- RDF 形式の例
- mlcp を使用したトリプルのロード
- XQuery を使用したトリプルのロード
- REST API を使用したトリプルのロード

3.2.1 サポートされている RDF トリプルの形式

MarkLogic サーバーでは、次に示す RDF データの形式を読み込むことができます。

形式	説明	ファイル形式	MIME タイプ
RDF/XML	RDF グラフを XML ドキュメントとしてシリアライズするために使用するシンタックス。例については、「RDF/XML」（32 ページ）を参照してください。	.rdf	application/rdf+xml
Turtle	Turtle（Terse RDF Triple Language）シリアライゼーションは、N3（Notation 3）の単純化されたサブセットであり、必要最小限のシリアライゼーションでデータを表現するために使用されます。例については、「Turtle」（32 ページ）を参照してください。	.ttl	text/turtle
RDF/JSON	RDF データを JSON オブジェクトとしてシリアライズするために使用するシンタックス。例については、「RDF/JSON」（33 ページ）を参照してください。	.json	application/rdf+json
N3	N3（Notation 3）シリアライゼーションは、RDF データをシリアライズするために使用する非 XML シンタックスです。例については、「N3」（34 ページ）を参照してください。	.n3	text/n3
N-Triples	RDF グラフのプレーンテキストシリアライゼーションです。N-Triples は、Turtle および N3（Notation 3）のサブセットです。例については、「N-Triples」（34 ページ）を参照してください。	.nt	application/n-triples
N-Quads	オプションのコンテキスト値で N-Triples を拡張するスーパーセットのシリアライゼーションです。例については、「N-Quads」（35 ページ）を参照してください。	.nq	application/n-quads
TriG	RDF 名前付きグラフおよび RDF データセットのプレーンテキストシリアライゼーションです。例については、「TriG」（36 ページ）を参照してください。	.trig	application/trig

3.2.2 RDF 形式の例

このセクションでは、次の RDF 形式の例について説明します。

- RDF/XML
- Turtle
- RDF/JSON
- N3
- N-Triples
- N-Quads
- TriG

3.2.2.1 RDF/XML

RDF/XML は、一意の RDF シンタックスを XML として記述するオリジナルの標準です。RDF グラフを XML ドキュメントとしてシリアライズするために使用します。

この例では、3 つのプレフィックス「rdf」、「xsd」、「d」を定義します。

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#"
  xmlns:d="http://example.org/data/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
  <rdf:Description rdf:about="http://example.org/
  data#item22">
    <d:shipped rdf:datatype="xsd:date">2013-05-14
    </d:shipped>
    <d:quantity rdf:datatype="xsd:integer">27</d:quantity>
    <d:invoiced rdf:datatype="xsd:boolean">true
    </d:invoiced>
    <d:costPerItem rdf:datatype="xsd:decimal">10.50
    </d:costPerItem>
  </rdf:Description>
</rdf:RDF>
```

3.2.2.2 Turtle

Turtle（Terse RDF Triple Language）シリアライゼーションは、SPARQL に類似のシンタックスを使用して RDF データモデル内のデータを表現します。Turtle シンタックスは、RDF データモデル内のトリプルを 3 つの IRI のグループで表現します。

例えば以下のようになります。

```
<http://example.org/item/item22>
<http://example.org/details/shipped>
"2013-05-14"^^<http://www.w3.org/2001/XMLSchema#dateTime> .
```

このトリプルは、アイテム 22 が 2013 年 5 月 14 日に出荷されたことを記述しています。

Turtle シンタックスでは、`@prefix` を使用して IRI の共通部分を取り除くことで、複数のステートメントに関する情報を短縮化できます。これにより、RDF の Turtle ステートメントは迅速に記述できます。シンタックスは RDF/XML に似ていますが、RDF/XML とは異なり、XML に依存しません。Turtle シンタックスは、有効な N3（Notation 3）でもあります。これは、Turtle が N3 のサブセットであるためです。

注： Turtle は、有効な RDF グラフだけをシリアライズできます。

この例では 4 つのトリプルで 1 つのトランザクションを記述しています。「shipped」オブジェクトには「date」データ型が割り当てられているため、引用符に囲まれた型付きリテラルになります。「quantity」、「invoiced」、および「costPerItem」オブジェクトに対応する 3 つの型なしリテラルが存在します。

```
@prefix i: <http://example.org/item> .
@prefix dt: <http://example.org/details#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
  i:item22 dt:shipped "2013-05-14"^^xsd:date .
  i:item22 dt:quantity 100 .
  i:item22 dt:invoiced true .
  i:item22 dt:costPerItem 10.50 .
```

3.2.2.3 RDF/JSON

RDF/JSON は RDF のテキストシンタックスであり、JSON（JavaScript Object Notation）と互換性のある形式で RDF グラフを記述できます。

例えば以下のようになります。

```
{ "http://example.com/directory#m":
  { "http://example.com/ns/person#firstName":
    [ { "value": "Michelle",
        "type": "literal",
        "datatype": "http://www.w3.org/2001/
        XMLSchema#string" }
    ]
  }
}
```

3.2.2.4 N3

N3（Notation 3）は非 XML シンタックスであり、XML RDF 表記よりもよりコンパクトでわかりやすい形式で RDF グラフをシリアライズする目的で使用します。N3 では RDF ベースのルールがサポートされます。

N3 では、同一の主語について複数のステートメントが存在する場合は、セミコロン（;）を使用して同じ主語に別のプロパティを導入できます。また、カンマを使用すると、同じ述語と主語に対して別の目的語を導入できます。

例えば以下のようになります。

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix foafcorp: <http://xmlns.com/foaf/corp/> .
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0> .
@prefix sec: <http://www.rdfabout.com/rdf/schema/ussec> .
@prefix id: <http://www.rdfabout.com/rdf/usgov/sec/id> .

id:cik0001265081 sec:hasRelation [
  dc:date "2008-06-05";
  sec:corporation id:cik0001000045;
  rdf:type sec:OfficerRelation;
  sec:officerTitle "Senior Vice President, CFO"] .
id:cik0001000180 sec:cik "0001000180";
  foaf:name "SANDISK CORP";
  sec:tradingSymbol "SNDK";
  rdf:type foafcorp:Company.
id:cik0001009165 sec:cik "0001009165";
  rdf:type foaf:Person;
  foaf:name "HARARI ELIYAHOU ET AL";
  vcard:ADR [ vcard:Street "601 MCCARTHY BLVD.; ";
vcard:Locality "MILPITAS, CA"; vcard:Pcode "95035" ] .
```

3.2.2.5 N-Triples

N-Triples は、RDF グラフのプレーンテキストシリアライゼーションです。Turtle のサブセットであり、Turtle や N3 よりもシンプルに使用できるように設計されています。N-Triples シンタックスの各行は、1 つの RDF トリプルステートメントをエンコーディングし、次のように構成されています。

- 主語（IRI または空白ノード識別子）。1 つあるいは複数のスペース文字が続きます。
- 述語（IRI）。1 つあるいは複数のスペース文字が続きます。

- 目的語（IRI、空白ノード識別子、またはリテラル）。ピリオド（.）と改行が続きます。

型付きリテラルには、言語を識別する言語タグを含めることができます。この N-Triples の例では、`@en-US` は、リソースのタイトルが米国英語であることを示します。

```
<http://www.w3.org/2001/sw/RDFCore/ntriples>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/0.1/Document> .
<http://www.w3.org/2001/sw/RDFCore/ntriples/>
<http://purl.org/dc/terms/title> "Example Doc"@en-US .
<http://www.w3.org/2001/sw/RDFCore/ntriples/>
<http://xmlns.com/foaf/0.1/maker> _:jane .
<http://www.w3.org/2001/sw/RDFCore/ntriples/>
<http://xmlns.com/foaf/0.1/maker> _:joe .
  _:jane <http://www.w3.org/1999/02/22-rdf-syntax-ns>
  <http://xmlns.com/foaf/0.1/Person> .
  _:jane <http://xmlns.com/foaf/0.1/name> "Jane Doe".

  _:joe <http://www.w3.org/1999/02/22-rdf-syntax-ns>
  <http://xmlns.com/foaf/0.1/Person> .
  _:joe <http://xmlns.com/foaf/0.1/name> "Joe Bloggs".
```

注： 各行は、末尾のピリオドの後で改行されます。わかりやすくするため、さらに改行が加えられています。

3.2.2.6 N-Quads

N-Quads は、RDF データセットのエンコーディングを目的とした、行ベースのプレーンテキストシリアライゼーションです。N-Quads シンタックスは N-Triples のスーパーセットであり、オプションのコンテキスト値で N-Triples を拡張します。最もシンプルなステートメントは、RDF トリプルを形成するターム（主語、述語、目的語）と、トリプルが属するデータセット内のグラフにラベルを付けるオプションの IRI で構成されたシーケンスです。これらすべてはスペースで区切られ、各ステートメントの末尾にあるピリオド（.）で終了します。

この例では、関係性の語彙を使用します。語彙のクラスまたはプロパティには、語彙の IRI に「acquaintanceOf」というターム名を付加して構築される IRI があります。

```
<http://example.org/#Jane>
<http://http://purl.org/vocab.org/relationship/#acquaintan
ceOf>
<http://example.org/#Joe>
<http://example.org/graphs/directory> .
```

3.2.2.7 TriG

TriG は、RDF グラフをシリアライズするためのプレーンテキストシリアライゼーションです。TriG は Turtle に似ていますが、波括弧（{ と }）を使用して拡張されています。そのため、トリプルを複数のグラフにグループ化し、名前付きグラフの前にその名前を付けることができます。オプションの等号演算子（=）はグラフ名を割り当てるために使用できます。また、Notation 3 との互換性を確保するため、オプションで末尾のピリオド（.）を含めることができます。

TriG シリアライゼーションには次の特徴があります。

- グラフ名は TriG ドキュメント内で一意である必要があります。名前なしグラフは、TriG ドキュメントごとに 1 つです。
- TriG のコンテンツは、「.trig」サフィックスの付いたファイルに格納されます。また、TriG の MIME タイプは application/trig で、コンテンツのエンコーディングは UTF-8 です。

この例では、デフォルトグラフと 2 つの名前付きグラフが使用されています。

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

# default graph is http://marklogic.com/semantics#default-
graph
    {
    <http://example.org/joe> dc:publisher "Joe" .
    <http://example.org/jane> dc:publisher "Jane" .
    }
# first named graph
    <http://example.org/joe>
    {
      _:a foaf:name "Joe" .
      _:a foaf:mbox <mailto:joe@jbloggs.example.org> .
    }
# second named graph
  <http://example.org/jane>
    {
      _:a foaf:name "Jane" .
      _:a foaf:mbox <mailto:jane@jdoe.example.org> .
    }
```

3.2.3 mlcp を使用したトリプルのロード

mlcp（MarkLogic Content Pump）は、コンテンツをローカルのファイルシステムまたは HDFS（Hadoop 分散ファイルシステム）から MarkLogic に対してインポート、エクスポート、コピーを実行するコマンドラインツールです。

mlcp を使用すると、数十億にも及ぶトリプルおよびクアッドを MarkLogic データベースに一括読み込みしたり、インポート用のオプションを指定したりできます。例えば、トリプルまたはクアッドのロード先ディレクトリを指定できます。mlcp は、トリプルを一括読み込みする場合の推奨ツールです。mlcp の詳細については、『*Loading Content Into MarkLogic Server Guide*』の「Loading Content Using MarkLogic Content Pump」を参照してください。

このセクションでは、mlcp を使用して MarkLogic サーバーにトリプルをロードする方法について説明します。このセクションは、次のトピックから構成されています。

- 準備
- import コマンドのシンタックス
- トリプルおよびクアッドのロード
- コレクションおよびディレクトリの指定

3.2.3.1 準備

mlcp を使用してコンテンツをロードするには、次の手順を使用します。

1. mlcp バイナリファイルを developer.marklogic.com からダウンロードして抽出します。最新バージョンの mlcp であることを確認してください。mlcp のインストールと使用およびシステム要件の詳細については、『*mlcp User Guide*』の「Installation and Configuration」を参照してください。

   注： 抽出された mlcp バイナリファイルは同じ MarkLogic ホストマシン上に配置されている必要はありませんが、トリプルのロード先となるホストマシンに対するアクセス権やパーミッションが必要になります。

2. ここで取り上げる例では、デフォルトのデータベース（Documents）およびフォレスト（Documents）を使用します。独自のデータベースを作成するには、『*Administrator's Guide*』の「Creating a New Database」を参照してください。

3. トリプルインデックスがオンになっていることを確認します。これを行うには、管理画面の Documents データベース設定ページをチェックするか、Admin API を使用します。詳細については、「トリプルインデックスの有効化」（58 ページ）を参照してください。

注： REST API インスタンスで使用されるグラフストア HTTP プロトコルを使用したり、SPARQL クエリで GRAPH '?g' コンストラクタを使用したりするには、コレクションレキシコンインデックスが必要です。コレクションレキシコンについては、「トリプルを使用するためのデータベースの設定」（18 ページ）を参照してください。

4. mlcp はデフォルトサーバーのポート 8000 で使用できます。mlcp には XDBC サーバーが含まれています。独自の XDBC サーバーを作成するには、『*Administrator's Guide*』の「Creating a New XDBC Server」を参照してください。

5. （オプション）mlcp の bin ディレクトリにパスを通しておきます。例えば以下のようになります。

```
$ export PATH=${PATH}:/space/marklogic/directory-name/bin
```

「directory-name」は、ダウンロードした mlcp のバージョンから生成されます。

6. コマンドラインインタプリタまたはインターフェイスを使用して、import コマンドを 1 行のコマンドとして入力します。

3.2.3.2 import コマンドのシンタックス

トリプルやクアッドを MarkLogic にロードするために必要な mlcp import コマンドのシンタックスは次に示すとおりです。

```
mlcp_command import -host hostname -port port number \
-username username -password password \
-input_file_path filepath -input_file_type filetype
```

注： このセクションでは、行継続文字「\」または「^」を使用して長いコマンドラインを複数の行に分割しています。import コマンドを使用するときは、行継続文字を削除してください。

使用する *mlcp_command* は、環境によって異なります。UNIX システムでは mlcp シェルスクリプト `mclp.sh`、Windows システムではバッチスクリプト `mlcp.bat` を使用します。`-host` および `-port` の値には、トリプルのロード先になる MarkLogic ホストマシンを指定します。また、ユーザー資格情報 `-username` および `-password` の後には、コンテンツへのパス `-input_file_path` が続きます。独自のデータベースを使用する場合は、データベースの `-database` パラメータを追加してください。データベースパラメータを指定しない場合、コンテンツはデフォルトの Documents データベースに配置されます。

-input_file_path は、ディレクトリ、ファイル、または圧縮ファイル（.zip または .gzip 形式）をポイントします。-input_file_type は、ロードするコンテンツのタイプです。トリプルの場合、-input_file_type は RDF にする必要があります。

注： -input_file_path で見つかったファイルのファイル拡張子は、ロードするコンテンツのタイプを識別するために mlcp によって使用されます。RDF シリアライゼーションのタイプは、ファイル拡張子（.rdf、.ttl、.nt など）によって判別されます。

ファイル拡張子が .nq または .trig であるドキュメントは、クアッドデータであると識別されます。その他のファイル拡張子は、すべてトリプルデータとして識別されます。ファイル拡張子の詳細については、「サポートされている RDF トリプルの形式」（31 ページ）を参照してください。

注： 指定したホストにインポートするには、十分な MarkLogic 権限が必要です。『*mlcp User Guide*』の「Security Considerations」を参照してください。

3.2.3.3 トリプルおよびクアッドのロード

必須のインポートオプションだけでなく、複数の入力および出力オプションを指定できます。そのようなオプションの詳細については、「インポートオプション」（40 ページ）を参照してください。例えば、-input_file_type オプションとして RDF を指定すると、トリプルおよびクアッドをロードできます。

```
$ mlcp.sh import -host localhost -port 8000 -username
user \
-password passwd -input_file_path
/space/tripledata/example.nt \
-mode local -input_file_type RDF
```

この例では、シェルスクリプトを使用して、単一の N-Triples ファイル example.nt からトリプルをロードします。ローカルのファイルシステムのディレクトリ /space/tripledata からポート 8000 上の MarkLogic ホストにインポートしています。

Windows 環境では、次のようなコマンドになります。

```
> mlcp.bat import -host localhost -port 8000 ^
-username admin -password passwd ^
-input_file_path c:\space\tripledata\example.nt -mode
local^
-input_file_type RDF
```

注： わかりやすくするため、長いコマンドラインは行継続文字「\」または「^」を使用して複数の行に分割しています。import コマンドを使用するときは、行継続文字を削除してください。

`-input_file_type` として RDF を指定すると、mlcp RDFReader はトリプルをパースして XML ドキュメントを生成します。ドキュメントのルート要素は `sem:triple` です。

3.2.3.4 インポートオプション

トリプルまたはクアッドをロードするときは、`import` コマンドで次のオプションを使用できます。

オプション	説明
`-input_file_type` *string*	入力ファイルのタイプを指定します。デフォルトは document です。トリプルの場合は、RDF を使用します。
`-input_compressed` *boolean*	「true」に設定すると、インポート時に圧縮ファイルを展開します。デフォルトは false です。
`-fastload` *boolean*	「true」に設定すると、フォレストを直接更新して、パフォーマンスの最適化を強制的に実行します。その結果、ドキュメント IRI が重複することがあります。『*mlcp User Guide*』の「Time vs. Correctness: Understanding -fastload Tradeoffs」を参照してください。
`-output_directory`	ロードしたドキュメントを作成する宛先データベースディレクトリを指定します。このオプションを使用すると、`-fastload` がデフォルトでオンになります。そのため、重複した IRI が作成されることがあります。『*mlcp User Guide*』の「Time vs. Correctness: Understanding -fastload Tradeoffs」を参照してください。デフォルト：`/triplestore`
`-output_graph`	データでの明示的なグラフ指定がないクアッドに割り当てるグラフ値です。`-output_override_graph` との併用はできません。
`-output_override_graph`	データでクアッドが指定されている場合も、そうでない場合も、すべてのクアッドに割り当てるグラフ値です。`-output_graph` との併用はできません。

オプション	説明
-output_collections	コレクションのカンマ区切りのリストを作成します。デフォルト：http://marklogic.com/semantics#default-graph -output_collections を -output_graph や -output_override_graph と併用すると、指定したコレクションが、読み込まれるドキュメントに追加されます。
-database string	（オプション）宛先データベースの名前。デフォルト：-host および -port で指定された宛先のアプリケーションサーバーに関連付けられているデータベース。

注： mlcp を使用してトリプルを読み込んだ場合、-output_permissions オプションは無視されます。トリプル（およびその背後にあるトリプルドキュメント）は、読み込み先で設定されているグラフのパーミッションを継承します。

-output_collections と -output_override_graph を同時に設定した場合、-output_override_graph で指定したグラフについてグラフドキュメントが作成され、-output_collections と -output_override_graph で指定したコレクションにトリプルドキュメントが読み込まれます。

-output_collections と -output_graph を同時に設定した場合、-output_graph で指定したグラフについてグラフドキュメントが作成されます（そのデータには、明示的なグラフ指定はありません）。データに明示的なグラフ指定がないクアッドは、-output_collections で指定したコレクションおよび -output_graph で指定したグラフに読み込まれます。一方、明示的なグラフデータを含んでいるクアッドは -output_collections で指定したコレクション、および指定したグラフに読み込まれます。

大規模なトリプルドキュメントを小さなドキュメントに分割して、mlcp を使用したロードを並列化したり、-input_file_path で指定するディレクトリ内のすべてのファイルをロードしたりできます。

mlcp のインポートオプションおよび出力オプションの詳細については、『*mlcp User Guide*』の「Import Command Line Options」を参照してください。

例えば以下のようになります。

```
# Windows users, see Modifying the Example Commands for Windows
```

```
$ mlcp.sh import -host localhost -port 8000 -username user \
-password passwd -input_file_path /space/tripledata \
-mode local -input_file_type RDF
```

3.2.3.5 コレクションおよびディレクトリの指定

トリプルを名前付きグラフにロードするには、-output_collections オプションを使用してコレクションを指定します。

注： 新しいグラフを作成するには、sparql-update-user ロールが必要です。ロールの詳細については、『*Security Guide*』の「Understanding Roles」を参照してください。

例えば以下のようになります。

```
# Windows users, see Modifying the Example Commands for Windows

$ mlcp.sh import -host localhost -port 8000 -username user \
-password passwd -input_file_path /space/tripledata \
-mode local -input_file_type RDF \
-output_collections /my/collection
```

このコマンドは、tripledata ディレクトリ内にあるすべてのトリプルを名前付きグラフに配置し、グラフの IRI を /my/collection に上書きします。

注： デフォルトグラフの IRI を上書きするときは、-filename_as_collection ではなく -output_collections を使用してください。

トリプルデータでは、コレクションを指定しない場合、ドキュメントはデフォルトコレクション（http://marklogic.com/semantics#default-graph）に配置されます。

クアッドデータでは、コレクションを指定しない場合、トリプルがパースおよびシリアライズされて、ドキュメントに格納され、クアッドの 4 番目の部分がコレクションになります。

例えばこのクアッドの 4 番目の部分は、主語のホームページを識別する IRI です。

```
<http://dbpedia.org/resource/London_Heathrow_Airport>
<http://xmlns.com/foaf/0.1/homepage>
<http://www.heathrowairport.com/>
<http://en.wikipedia.org/wiki/London_Heathrow_Airport?oldi
d=495283228#
absolute-line=26/> .
```

このクアッドがデータベースにロードされると、コレクションが名前付きグラフ `http://en.wikipedia.org/wiki/London_Heathrow_Airport?oldid=495283228#absolute-line=26` として生成されます。

> **注：** `-output_collections` インポートオプションで名前付きグラフを指定している場合、クアッドの 4 番目の要素は無視され、その名前付きグラフが使用されます。

複数のロード手法を使用している場合は、すべてのトリプルドキュメントを共通のディレクトリに配置することを検討してください。`sem:rdf-insert` および `sem:rdf-load` 関数は `/triplestore` ディレクトリにトリプルドキュメントを配置するため、`-output_uri_prefix /triplestore` を使用して mlcp が生成するトリプルドキュメントもそこに配置します。

例えば以下のようになります。

```
$ mlcp.sh import -host localhost -port 8000 -username user \
-password passwd -input_file_path /space/tripledata/
example.zip \
-mode local -input_file_type RDF -input_compressed true
-output_collections /my/collection -output_uri_prefix
'/triplestore'
```

トリプルまたはクアッドを圧縮された `.zip` または `.gzip` ファイルから指定した名前付きグラフにロードすると、mlcp によってコンテンツが抽出され、シリアライゼーションに基づいてシリアライズされます。例えば、Turtle ドキュメント（`.ttl`）が含まれる圧縮ファイルは、トリプルとして識別されパースされます。

mlcp を使用して MarkLogic にコンテンツをロードすると、一意の IRI を持つ XML ドキュメントとして読み込まれるときにトリプルがパースされます。このときの一意の IRI は、16 進数で表現されたランダムな番号です。この例では、mlcp を使用してトリプルが `persondata.ttl` ファイルからロードされ、`-output_uri_prefix` には `/triplestore` が指定されています。

```
/triplestore/d2a0b25bda81bb58-0-10024.xml
/triplestore/d2a0b25bda81bb58-0-12280.xml
/triplestore/d2a0b25bda81bb58-0-13724.xml
/triplestore/d2a0b25bda81bb58-0-14456.xml
```

トリプルをロードするために選択する手法は慎重に検討してください。mlcp によってドキュメント IRI を生成するアルゴリズムは、`sem:rdf-load` でシステムファイルディレクトリからロードする手法など、その他のロード手法の場合と異なります。

例えば、同じ persondata.ttl ファイルを sem:rdf-load でロードしても、お互いに無関係な IRI になります。

```
/triplestore/11b53cf4db02080a.xml
/triplestore/19b3a986fcd71a5c.xml
/triplestore/215710576ebe4328.xml
/triplestore/25ec5ded9bfdb7c2.xml
```

sem:rdf-load でトリプルをロードすると、結果として得られるドキュメントでは、トリプルが http://marklogic.com/semantics プレフィックスにバインドされます。

例えば以下のようになります。

```
<?xml version="1.0" encoding="UTF-8"?>
<sem:triples xmlns:sem="http://marklogic.com/semantics">
  <sem:triple>
    <sem:subject>http://dbpedia.org/resource/Wayne_Stenehjem
    </sem:subject>
    <sem:predicate>http://purl.org/dc/elements/1.1/
     description</sem:predicate>
    <sem:object datatype=
    "http://www.w3.org/2001/XMLSchema#string"
    xml:lang="en">American politician
    </sem:object>
  </sem:triple>
  <sem:triple>
    <sem:subject>http://dbpedia.org/resource/Wayne_Stenehjem
    </sem:subject>
    <sem:predicate>http://dbpedia.org/ontology/birthDate
    </sem:predicate>
    <sem:object datatype=
    "http://www.w3.org/2001/XMLSchema#date">
    1953-02-05
    </sem:object>
    </sem:triple>
  </sem:triples>
```

注： sem:triples タグは省略して構いませんが、sem:triple タグは省略できません。

3.2.4 XQuery を使用したトリプルのロード

通常、トリプルは MarkLogic サーバーの外部で作成され、Query Console で次の sem: 関数を使用してロードされます。

- sem:rdf-insert
- sem:rdf-load
- sem:rdf-get

`sem:rdf-insert` および `sem:rdf-load` 関数は更新用の関数です。`sem:rdf-get` 関数は値を返し、トリプルをインメモリにロードします。これらの関数は、XQuery ライブラリモジュールとして実装されている XQuery セマンティック API に含まれています。

XQuery で `sem:` 関数を使用するには、Query Console で次の XQuery プロローグステートメントを使用してモジュールをインポートします。

```
import module namespace sem = "http://marklogic.com/
  semantics" at "/MarkLogic/semantics.xqy";
```

XQuery のセマンティック関数の詳細については、『*MarkLogic XQuery and XSLT Function Reference*』でセマンティック（`sem:`）のドキュメントを参照してください。

3.2.4.1 sem:rdf-insert

`sem:rdf-insert` 関数は、トリプルをトリプルドキュメントとしてデータベースに挿入します。トリプルは、`sem:triple` および `sem:iri` コンストラクタを使用してインメモリで作成されます。挿入されるドキュメントの IRI は、実行時に返されます。

例えば以下のようになります。

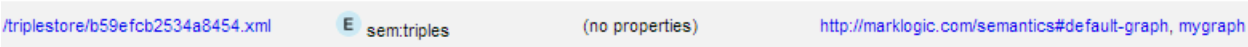
```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/
  semantics" at "/MarkLogic/semantics.xqy";

  sem:rdf-insert(
    sem:triple(
      sem:iri("http://example.org/people#m"),
      sem:iri("http://example.com/person#firstName"),
      "Michael"))

=>
(: Returns the document IRI :)
/triplestore/70eb0b7139816fe3.xml
```

デフォルトでは、`sem:rdf-insert` はドキュメントをディレクトリ `/triplestore/` に配置し、デフォルトグラフを割り当てます。名前付きグラフは、4 番目のパラメータでコレクションとして指定できます。

例えば以下のようになります。

```
xquery version "1.0-ml";import module namespace sem =
  "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

  sem:rdf-insert(sem:triple(
        sem:iri("http://example.com/ns/directory#jp"),
        sem:iri("http://example.com/ns/person#firstName"),
        "John-Paul"), (), (),  "mygraph")
```

この例を実行すると、ドキュメントがデフォルトグラフと mygraph の両方に挿入されます。



注： TriG シリアライゼーションでクアッドまたはトリプルを挿入する場合、グラフ名は quads/trig ファイルの「4 番目の位置」にある値に基づきます。

3.2.4.2 sem:rdf-load

`sem:rdf-load` 関数は、指定した場所にあるファイルからトリプルをロードし、パースしてデータベースに配置し、トリプルドキュメントの IRI を返します。トリプルのシリアライゼーションは指定できます。例えば Turtle ファイルの場合は `turtle`、RDF ファイルの場合は `rdfxml` です。

例えば以下のようになります。

```
sem:rdf-load('C:\rdfdata\example.rdf', "rdfxml")

=>
/triplestore/fbd28af1471b39e9.xml
```

`sem:rdf-insert` の場合と同様に、オプションでディレクトリまたは名前付きグラフが指定されていない限り、この関数もトリプルドキュメントをデフォルトグラフおよび `/triplestore/` ディレクトリに配置します。この例では、パラメータで `mynewgraph` を名前付きグラフとして指定しています。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/
  semantics"
  at "/MarkLogic/semantics.xqy";

   sem:rdf-load("C:\turtledata\example.ttl", "turtle",
   (), (),
        "mynewgraph")
```

次のようにドキュメントが挿入されます。

/triplestore/91b14a8e61b07c11.xml	E sem:triples	(no properties)	http://marklogic.com/semantics#default-graph, mygraph
/triplestore/9f2cfe9ecf2f87c9.xml	E sem:triples	(no properties)	http://marklogic.com/semantics#default-graph, mynewgraph

注： `sem:rdf-load` を使用するには、`xdmp:document-get` 権限が必要です。

3.2.4.3 sem:rdf-get

`sem:rdf-get` 関数は、指定された場所からトリプルファイルのトリプルを返します。以下の例では、Turtle シリアライゼーションでシリアライズされたトリプルをローカルのファイルシステムから取得します。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/
  semantics" at "/MarkLogic/semantics.xqy";

sem:rdf-get('C:\turtledata\people.ttl', "turtle")
```

トリプルは、Turtle シリアライゼーションのトリプルとして返され、行ごとに 1 つのトリプルが示されます。各トリプルはピリオドで終わります。

Run Result Auto Raw Profile Explorer

```
@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/Abraham_Lincoln> <http://dbpedia.org/ontology/birthDate> "1809-02-1
  <http://dbpedia.org/ontology/deathDate> "1865-04-15"^^xs:date ;
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> ;
  <http://xmlns.com/foaf/0.1/name> "Abraham Lincoln"@en ;
  <http://dbpedia.org/ontology/birthPlace> <http://dbpedia.org/resource/Hardin_County,_Kentucky>
  <http://purl.org/dc/elements/1.1/description> "16th President of the United States"@en .
<http://dbpedia.org/resource/Alexander_Mackenzie> <http://dbpedia.org/ontology/birthDate> "1822-(
  <http://dbpedia.org/ontology/deathDate> "1892-04-17"^^xs:date ;
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> ;
  <http://xmlns.com/foaf/0.1/surname> "Mackenzie"@en ;
  <http://dbpedia.org/ontology/deathPlace> <http://dbpedia.org/resource/Toronto> ;
  <http://xmlns.com/foaf/0.1/name> "Alexander Mackenzie"@en ;
  <http://dbpedia.org/ontology/birthPlace> <http://dbpedia.org/resource/Logierait> ,
    <http://dbpedia.org/resource/Perthshire> ,
    <http://dbpedia.org/resource/Scotland> ;
  <http://xmlns.com/foaf/0.1/givenName> "Alexander"@en ;
  <http://purl.org/dc/elements/1.1/description> "2nd Prime Minister of Canada (1873-1878)"@en .
<http://dbpedia.org/resource/Alan_Turing> <http://dbpedia.org/ontology/birthDate> "1912-06-23"^^
  <http://dbpedia.org/ontology/deathDate> "1954-06-07"^^xs:date ;
  <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.com/foaf/0.1/Person> ;
  <http://xmlns.com/foaf/0.1/surname> "Turing"@en ;
  <http://dbpedia.org/ontology/deathPlace> <http://dbpedia.org/resource/Manchester> ;
  <http://xmlns.com/foaf/0.1/name> "Alan Mathison Turing"@en ;
  <http://dbpedia.org/ontology/birthPlace> <http://dbpedia.org/resource/Paddington> ;
  <http://xmlns.com/foaf/0.1/givenName> "Alan Mathison"@en ;
  <http://purl.org/dc/elements/1.1/description> "Computer scientist, mathematician, and cryptogr
```

この Query Console の表示形式は、[Result] 部分から簡単にコピーできるようになっています。

3.2.5 REST API を使用したトリプルのロード

REST エンドポイントは MarkLogic サーバーの XQuery モジュールであり、HTTP リクエストをルーティングし、応答します。HTTP クライアントは、エンドポイントを呼び出して MarkLogic 内のコンテンツを作成、読み取り、更新、または削除します。このセクションでは、REST エンドポイントでトリプルをロードするために REST API を使用する方法について説明します。次のトピックから構成されます。

- 準備
- グラフストアの処理
- パラメータの指定
- サポートされている動詞
- サポートされているメディア形式
- トリプルのロード
- レスポンスエラー

3.2.5.1 準備

REST API およびエンドポイントに慣れていない場合は、『*REST Application Developer's Guide*』の「Introduction to the MarkLogic REST API」を参照してください。

REST エンドポイントでリクエストを実行するには、次の手順を実行します。

1. MarkLogic サーバーのバージョン 8.0-4 以降をインストールします。

2. HTTP リクエストを発行する `curl`、またはそれに相当するコマンドラインツールをインストールします。

3. デフォルトのデータベースおよびフォレスト（Documents）をポート 8000 で使用することも、独自に作成することもできます。新しいデータベースおよびフォレストを作成するには、『*Administrator's Guide*』の「Creating a New Database」を参照してください。

4. 管理画面の Documents データベース設定ページ、または Admin API を使用して、Documents データベースでトリプルインデックスおよびコレクションレキシコンがオンになっていることを確認します。「トリプルインデックスの有効化」（58 ページ）を参照してください。

注： コレクションレキシコンは、REST API インスタンスのグラフストア HTTP プロトコルで必須です。

5. ポート 8000 に関連付けられたデフォルトの REST API インスタンスを使用できます。新しい REST API インスタンスを作成する場合は、『*REST Application Developer's Guide*』の「Creating an Instance」を参照してください。

3.2.5.2 グラフストアの処理

グラフエンドポイントは、W3C のグラフストア HTTP プロトコルの実装であり、SPARQL 1.1 グラフストア HTTP プロトコルで指定されています。

http://www.w3.org/TR/2012/CR-sparql11-http-rdf-update-20121108

グラフストアのベース URL は次のとおりです。

```
http://hostname:port/version/graphs
```

「hostname」は MarkLogic サーバーのホストマシン、「port」は REST API インスタンスが実行されているポート、「version」は API のバージョン番号です。グラフストア HTTP プロトコルは、RESTful HTTP リクエストから対応する SPARQL 1.1 更新操作へのマッピングです。『*REST Application Developer's Guide*』の「Summary of the /graphs Service」を参照してください。

3.2.5.3 パラメータの指定

グラフエンドポイントでは、特定の名前付きグラフに対してオプションのパラメータを利用できます。例えば以下のようになります。

```
http://localhost:8000/v1/graphs?graph=http://named-graph
```

省略した場合、デフォルトグラフが値なしのデフォルトパラメータとして指定されている必要があります。

例えば以下のようになります。

```
http://localhost:8000/v1/graphs?default
```

パラメータなしで `GET` リクエストが発行されると、グラフのリストがリスト形式で返されます。詳細については、`GET:/v1/graphs` を参照してください。

3.2.5.4 サポートされている動詞

REST クライアントでは、MarkLogic サーバーとやり取りするために `GET` や `PUT` といった HTTP 動詞を使用します。次の表は、サポートされている動詞と、それぞれを使用するために必要なロールを示したものです。

動詞	説明	ロール
`GET`	名前付きグラフを取得します。	rest-reader
`POST`	名前付きグラフにトリプルをマージするか、空のグラフにトリプルを追加します。	rest-writer
`PUT`	名前付きグラフ内のトリプルを置換するか、空のグラフにトリプルを追加します。機能的には、DELETE の後に `POST` を実行することと同等です。例については、「トリプルのロード」（50 ページ）を参照してください。	rest-writer
`DELETE`	名前付きグラフ内のトリプルを削除します。	rest-writer
`HEAD`	グラフが存在するかどうかをテストします。ボディなしで、名前付きグラフを取得します。	rest-reader

MarkLogic の REST API リクエストを実行するために使用するロールでは、HTTP 呼び出しによってアクセスされるコンテンツに対して適切な権限（例えば、ターゲットデータベース内のドキュメントを読み取りまたは更新するパーミッション）が必要です。REST API のロールと権限の詳細については、『*REST Application Developer's Guide*』の「Security Requirements」を参照してください。

> **注：** このエンドポイントは、要素 `sem:triple` がルートであるドキュメントだけを更新します。

3.2.5.5 サポートされているメディア形式

`Content-type` HTTP ヘッダでサポートされているメディア形式のリストについては、「サポートされている RDF トリプルの形式」（31 ページ）を参照してください。

3.2.5.6 トリプルのロード

トリプルを挿入するには、`PUT` または `POST` リクエストを次の形式の URL にします。

```
http://host:port/v1/graphs?graph=graphname
```

リクエストを構築するときは、次の手順に従います。

1. トリプルの読み込み先グラフを指定します。

a. デフォルトグラフを指定する場合は、`graph` パラメータをデフォルトグラフに設定します。

b. 名前付きグラフを指定する場合は、`graph` パラメータをその名前付きグラフに設定します。

2. リクエストのボディにコンテンツを配置します。

3. `Content-type` HTTP ヘッダでコンテンツの MIME タイプを指定します。「サポートされている RDF トリプルの形式」（31 ページ）を参照してください。

4. ユーザー資格情報を指定します。

トリプルがデフォルトディレクトリの `/triplestore` に読み込まれます。

UNIX または Cygwin コマンドラインインタプリタでの `curl` コマンドの例を次に示します。このコマンドは、`PUT` HTTP リクエストを送信して、ファイル `example.nt` のコンテンツを XML ドキュメントとしてデータベースのデフォルトグラフに挿入します。

```
# Windows users, see  Modifying the Example Commands for Windows

$ curl -s -X PUT --data-binary '@example.nt' \
    -H "Content-type: application/n-triples" \
    --digest --user "admin:password" \
    "http://localhost:8000/v1/graphs?default"
```

注： `PUT` または `POST` を使用して REST エンドポイントでトリプルをロードする場合は、デフォルトグラフまたは名前付きグラフを指定する必要があります。

上記の例では、次の `curl` コマンドオプションを使用しています。

オプション	説明
`-s`	サイレントモードを指定します。そのため、`curl` の出力には、HTTP レスポンスヘッダは含まれません。レスポンスヘッダを含める場合の代替オプションは `-i` です。
`-X` *`http_method`*	`curl` で送信する HTTP リクエストのタイプ（`PUT`）。サポートされているその他のリクエストは、`GET`、`POST`、および `DELETE` です。「サポートされている動詞」（50 ページ）を参照してください。

オプション	説明
`--data-binary data`	リクエストのボディに含めるデータ。データは、`--data-binary` の引数としてコマンドラインに直接配置したり、`@filename` を使用してファイルから読み取ったりできます。Windows を使用している場合は、「`@`」演算子をサポートする `curl` の Windows バージョンが必要です。
`-H headers`	リクエストに含める HTTP ヘッダ。このガイドの例では、`Content-type` を使用しています。
`--digest`	ユーザーのパスワードは、指定した認証手法によって暗号化されます。
`--user user:password`	リクエストを認証するために使用されるユーザー名とパスワード。リクエストされた操作を実行するのに十分な権限を持つ MarkLogic サーバーユーザーを使用します。詳細については、『*REST Application Developer's Guide*』の「Security Requirements」を参照してください。

REST API の詳細については、『*REST Client API*』でセマンティックのドキュメントを参照してください。REST およびセマンティックの詳細については、「REST クライアント API でセマンティックを使用する」（194 ページ）を参照してください。

3.2.5.7 レスポンスエラー

このセクションでは、MarkLogic の REST API で採用されているエラーレポートの表記規則について説明します。

MarkLogic の REST API インスタンスに対するリクエストで障害が発生すると、エラーレスポンスコードが返され、追加情報の詳細がレスポンスのボディに含まれます。

次のようなレスポンスエラーが返されます。

- パラメータがまったくない `PUT` または `POST` リクエストのときは、`400 Bad Request` が返されます。
- パースで障害が発生するペイロードの `PUT` または `POST` リクエストのときは、`400 Bad Request` が返されます。
- 存在しない（コレクションレキシコンに IRI が存在しない）グラフに対する `GET` リクエストのときは、`404 Not Found` が返されます。

- サポートされていないシリアライゼーションのトリプルに対する GET リクエストのときは、406 Not Acceptable が返されます。
- サポートされていない形式の POST または PUT リクエストのときは、415 Unsupported Media Type が返されます。

  注： POST および PUT リクエストの repair パラメータは、true または false に設定できます。デフォルトでは false です。true に設定すると、適切にパースを実行しないペイロードであっても、パースを実行するトリプルを挿入します。false に設定すると、どのようなペイロードエラーでも 400 Bad Request レスポンスが返されます。

4.0 トリプルインデックスの概要

この章では、MarkLogic サーバーにおけるトリプルインデックスの概要について説明します。この章は、次のセクションで構成されています。

- トリプルインデックスとは / その使い方について
- トリプルインデックスの有効化
- その他の考慮事項

4.1 トリプルインデックスとは / その使い方について

トリプルインデックスは、ドキュメントの任意の場所で見つかった、スキーマが有効な `sem:triple` 要素をインデックス付けする目的で使用します。トリプルのインデックス付けは、トリプルが含まれているドキュメントが MarkLogic に読み込まれるとき、またはデータベースを再インデックス付けするときに実行されます。トリプルインデックスは、一意の値をそれぞれ一度だけ辞書に格納します。辞書により、それぞれの値に ID が与えられ、トリプルデータは、辞書内のその値を参照するときにその ID を使用します。

`sem:triple` 要素の有効性を判断するには、ドキュメント内の要素および属性を `sem:triple` スキーマ（`/MarkLogic/Config/semantics.xsd`）に対して確認します。`sem:triple` 要素が有効な場合、トリプルインデックスにエントリーが作成されます。有効でない場合、この要素はスキップされます。レンジインデックスとは異なり、トリプルインデックスはインメモリに収める必要がありません。そのため、事前のメモリ割り当てはほとんどありません。

> **注：** MarkLogic 9 以降の新しいインストールはすべて、トリプルインデックスおよびコレクションレキシコンがデフォルトでオンになります。また、新しいデータベースもすべて、トリプルインデックスおよびコレクションレキシコンがオンになります。

このセクションでは、次の内容を取り上げます。

- トリプルデータと値のキャッシュ
- トリプル値とタイプ情報
- トリプルポジション
- インデックスファイル
- 順列

4.1.1 トリプルデータと値のキャッシュ

MarkLogic では、内部的には 2 つの方法でトリプルが格納されます。トリプル値とトリプルデータです。トリプル値は、各トリプルの個々の値で、すべての型付きリテラル、

IRI、および空白ノードが含まれます。トリプルデータは、異なる並び順のトリプルを、ドキュメント ID およびポジションとともに保持します。トリプルデータは、トリプル値を ID で参照するため、参照が非常に効率的になります。トリプルデータはディスク上で圧縮されて格納され、トリプル値は別の圧縮値ストアに格納されます。トリプルインデックスと値ストアのどちらも、圧縮された 4KB のブロックに格納されます。

トリプルデータが必要になると（例えば参照時）、関連性のあるブロックがトリプルキャッシュまたはトリプル値キャッシュにキャッシュされます。他の MarkLogic キャッシュとは異なり、トリプルキャッシュやトリプル値キャッシュは拡大したり縮小したりするため、メモリを消費するのはキャッシュへの追加が必要なときだけです。

注： トリプルストアのホスト用のトリプルキャッシュおよびトリプル値キャッシュのサイズは、「キャッシュのサイズ設定」（61 ページ）の説明に従って設定できます。

4.1.1.1 トリプルキャッシュとトリプル値キャッシュ

トリプルキャッシュは、ディスクからの圧縮トリプルのブロックを保持します。このブロックは、LRU（least-recently-used：最近最も使用されなかったもの）アルゴリズムを使用してフラッシュされます。トリプルキャッシュ内のブロックは、辞書からの値を参照します。トリプル値キャッシュは、トリプルインデックス辞書からの非圧縮値を保持します。トリプル値キャッシュも、LRU のキャッシュです。

トリプルインデックス内のトリプルは、クエリのタイムスタンプおよびトリプルが属していたドキュメントのタイムスタンプに応じて除外されます。トリプルキャッシュは、フィルタリングが発生する前に生成された情報を保持するため、トリプルを削除しても、トリプルキャッシュには反映されません。ただし、マージ後、古いスタンドは削除される場合があります。スタンドが削除されると、そのブロックすべてがトリプルキャッシュからフラッシュされます。

トリプルインデックスブロックが最後に使用されてから、MarkLogic サーバーでそのブロックをキャッシュに保持する時間は、キャッシュタイムアウトで制御します（別のブロックのための空間を作るためにフラッシュされていない場合）。不定期で実行されるクエリに対してキャッシュを維持したい場合に、キャッシュタイムアウトを大きくすることが有効な可能性があります。不定期のクエリが再実行される前に、頻繁に発生する他のクエリによりブロックがキャッシュから押し出されることがあります。

4.1.2 トリプル値とタイプ情報

値は、ディスク上の別の値ストアに「値の等価性」のソート順で格納されます。つまり、ある特定スタンドにおいて、値 ID の順序は値の等価性の順序と同等ということになります。

値の中の文字列は、レンジインデックスの文字列ストレージに格納されます。タイムゾーンや生成されたタイプ情報など、値の等価性と関係のない部分は格納される値から削除されます。

タイプ情報は別に格納されるため、トリプルはトリプルインデックスから直接返すことができます。この情報は、SPARQL の単純含意で必要な RDF 固有の「sameTerm」比較に使用することもできます。

4.1.3 トリプルポジション

トリプルポジションインデックスは、`cts:triples` の `cts:triple-range-query` および `item-frequency` オプションを使用するクエリを正確に解決するために使用されます。また、トリプルポジションインデックスは、`cts:near-query` および `cts:element-query` コンストラクタを使用する検索を正確に解決する目的でも使用されます。トリプルポジションインデックスは、フラグメント内でのトリプルの相対位置をそのフラグメント内に格納します（通常、フラグメントはドキュメントです）。トリプルポジションインデックスを有効にするとインデックスのサイズが大きくなり、ドキュメントの読み込み速度が低下しますが、ポジション情報を必要とするクエリの精度は向上します。

例えば以下のようになります。

```
xquery version "1.0-ml";

cts:search(doc(),
  cts:near-query((
  cts:triple-range-query(sem:iri("http://www.rdfabout.com/
   rdf/usgov/sec/id/cik0001075285"), (), ()),

  cts:triple-range-query(sem:iri("http://www.rdfabout.com/
   rdf/usgov/sec/id/cik0001317036"), (), ())
   ),11), "unfiltered")
```

`cts:near-query` は、指定された距離内でマッチしているクエリのシーケンスを返します。ここで指定する距離は、2 つのマッチングクエリの間の単語数です。

フィルタリングされていない検索では、指定された `cts:query` を満たす候補であるインデックスからフラグメントを選択し、ドキュメントを返します。

4.1.4 インデックスファイル

メモリを効率的に使用するため、トリプルおよび値ストアのインデックスファイルは、メモリに直接マッピングされます。タイプストアは、全体がメモリにマッピングされます。

トリプルおよび値ストアのどちらも、64 バイトのセグメントで構成されたインデックスファイルを持ちます。それぞれの最初のセグメントは、チェックサム、バージョン番号、および（トリプルまたは値の）件数が含まれたヘッダです。その後は次の要素が続きます。

- トリプルインデックス：ヘッダセグメントの後に続くトリプルインデックスには、最初の 2 つの値のインデックスと、各ブロック内の最初のトリプルの異なる並び順（順列）のインデックスが、64 バイトのセグメントに編成されて格納されます。これは、トリプルからの値に基づいて指定された参照の回答を返すために必要なブロックを検索する目的で使用されます。現在、トリプルは序数でアクセスできないため、序数インデックスは必須ではありません。
- 値インデックス：ヘッダセグメントの後に続く値インデックスには、各ブロックの最初の値のインデックスが、64 バイトのセグメントに編成されて格納されます。値インデックスは、値に基づいて指定された参照の回答を返すために必要なブロックを検索する目的で使用されます。値インデックスの後には、各ブロックの開始序数のインデックスが続きます。このインデックスは、値 ID に基づいて指定された参照の回答を返すために必要なブロックを検索する目的で使用されます。

  注： トリプルインデックスは、`triple positions` がオンの場合にポジションを格納します。「トリプルインデックスの有効化」（58 ページ）を参照してください。

タイプストアには、格納されるタイプごとにタイプデータファイルに対するオフセットを格納するインデックスファイルが含まれています。これもメモリにマップされます。

次の表は、トリプルインデックスおよび値ストアで使用する情報を格納し、メモリにマップされるインデックスファイルについて説明したものです。

インデックスファイル	説明
`TripleIndex` `TripleValueIndex`	TripleData および TripleValueData のブロックインデックス
`TripleTypeData` `TripleTypeIndex`	トリプル値のタイプ情報
`StringData` `StringIndex` `AtomData` `AtomIndex`	文字列ベースのレンジインデックスでも使用されます。
`TripleValueFreqs` `TripleValueFreqsIndex`	トリプルに関する統計情報。トリプルインデックスは、データベースに保持されている各値のトリプルに関する統計情報を保持します。

4.1.5 順列

順列として、元のトリプル内の値を異なる並び順で保持します。異なるソート順で、またトリプルの各部分を効率的に参照できるように、3 つの順列（並び順）で格納しています。順列は、3 つの RDF 要素（主語（Subject）、述語（Predicate）、目的語（Object））のイニシャルから構成された略語として表現され、例えば { `SOP`, `PSO`, `OPS` } のようになります。

オプションで次のいずれかのソート順を指定するには、`cts:triples` 関数を使用します。

- `order-pso`：述語、主語、目的語の順に並んだ結果を返します。
- `order-sop`：主語、目的語、述語の順に並んだ結果を返します。
- `order-ops`：目的語、述語、主語の順に並んだ結果を返します。

4.2 トリプルインデックスの有効化

MarkLogic 9 以降のデータベースの場合、トリプルインデックスはデフォルトでオンになっています。このセクションでは、トリプルインデックスをオンにする方法またはトリプルインデックスがオンになっていることを確認する方法について説明します。また、関連するインデックスや設定についても説明します。次のトピックから構成されます。

- データベース設定ページの使用
- Admin API の使用

4.2.1 データベース設定ページの使用

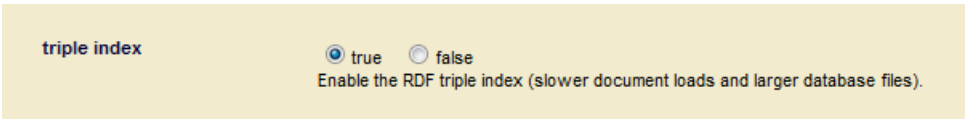
トリプルインデックスは、管理画面（`http://hostname:8001`）のデータベース設定ページでオン / オフを切り替えることができます。「hostname」は、トリプルインデックスをオンにする MarkLogic サーバーのホストです。

インデックス設定の詳細については、『*Administrator's Guide*』の「Index Settings that Affect Documents」および「トリプルを使用するためのデータベースの設定」（18 ページ）を参照してください。

> **注：** MarkLogic 9 以降の新しいインストールはすべて、トリプルインデックスがデフォルトでオンになります。また、新しいデータベースもすべて、トリプルインデックスがオンになります。既存のデータベースについて、トリプルインデックスがオンになっていることを確認できます。

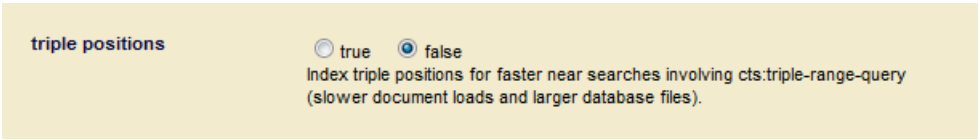
次の手順に従って、トリプルインデックスや関連の設定を確認または設定します。トリプルポジションインデックス、インメモリのトリプルインデックスサイズ、およびコレクションレキシコンをオンにするには、管理画面（`http://hostname:8001`）または Admin API を使用します。詳細については、「Admin API の使用」（59 ページ）を参照してください。

管理画面で、トリプルインデックスの設定まで下方向にスクロールし、true に設定します。



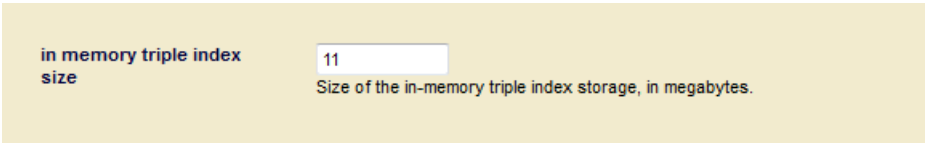
初めてトリプルインデックスをオンにする場合、またはトリプルインデックスをオンにした後でデータベースを再インデックス付けする場合は、有効な `sem:triple` 要素が含まれるドキュメントだけがインデックス付けされます。

トリプルポジションインデックスを有効にすることで、`cts:triple-range-query` を使用した近接検索を高速化できます。



ネイティブの SPARQL を使用したクエリでは、トリプルポジションインデックスをオンにする必要はありません。

インメモリスタンドのトリプルインデックスのデータを管理するために割り当てられるキャッシュおよびバッファメモリのサイズは設定できます。



注： データベースのいずれかのインデックス設定を変更すると、新しい設定は再インデックス付けがオンであるかどうか（`reindexer enable` が `true` に設定されているかどうか）に基づいて有効になります。

4.2.2 Admin API の使用

トリプルインデックスやトリプルインデックスのポジションをオンにしたり、データベースのインメモリトリプルインデックスのサイズを設定したりするには、Admin API 関数を使用します。

- `admin:database-set-triple-index`
- `admin:database-set-triple-positions`
- `admin:database-set-in-memory-triple-index-size`

次の例では、Admin API を使用して「Sample-Database」のトリプルインデックスを `true` に設定しています。

```
xquery version "1.0-ml";
import module namespace admin = "http://marklogic.com/
  xdmp/admin" at "/MarkLogic/admin.xqy";

(: Get the configuration :)
let $config := admin:get-configuration()

(: Obtain the database ID of 'Sample-Database' :)
let $Sample-Database := admin:database-get-id(
  $config, "Sample-Database")
let $c := admin:database-set-triple-index($config,
$Sample-Database, fn:true())
return admin:save-configuration($c)
```

この例は、Admin API を使用してデータベースのトリプルポジションを `true` に設定します。

```
xquery version "1.0-ml";
import module namespace admin = "http://marklogic.com/
  xdmp/admin" at "/MarkLogic/admin.xqy";

let $config := admin:get-configuration()
let $Sample-Database := admin:database-get-id(
  $config, "Sample-Database")
let $c := admin:database-set-triple-positions($config,
  $Sample-Database, fn:true())
return admin:save-configuration($c)
```

次の例では、データベースのインメモリトリプルインデックスのサイズを 256MB に設定しています。

```
xquery version "1.0-ml";
import module namespace admin =
"http://marklogic.com/xdmp/admin" at
"/MarkLogic/admin.xqy";

let $config := admin:get-configuration()
let $Sample-Database := admin:database-get-id(
  $config, "Sample-Database")
let $c := admin:database-set-in-memory-triple-index-
  size($config,$Sample-Database, 256)
return admin:save-configuration($c)
```

注： 関数のシグネチャおよび説明の詳細については、『*XQuery and XSLT Reference Guide*』の「`admin:database` 関数（database）」を参照してください。

4.3 その他の考慮事項

このセクションでは、次の内容を取り上げます。

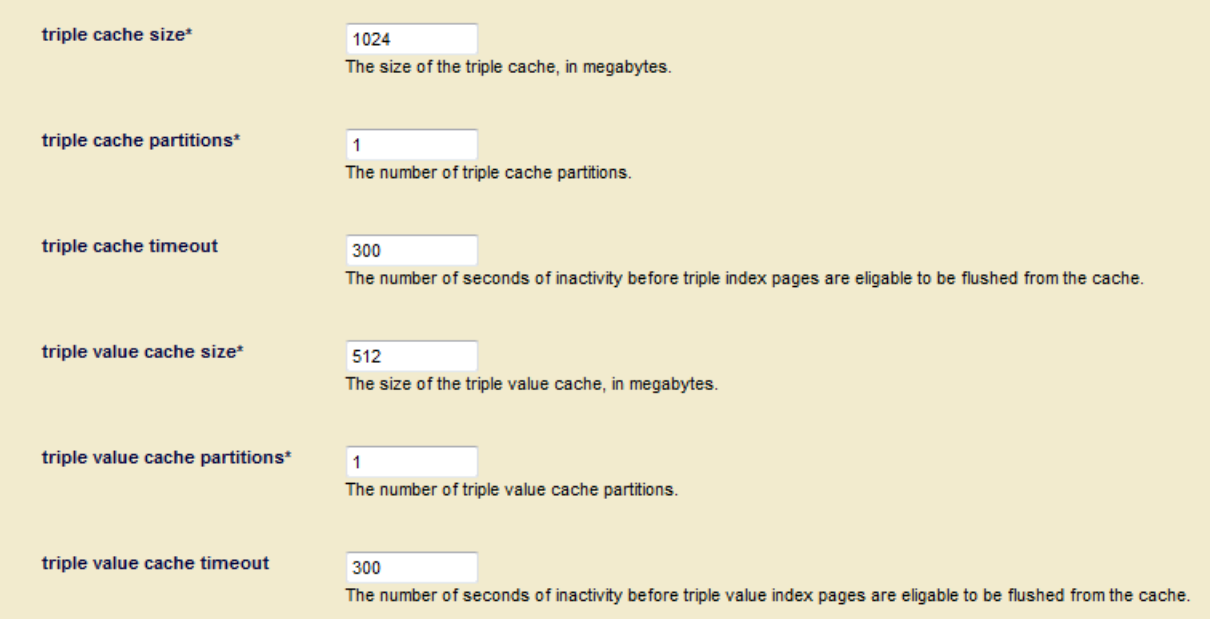
- キャッシュのサイズ設定
- 未使用の値とタイプ
- スケーリングとモニタリング

4.3.1 キャッシュのサイズ設定

トリプルキャッシュおよびトリプル値キャッシュは D ノードキャッシュであり、ロックの競合を避けるためパーティション化されています。このパーティション化により、並列処理がオンになり、処理がスピードアップされます。

キャッシュの最大サイズとパーティション数は設定可能です。ホストのトリプルまたはトリプル値のキャッシュサイズは、管理画面の［Groups］設定ページを使用するか、Admin API を使用して変更できます。

管理画面（`http://hostname:8001`）の［Groups］設定ページで、キャッシュサイズ、パーティション、およびタイムアウトの値を指定します。



次の表は、グループキャッシュの設定に対応する Admin API 関数について説明したものです。

関数	説明
admin:group-set-triple-cache-size	指定した ID を持つグループのトリプルのキャッシュサイズ設定を指定した値に変更します。
admin:group-set-triple-cache-partitions	指定した ID を持つグループのトリプルのキャッシュパーティション設定を指定した値に変更します。
admin:group-set-triple-cache-timeout	キャッシュ内で未使用のトリプルブロックが、キャッシュからフラッシュされるまでの秒数を変更します。
admin:group-set-triple-value-cache-timeout	キャッシュ内で未使用のトリプル値ブロックが、キャッシュからフラッシュされるまでの秒数を変更します。
admin:group-set-triple-value-cache-size	指定した ID を持つグループのトリプル値のキャッシュサイズ設定を指定した値に変更します。
admin:group-set-triple-value-cache-partitions	指定した ID を持つグループのトリプル値のキャッシュパーティション設定を指定した値に変更します。

4.3.2 未使用の値とタイプ

トリプル値およびタイプは、マージ中にトリプルインデックスによって使用されない場合があります。トリプルインデックスを 1 回のストリーミングパスでマージするために、タイプおよび値ストアはトリプルよりも前にマージされます。未使用の値とタイプは、トリプルのマージ中に識別されます。識別された未使用のタイプと値は次回のマージ中に削除され、それまで使用されていた領域が解放されます。

注： コンパクションを最大限に高めるためには、2 回のマージが必要です。MarkLogic サーバーは定期的にマージを実行するように設計されているため、これは通常の運用では問題になりません。

タイプストアは頻度によってソートされるため、インメモリで完全にマージされます。値およびトリプルストアは、ストリーミング方式で直接ディスクに対してマージされます。

マージの詳細については、『*Administrator's Guide*』の「Understanding and Controlling Database Merges」を参照してください。

4.3.3 スケーリングとモニタリング

SPARQL の実行ではフラグメントをフェッチしないため、トリプルのみを導入する場合には展開および圧縮ツリーキャッシュを縮小できる可能性があります。ツリーキャッシュは、管理画面の［Group］設定ページ、または次の関数を使用して設定できます。

```
admin:group-set-expanded-tree-cache-size
admin:group-set-compressed-tree-cache-size
```

データベースおよびフォレストのステータスは、管理画面のデータベースの［Status］ページから監視できます。

```
http://hostname:8001/
```

また、MarkLogic のモニタリングツールである Monitoring Dashboard および Monitoring History を使用することもできます。

```
http://hostname:8002/dashboard
http://hostname:8002/history
```

詳細については、『*Monitoring MarkLogic Guide*』の「Using the MarkLogic Server Monitoring Dashboard」を参照してください。

また、クエリのメトリックとして、またフォレストおよびキャッシュのステータスを監視する目的で、次の関数を使用できます。

- `xdmp:query-meters`：クエリのキャッシュヒット数またはミス数
- `xdmp:forest-status`：各スタンドのキャッシュヒット数、ミス数、ヒット率、およびミス率
- `xdmp:cache-status`：キャッシュパーティションによるビジー、使用、および解放の割合

5.0 非管理対象トリプル

XML または JSON ドキュメントの一部として含まれ、要素ノード sem:triple を持つトリプルは、unmanaged triples と呼ばれます。また、組み込みトリプルと呼ばれることもあります。この非管理対象トリプルは、sem:triple のスキーマで定義された MarkLogic XML または JSON 形式にする必要があります（semantics.xsd）。

> **注：** 非管理対象トリプルは、SPARQL Update では修正できません。このようなトリプルを修正する場合は、XQuery か JavaScript を使用してください。詳細については、「トリプルの更新」（267 ページ）を参照してください。

非管理対象トリプルを使用すると、MarkLogic はトリプルストアやドキュメントストアのように機能します。つまり、データに対してトリプルストアおよびドキュメントストアの機能を持つことになります。

次の例では、非管理対象トリプルを XML ドキュメント（Article.xml）に挿入しています。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

xdmp:document-insert("Article.xml",
<article>
  <info>
   <title>News for April 9, 2013</title>
    <sem:triples xmlns:sem="http://marklogic.com/semantics">
      <sem:triple>
       <sem:subject>http://example.com/article</sem:subject>
       <sem:predicate>http://example.com/mentions</
       sem: predicate>
       <sem:object
datatype="http://www.w3.org/2001/XMLSchema#string">London
</sem:object>
      </sem:triple>
    </sem:triples>
  </info>
</article>)
```

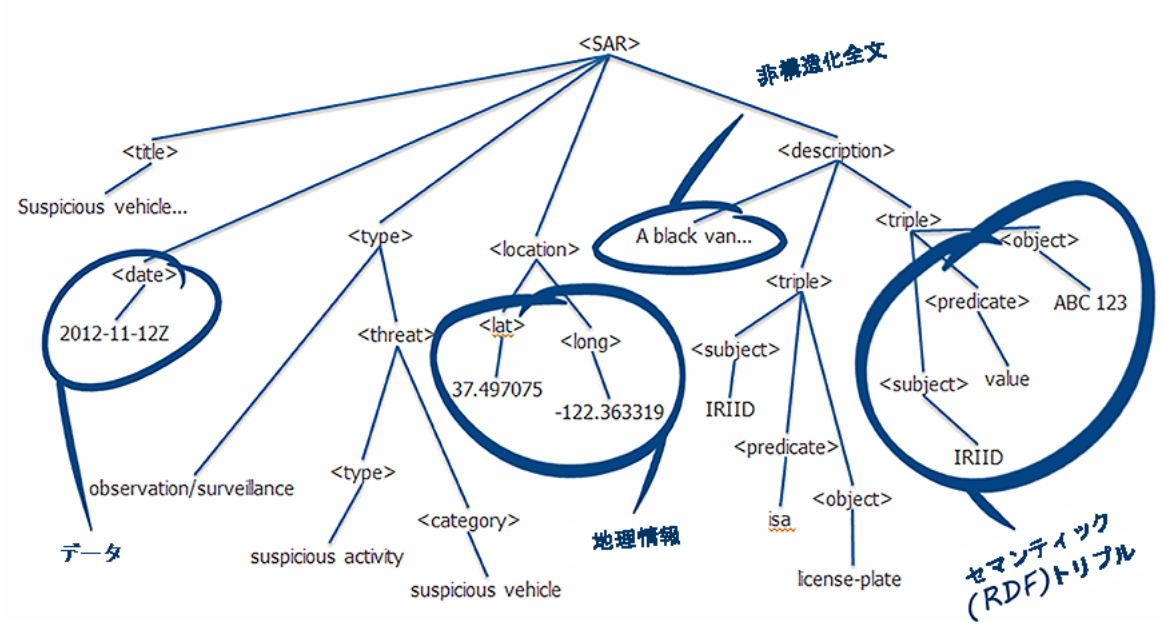
> **注：** sem:triples タグは省略して構いませんが、sem:triple タグは省略できません。

XML または JSON ドキュメントには、トリプルとともに多くの種類の情報を格納できます。

この例は、疑わしいアクティビティのレポートドキュメントを示したもので、XML とトリプルの両方が含まれます。

```
<SAR>
  <title>Suspicious vehicle...Suspicious vehicle near
  airport</title>
  <date>2014-11-12Z</date>
  <type>observation/surveillance</type>
  <threat>
    <type>suspicious activity</type>
    <category>suspicious vehicle</category>
  </threat>
  <location>
    <lat>37.497075</lat>
    <long>-122.363319</long>
  </location>
  <description>A blue van with license plate ABC 123 was
observed parked behind the airport sign...
    <sem:triple>
      <sem:subject>IRIID</sem:subject>
      <sem:predicate>isa</sem:predicate>
      <sem:object datatype="http://www.w3.org/2001/
XMLSchema#string">license-plate</sem:object>
    </sem:triple>
    <sem:triple>
      <sem:subject>IRIID</sem:subject>
      <sem:predicate>value</sem:predicate>
      <sem:object
datatype="http://www.w3.org/2001/XMLSchema#string">ABC
123</sem:object>
    </sem:triple>
  </description>
</SAR>
```

MarkLogic データベースに読み込まれた非管理対象トリプルは、triple index によってインデックス付けされ、SPARQL でアクセスおよびクエリできるように格納されます。同じ情報を別の方法で表現すると、次のようになります。



トリプルを JSON ドキュメントに埋め込むこともできます。JavaScript を使ってトリプルを挿入する方法を示します。

```
declareUpdate();
var sem = require("/MarkLogic/semantics.xqy");
xdmp.documentInsert(
 "testDoc.json", {
  "my": "data","triple":{
    "subject": "http://example.org/ns/dir/js/",
    "predicate": "http://xmlns.com/foaf/0.1/firstname/",
    "object": {"datatype" : "http://www.w3.org/2001/
     XMLSchema#string","value": "John"
      }
    }
  }
)
```

JSON ドキュメントに組み込まれたトリプルを次に示します。

```
{
"my": "data",
"triple":{
  "subject": "http://example.org/ns/dir/js/",
  "predicate": "http://xmlns.com/foaf/0.1/firstname/",
  "object": {
    "datatype" : "http://www.w3.org/2001/
     XMLSchema#string","value":"John"
```

```
    }
  }
}
```

同様に、XQuery を使用してもドキュメントへの挿入ができます。

```
xquery version "1.0-ml";
import module namespace sem =
"http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

xdmp:document-insert("myData.xml",
 <sem:triples xmlns:sem="http://marklogic.com/semantics">
   <sem:triple>
    <sem:subject>http://example.org/ns/dir/js/</sem:subject>
    <sem:predicate>http://xmlns.com/foaf/0.1/firstname/</sem
:predicate>
    <sem:object
datatype="http://www.w3.org/2001/XMLSchema#string">John</sem
:object>
   </sem:triple>
  </sem:triples>
)
```

トリプルを非管理対象トリプルとして XML または JSON ドキュメントに組み込む場合は、トリプルに関する追加情報を、追加メタデータ（トリプルの日時情報、バイテンポラル情報、ソース）とともに含めることができます。また、トリプルの出自など、有用な情報を XML または JSON ファイルに追加できます。トリプルを更新するときは、ドキュメントとトリプルを同時に更新します。

ドキュメントへのトリプルの追加のほか、インデックス付けするコンテンツをトリプルとして識別するためのテンプレートを使用することもできます。テンプレートの詳細については、「ドキュメント内のトリプルを識別するテンプレートを 使用する」（274 ページ）を参照してください。

5.1 XML ドキュメントでのトリプルの使用方法

非管理対象トリプルを使用すると、ドキュメントと、そのドキュメントが含んでいるトリプルの両方で複合クエリを実行できます。トリプルは、埋め込まれたドキュメント内の他の情報とともに「コンテキスト内」に留まります。トリプルはそのドキュメントに関連付けられたセキュリティおよびパーミッションを持ちます。このようなトリプルは、ドキュメントとともに更新され、そのドキュメントが削除されると削除されます。

5.1.1 ドキュメントからのコンテキスト

ドキュメントにトリプルがある場合は、ドキュメントでは、トリプルによって記述されたデータのコンテキストを提供できます。トリプルのソースならびにドキュメントとトリプルの作成時期に関する詳細な情報を、ドキュメントの一部として含めることができます。

```
<article>
 <info>AP Newswire - Nixon went to China</info>
 <triples-context>
  <confidence>80</confidence>
  <pub-date>2011-10-14</pub-date>
  <source>AP Newswire</source>
 </triples-context>
 <sem:triple xmlns:sem="http://marklogic.com/semantics">
  <sem:subject>http://example.org/news/Nixon</sem:subject>
  <sem:predicate>http://example.org/wentTo</sem:predicate>
  <sem:object datatype="http://www.w3.org/2001/
   XMLSchema#string">China</sem:object>
 </sem:triple>
</article>
```

トリプルに注釈を付けることで、情報に関する信頼性レベルなど、さらに詳細な情報を提供できます。

5.1.2 複合クエリ

複合クエリは、ドキュメントとトリプル（存在する場合）の両方に対して機能します。次の例は、AP Newswire ドキュメント内の情報に対する複合クエリを示したものです。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

sem:sparql('
  SELECT ?country
  WHERE {
    <http://example.org/news/Nixon> <http://example.org/
wentTo> ?country
  }
  ',
  (),
  (),
  cts:and-query( (
    cts:path-range-query( "//triples-context/confidence",
">=", 80) ,
   cts:path-range-query( "//triples-context/pub-date", ">",
```

```
        xs:date("1974-01-01")),cts:or-query( (
        cts:element-value-query( xs:QName("source"),
        "AP Newswire" ),
          cts:element-value-query( xs:QName("source"), "BBC" )
       ) )
     ) )
   )
```

この例の cts クエリは、フラグメントの集まりを識別しています。識別されたフラグメントにあるトリプルはセマンティックストアの構築に使用され、その後、そのストアに対して SPARQL クエリが実行されます。これが意味している、クエリが述べる内容は、次のとおりです。「cts クエリによって識別されたフラグメントにあるトリプルの中の国名を検索してください。ここで、識別されたフラグメントとは、`sem:triple/@confidence > 80` を持ち、1974 年よりも前の以前の `sem:triple/@date` を持ち、ソース要素に「`AP Newswire`」または「`BBC`」があるすべてのフラグメントのことです。」

5.1.3 セキュリティ

非管理対象トリプルでは、ドキュメントのセキュリティパーミッションがトリプルにも適用されます。そのため、トリプルを修正したりトリプルをドキュメントに追加したりするには、適切なパーミッションが必要です。ドキュメントの現在のパーミッションを調べるには、`xdmp:document-get-permissions` を使用します。

```
xquery version "1.0-ml";
xdmp:document-get-permissions("/example.json")

=>
<sec:permission xmlns:sec="http://marklogic.com/xdmp/
  security">
   <sec:capability>read</sec:capability>
   <sec:role-id>11180836995942796002</sec:role-id>
</sec:permission>
<sec:permission xmlns:sec="http://marklogic.com/xdmp/
  security">
   <sec:capability>update</sec:capability>
   <sec:role-id>11180836995942796002</sec:role-id>
</sec:permission>
```

ドキュメントのパーミッションを設定するには、`xdmp:document-set-permissions` を使用します。

```
xdmp:document-set-permissions("/example.json",
  (xdmp:permission("sparql-update-user","update"),
```

```
    xdmp:permission("sparql-update-user","read"))
)
```

ドキュメントパーミッションの詳細については、『*Security Guide*』の「Document Permissions」を参照してください。

5.2 バイテンポラルトリプル

SPARQL を使用すると、非管理対象トリプルでバイテンポラル検索クエリを実行できます。この例では、バイテンポラルクエリが SPARQL クエリ内部に `cts:period-range-query` としてラップされています。

```
let $q := '
SELECT
  ?derivation
WHERE {
  <http://example.com/prov/trader/>
   <http://www.w3.org/ns/prov#wasDerivedFrom/> ?derivation
  }
  '
return
  sem:sparql(
    $q,
    (),
    (),
    sem:store(
      (),
      cts:period-range-query(
        "valid",
        "ISO_CONTAINS",
      cts:period(
        xs:dateTime("2014-04-01T16:10:00"),
        xs:dateTime("2014-04-01T16:12:00")) )
    )
  )
```

このバイテンポラル SPARQL クエリは、`2014-04-01T16:10:00` から `2014-04-01T16:12:00` の間のイベントを検索します。テンポラルドキュメントの詳細については、『*Temporal Developer's Guide*』の「Understanding Temporal Documents」を参照してください。

6.0 セマンティッククエリ

この章では、RDF トリプルに対してセマンティッククエリを実行するときに使用される基本的な技術やツールについて説明します。トリプルのロードや削除と同様に、MarkLogic で RDF トリプルをクエリするときに優先的に使用する手法を選択できます。トリプルをクエリする方法はいくつかありますが、この章では主に SPARQL を使用してトリプルをクエリする方法を取り上げます。

MarkLogic では、SPARQL 1.1 のシンタックスおよび機能をサポートしています。SPARQL は、RDF トリプルのためのクエリ言語規格です。SPARQL 言語は、RDF データアクセスワーキンググループによる公式の W3C 勧告です。SPARQL 言語については、「SPARQL Query Language for RDF」勧告で説明されています。

http://www.w3.org/TR/rdf-sparql-query/

SPARQL クエリは、インメモリのトリプルまたはデータベースに格納されたトリプルをクエリするために、MarkLogic でネイティブに実行されます。データベースに格納されているトリプルをクエリする場合、SPARQL クエリはすべてトリプルインデックスに対して実行されます。SPARQL クエリを実行する例については、「トリプルのクエリ」（24 ページ）を参照してください。

SPARQL は、XQuery または JavaScript と組み合わせることができます。例えば、SPARQL クエリの範囲を `cts:query`（XQuery）や `cts.query`（JavaScript）で制限したり、ビルトイン関数（全文検索の `cts:contains` や `cts.contains` など）を SPARQL クエリの一部として呼び出したりできます。詳細については、「SPARQL クエリでのビルトイン関数の使用」（96 ページ）を参照してください。

トリプルをクエリするときは次の手法を使用できます。

- Query Console の SPARQL モード。詳細については、「SPARQL を使用したトリプルのクエリ」（72 ページ）を参照してください。
- セマンティック関数を使用した XQuery および search API、または XQuery と SPARQL の組み合わせ。詳細については、「XQuery または JavaScript によるトリプルのクエリ」（125 ページ）を参照してください。
- SPARQL エンドポイント経由の HTTP。詳細については、「REST クライアント API でセマンティックを使用する」（194 ページ）を参照してください。

注： この章では SPARQL キーワードを大文字で表記しますが、SPARQL キーワードには大文字 / 小文字の区別がありません。

この章は、次のセクションで構成されています。

- SPARQL を使用したトリプルのクエリ
- XQuery または JavaScript によるトリプルのクエリ
- オプティック API を使用したトリプルのクエリ
- シリアライゼーション
- セキュリティ

6.1 SPARQL を使用したトリプルのクエリ

このセクションでは、MarkLogic の SPARQL クエリ機能の概要について説明します。次のトピックから構成されています。

- SPARQL クエリのタイプ
- Query Console での SPARQL クエリの実行
- クエリ結果オプションの指定
- 結果のレンダリングの選択
- SPARQL クエリの構築
- プレフィックス宣言
- クエリパターン
- ターゲット RDF グラフ
- 結果節
- クエリ節
- ソリューション修飾子
- プロパティパス式
- SPARQL 集約
- SPARQL のリソース

注： このセクションの例では、http://downloads.dbpedia.org/3.8/en/persondata_en.ttl.bz2 の `persondata-en.ttl` データセットを使用します。「データセットのダウンロード」（20 ページ）を参照してください。

6.1.1 SPARQL クエリのタイプ

RDF データセットをクエリするときは、次の SPARQL クエリ形式を使用できます。

- SELECT クエリ：SPARQL の `SELECT` クエリは、ソリューションを返します。ソリューションは、変数と値のバインドの集まりです。
- CONSTRUCT クエリ：SPARQL の `CONSTRUCT` クエリは、RDF グラフ内の `sem:triple` 値のシーケンスとしてトリプルを返します。このトリプルは、トリプルテンプレート内の変数を置き換え、既存のトリプルから新しいトリプルを作成することにより構築されます。
- DESCRIBE クエリ：SPARQL の `DESCRIBE` クエリは、見つかったリソースを記述する RDF グラフとして `sem:triple` 値のシーケンスを返します。
- ASK クエリ：SPARQL の `ASK` クエリは、クエリパターンがデータセットにマッチするかどうかを示すブール型の値（`true` または `false`）を返します。

6.1.2 Query Console での SPARQL クエリの実行

SPARQL クエリを実行するには、次の手順を実行します。

1. Web ブラウザで、Query Console に移動します。

   ```
   http://hostname:8000/qconsole
   ```

   「hostname」は MarkLogic サーバーの名前です。

2. [Query Type] ドロップダウンリストから、[SPARQL Query] を選択します。

   Query Console では、SPARQL キーワードを強調表示するシンタックスをサポートしています。

   注： SPARQL Update を操作する場合は、[SPARQL Update] を選択してください。詳細については、「SPARQL Update」（172 ページ）を参照してください。

3. SPARQL クエリを構築します。「SPARQL クエリの構築」（78 ページ）を参照してください。

   ハッシュ記号（#）を先頭に付けることで、コメントを追加できます。

4. [Content Source] ドロップダウンリストから、ターゲットデータベースを選択します。

5. クエリウィンドウの下にあるコントロールバーで、[Run] をクリックします。

注： ターゲットデータベースでトリプルインデックスがオンになっていない場合は、XDMP-TRPLIDXNOTFOUND 例外がスローされます。詳細については、「トリプルインデックスの有効化」（58 ページ）を参照してください。

6.1.3 クエリ結果オプションの指定

Query Console では、SPARQL の結果は `json:object` 値のシーケンス（`SELECT` クエリの場合）、`sem:triple` 値のシーケンス（`CONSTRUCT` または `DESCRIBE` クエリの場合）、または単一の `xs:boolean` 値（`ASK` クエリの場合）として返されます。それぞれの結果に応じて、Query Console での表示が異なります。

このセクションでは、次の内容を取り上げます。

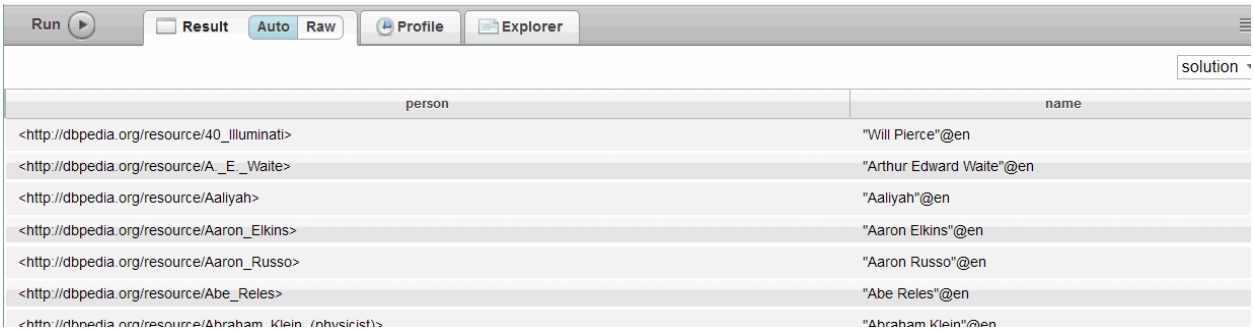
- Auto 形式と Raw 形式
- 結果のレンダリングの選択

6.1.3.1 Auto 形式と Raw 形式

SPARQL クエリの結果には、トリプルまたは `SELECT` ソリューションが表示されます。ソリューションオブジェクトは、変数名から型付き値へのマッピングを示します。結果シーケンスの別個の項目は固有のレンダリングを持ち、デフォルトでは Auto 形式で示されます。

例えば、この `SELECT` クエリは、次のソリューションを返します。

```
PREFIX db: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX onto: <http://dbpedia.org/ontology/>

SELECT ?person ?name
  WHERE { ?person onto:birthPlace db:Brooklyn;
    foaf:name ?name .}
```



表示形式を Raw に変更するには、[Result] タブの [Raw] をクリックします。Raw 形式では、同じクエリの結果が RDF/JSON シリアライゼーションで表示されます。

```
[
 {
  "person":"<http://dbpedia.org/resource/40_Illuminati>",
  "name":"\"Will Pierce\"@en"
 },
 {
  "person":"<http://dbpedia.org/resource/A._E._Waite>",
  "name":"\"Arthur Edward Waite\"@en"
 },
 {
  "person":"<http://dbpedia.org/resource/Aaliyah>",
  "name":"\"Aaliyah\"@en"
 },
 {
  "person":"<http://dbpedia.org/resource/Aaron_Elkins>",
  "name":"\"Aaron Elkins\"@en"
 },
 {
  "person":"<http://dbpedia.org/resource/Aaron_Russo>",
  "name":"\"Aaron Russo\"@en"
 },
 {
  "person":"<http://dbpedia.org/resource/Abe_Reles>",
  "name":"\"Abe Reles\"@en"
 },
 {
  "person":"<http://dbpedia.org/resource/Abraham_Klein_
  (physicist)>",
  "name":"\"Abraham Klein\"@en"
 },
 {
  "person":"<http://dbpedia.org/resource/Abraham_S._Fischler>",
  "name":"\"Abraham S.Fischler\"@en"
 },
 {
  "person":"<http://dbpedia.org/resource/Abraham_S._Luchins>",
  "name":"\"Abraham S.Luchins\"@en"
 },
 {
  "person":"<http://dbpedia.org/resource/Abram_Cohen>",
  "name":"\"Abram Cohen\"@en"
 }
]
```

同じように DESCRIBE クエリを実行すると、Query Console では出力がトリプル形式で返されます。

```
PREFIX db: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX onto: <http://dbpedia.org/ontology/>

DESCRIBE ?person ?name
WHERE { ?person onto:birthPlace db:Brooklyn;
foaf:name ?name .}

=>

@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/40_Illuminati>
<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/Brooklyn> ,
<http://dbpedia.org/resource/New_York> ;
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type/>
<http://xmlns.com/foaf/0.1/Person> ;
<http://xmlns.com/foaf/0.1/surname> "Pierce"@en ;
<http://purl.org/dc/elements/1.1/description> "Rapper"@en ;
<http://xmlns.com/foaf/0.1/givenName> "Will"@en ;
<http://xmlns.com/foaf/0.1/name> "Will Pierce"@en .
<http://dbpedia.org/resource/A._E._Waite>
<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/Brooklyn> ;
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type/>
<http://xmlns.com/foaf/0.1/Person> ;
<http://xmlns.com/foaf/0.1/givenName> "Arthur Edward"@en ;
<http://xmlns.com/foaf/0.1/name> "Arthur Edward Waite"@en ;
<http://purl.org/dc/elements/1.1/description> "English
writer"@en ;
<http://xmlns.com/foaf/0.1/surname> "Waite"@en .
<http://dbpedia.org/resource/Aaliyah>
<http://dbpedia.org/ontology/deathPlace>
<http://dbpedia.org/resource/Abaco_Islands> ,
<http://dbpedia.org/resource/Marsh_Harbour> ,
<http://dbpedia.org/resource/The_Bahamas> ;
<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/Brooklyn> ,
<http://dbpedia.org/resource/New_York_City> ;
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type/>
<http://xmlns.com/foaf/0.1/Person> ;
<http://xmlns.com/foaf/0.1/name> "Aaliyah"@en ;
<http://purl.org/dc/elements/1.1/description> "Singer,
```

```
dancer, actress, model"@en ;
<http://dbpedia.org/ontology/birthDate> "1979-01-16"
^^xs:date ;
<http://dbpedia.org/ontology/deathDate> "2001-08-25"
^^xs:date .
....
```

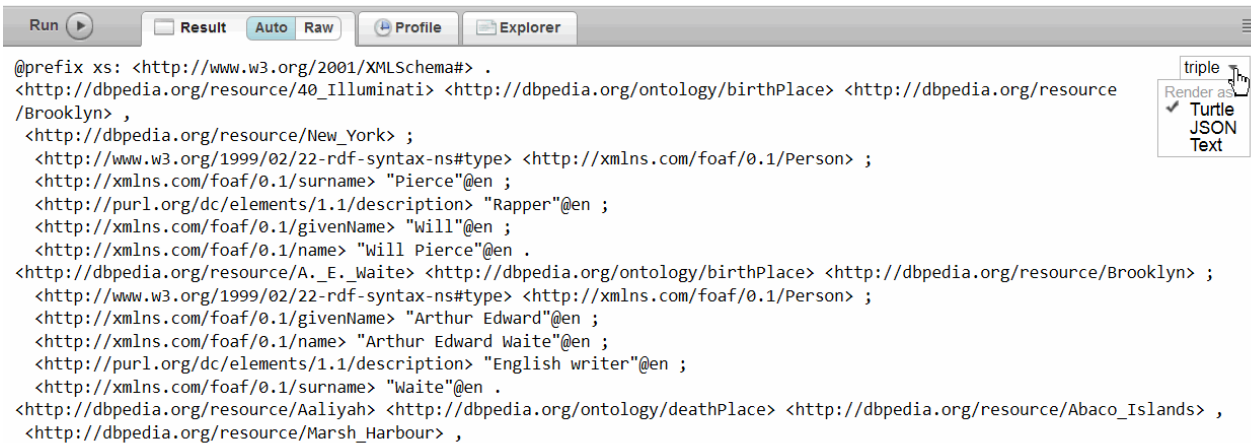
注： トリプルをサブグラフとして返すクエリを実行すると、デフォルトの出力シリアライゼーションは Turtle になります。

サーバーでは、DESCRIBE 節に 9999 個までというトリプル数制限があります。クエリに 1 つまたは複数の IRI を持つ DESCRIBE 節が含まれ、これらの IRI を合計してトリプルが 9999 個を超える場合、トリプルが結果から切り詰められます。切り詰められた場合でも、サーバーからの警告やメッセージは表示されません。

6.1.3.2 結果のレンダリングの選択

クエリ結果の表示方法は、[Render as] ドロップダウンリストのオプションを使用して選択します。例えば、この DESCRIBE クエリは、トリプルを Turtle シリアライゼーションで返します。

```
PREFIX db: <http://dbpedia.org/resource/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX onto: <http://dbpedia.org/ontology/>

DESCRIBE ?person ?name
WHERE { ?person onto:birthPlace db:Brooklyn;
foaf:name ?name .}
```



結果の形式として JSON やテキストを選択することもできます。

注： DESCRIBE クエリのレンダリングオプションは、Turtle、JSON、またはテキストです。cts:search を使用するクエリ、SPARQL と cts: クエリの組み合わせを使用するクエリ、またはシリアライゼーション関数でシリアライズされるクエリ結果を使用するクエリでは、レンダリングオプションが異なる場合があります。

6.1.4 SPARQL クエリの構築

SPARQL クエリを構築すると、トリプルに関して固有の質問をしたり、トリプルストア内のトリプルから新しいトリプルを作成したりできます。SPARQL クエリには、通常は次の各要素が（ここに示す順序で）含まれています。

- プレフィックス宣言：プレフィックス IRI を短縮します。
- クエリパターン：RDF グラフ内のクエリ対象を指定し、クエリパターンを比較してマッチします。
- ターゲット RDF グラフ：クエリするデータセットを識別します。
- 結果節：グラフから返される情報を指定します。
- クエリ節：クエリの範囲を拡張または制限します。
- ソリューション修飾子：結果を返す順序および結果の数を指定します。

クエリパターンおよび結果節は、クエリに最低限必要なコンポーネントです。プレフィックス宣言、ターゲット RDF グラフ、クエリ節、およびソリューション修飾子は、クエリを構成および定義するオプションのコンポーネントです。

次の例は、誕生地がパリである人を探すクエリパターンが含まれるシンプルな SPARQL の SELECT クエリです。

```
SELECT ?s
WHERE {?s <http://dbpedia.org/ontology/birthPlace/>
<http://dbpedia.org/resource/Paris>
}
```

以降のセクションでは、SPARQL クエリのコンポーネントの詳細と、シンプルなクエリおよび複雑なクエリを構成する方法について説明します。

6.1.5 プレフィックス宣言

IRI は長く扱いにくいものになってしまうことがあります。また、同じ IRI が 1 つのクエリで何回も使用されることがあります。クエリを簡潔にするため、SPARQL ではプレフィックスおよびベース IRI を定義できます。プレフィックスを定義することで時間を節約し、クエリがわかりやすいものになります。また、エラーを減らすことができます。広く使用される語彙のプレフィックスは、CURIE (Compact URI Expression) とも呼ばれます。

この例では、プレフィックス定義が宣言され、クエリパターンは短縮されたプレフィックスを使用して記述されています。

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX db: <http://dbpedia.org/resource/>
PREFIX onto: <http://dbpedia.org/ontology/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>

SELECT *
WHERE {
 ?s  dc:description "Physicist"@en ;
  rdf:type foaf:Person ;
 onto:birthPlace db:England .
  }
```

このクエリ結果は、イングランドで生まれた「物理学者」（Physicist）を返します。「@en」言語タグは、英単語「Physicist」を検索していることを意味します。このクエリは、「Physicist」と英語の言語タグが含まれるトリプルのみに一致します。

6.1.6 クエリパターン

SPARQL クエリの中核は、グラフパターンと呼ばれるトリプルパターンの集まりです。トリプルパターンは RDF トリプルに似ていますが、主語、述語、および目的語の各ノードが変数である点が異なります。

グラフパターンは、RDF データのサブグラフの RDF タームがこれらの変数と代替可能な場合に、そのサブグラフとマッチします。また、その結果はサブグラフと同等の RDF グラフです。

グラフパターンは、波括弧（{ }）内に含まれる 1 つあるいは複数のトリプルパターンです。この章では、クエリパターンに対応する次のタイプのグラフパターンについて説明します。

- 基本グラフパターン：トリプルパターンの集まりは、トリプルストア内のトリプルにマッチする必要があります。
- グループグラフパターン：グラフパターンの集まりは、同じ変数置換を使用してすべてマッチする必要があります。
- オプショングラフパターン：追加のパターンによってソリューションを拡張できます。
- 和集合グラフパターン：考えられる 2 つ以上のパターンについて試行されます。
- グラフグラフパターン：パターンは名前付きグラフにマッチされます。

SPARQL 変数は、疑問符（?）またはドル記号（$）付きで表記されます。変数は、任意の主語、述語、または目的語ノードにマッチさせたり、その位置の任意の値にマッチさせたりするように配置できます。つまり、変数は IRI またはリテラル（文字列、ブール型、日付など）にバインドできます。トリプルパターンは、トリプルストア内のトリプルにマッチするたびに、各変数ごとにバインドを生成します。

この例は、変数を含む基本グラフパターンを示したものです。このパターンは、目的語が「db:Paris」であるトリプルの主語（?s）と述語（?p）にマッチして、パリ出生またはパリ死没の人である主語を探します。このクエリは、2 つの部分で構成されています。SELECT 節は、クエリ結果に含ませる内容（主語と述語）を指定し、WHERE 節は、データグラフにマッチさせる基本グラフパターンを指定します。

```
PREFIX db: <http://dbpedia.org/resource/>
PREFIX onto: <http://dbpedia.org/ontology/>

SELECT ?s ?p
WHERE {?s ?p db:Paris }
```

このクエリは、パリで誕生または死亡した、データセットに含まれるすべての人物を返します。結果の数は、クエリの末尾に「LIMIT 10」を追加することで制限できます。詳細については、「LIMIT キーワード」（107 ページ）を参照してください。

注： 変数は 1 回だけバインドできます。SELECT 節の ?s および ?p は、WHERE 節内と同じ変数です。

クエリの結果は、トリプルの目的語の位置に「db:Paris」がある主語と述語の IRI（birthPlace の IRI および deathPlace の IRI）になります。

<http://dbpedia.org/resource/Étienne-Denis_Pasquier>	<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/Étienne-Louis_Malus>	<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/A._Kingsley_Macomber>	<http://dbpedia.org/ontology/deathPlace>
<http://dbpedia.org/resource/Abdul_Rasul_(Iraqi_scientist)>	<http://dbpedia.org/ontology/deathPlace>
<http://dbpedia.org/resource/Abdülmecid_II>	<http://dbpedia.org/ontology/deathPlace>
<http://dbpedia.org/resource/Abel_Decaux>	<http://dbpedia.org/ontology/deathPlace>

SPARQL の SELECT クエリは、ソリューションを返します。ソリューションは、変数と値のバインドの集まりです。デフォルトでは、SELECT クエリの結果は Auto 形式で返されます。これは確認しやすいように整形されたビューです。出力表示は変更できます。詳細については、「クエリ結果オプションの指定」（74 ページ）を参照してください。

前の例は、単一トリプルのパターンマッチ（基本グラフパターン）です。SPARQL では、複数トリプルのパターンマッチングを使用してクエリすることもできます。SPARQL では、Turtle に似たシンタックスを使用してクエリパターンを表現します。それぞれのトリプルパターンはピリオドで終わります。

SQL クエリの AND 節と同様に、クエリパターン内の各トリプルは完全にマッチする必要があります。例えば、Paris, Texas（テキサス州にあるパリ）と Paris, France（フランスにあるパリ）のように、データセット内に別の国で見つかる可能性のある場所名があるとします。

次の例は、フランスのパリで誕生し、「フットボール選手」（Footballer）と説明されているすべてのリソースの IRI を返します。

```
PREFIX db: <http://dbpedia.org/resource/>
PREFIX onto: <http://dbpedia.org/ontology/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT ?s
WHERE {?s onto:birthPlace db:Paris .
       ?s onto:birthPlace db:France .
       ?s dc:description "Footballer"@en .
  }
```

s	p
<http://dbpedia.org/resource/Abdoulaye_Baldé_(footballer)>	<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/Abdoulaye_Keita_(footballer_born_1990)>	<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/Abdoulaye_Méïté>	<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/Aboubacar_Sankhare>	<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/Aboubacar_Tandia>	<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/Ahmed_Soukouna>	<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/Alain_de_Martigny>	<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/Albert_Jourda>	<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/Alexandre_Raineau>	<http://dbpedia.org/ontology/birthPlace>

上記のクエリパターンを記述する別の方法として、WHERE 句でセミコロン（;）を使用して、同じ主語を共有するトリプルパターンを区切る方法が挙げられます。

例えば以下のようになります。

```
PREFIX db: <http://dbpedia.org/resource/>
PREFIX onto: <http://dbpedia.org/ontology/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
```

```
SELECT ?s
WHERE {?s onto:birthPlace db:Paris ;
          onto:birthPlace db:France ;
          dc:description "Footballer"@en .
  }
```

SPARQL 規格では、クエリのトリプルパターンの主語および目的語として空白ノードを使用することが認められています。

例えば以下のようになります。

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT  ?desc
WHERE { _:p rdf:type foaf:Person ;
            dc:description ?desc .
      }
```

このクエリは、データセットのトリプルで定義されたとおりにリソースの役割または肩書を返します。

desc
"Ukrainian musician"
"Competitive eater"
"American guitarist"
"American martial artist"
"Boxer"
"American guitarist"

注： クエリされたグラフに空白ノードがある場合、結果では空白のノードの識別子が返されることがあります。

6.1.7 ターゲット RDF グラフ

SPARQL クエリは、グラフが含まれる RDF データセットに対して実行されます。次のようなグラフが挙げられます。

- 単一のデフォルトグラフ：名前が割り当てられていないトリプルの集まりです。
- 1 つあるいは複数の名前付きグラフ：GRAPH 節の中では、各名前付きグラフは名前 1 つとトリプルの集まりで構成されたペアです。

例えば、次のクエリは http://my_collections という名前のグラフに対して実行されます。

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc:   <http://purl.org/dc/elements/1.1/>

SELECT ?name ?mbox ?date
FROM <http://my_collections>
WHERE { ?g dc:publisher ?name ;
       dc:date ?date .
      GRAPH ?g { ?person foaf:name ?name ;
      foaf:mbox ?mbox }
      }
```

「GRAPH キーワード」（87 ページ）では、クエリでの GRAPH の使い方について説明しています。

W3C の「SPARQL Query Language for RDF」で説明されているように、FROM および FROM NAMED キーワードは、SPARQL クエリで RDF データセットを指定する目的で使用されます。

http://www.w3.org/TR/rdf-sparql-query/#specifyingDataset

FROM または FROM NAMED キーワードがないと、SPARQL クエリはデータベース内に存在するすべてのグラフに対して実行されてしまいます。つまり、クエリでグラフ名を指定しない場合、すべてのグラフの和集合に対してクエリされます。

XQuery、REST、または JavaScript で次の方法を使用して、クエリ対象となる 1 つあるいは複数のグラフを指定することもできます。

- default-graph-uri*：クエリするグラフ名を選択します。通常は、使用可能なグラフのサブセットです。
- named-graph-uri*：FROM NAMED および GRAPH とともに使用して、特定の種類のクエリ内で名前を置換する IRI を指定します。クエリの一部として 1 つあるいは複数の named-graph-uri* パラメータを指定できます。

default-graph-uri* を指定した場合は、指定した 1 つあるいは複数のグラフ名がクエリされます。「*」は、1 つあるいは複数の default-graph-uri または named-graph-uri パラメータを指定できることを示します。

> **注：** この default-graph-uri は、名前が付いていないトリプルを含む「デフォルト」グラフ http://marklogic.com/semantics#default-graph ではありません。

この例では、http://example.org/bob/foaf.rdf グラフ内のデータセットを検索するために、SPARQL クエリが XQuery でラップされています。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
    at "/MarkLogic/semantics.xqy";

sem:sparql('
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?g ?name
WHERE {graph ?g { ?alum foaf:schoolHomepage
<http://www.ucsb.edu/> .
    ?alum foaf:knows ?person .
    ?person foaf:name ?name }
      }
      '
      ()
      ("default-graph-uri=http://example.org/bob/foaf.rdf")
```

SPARQL クエリの FROM は default-graph-uri と同様に機能し、FROM NAMED は named-graph-uri と同様に機能します。これら 2 つの節は、SPARQL クエリの一部として同様に機能しますが、一方はクエリ内に記述する（クエリ内にラップする）のに対し、他方はクエリの外側で指定します。

このセクションでは、次の内容を取り上げます。

- FROM キーワード
- FROM NAMED キーワード
- GRAPH キーワード

6.1.7.1 FROM キーワード

SPARQL クエリの FROM 節では、クエリ対象データの取得場所や、クエリ対象グラフを指定します。クエリの一部として FROM を使用するには、FROM 節に名前付きのグラフが記述されている必要があります。MarkLogic ではグラフ名はコレクションになっており、Query Console の Explore を使用して表示できます。

次の SPARQL クエリでは、FROM キーワードを使用して info:govtrack/people グラフ内のデータを検索します。

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

SELECT  ?name
FROM    <http://marklogic.com/semantics#info:
govtrack/people/>
```

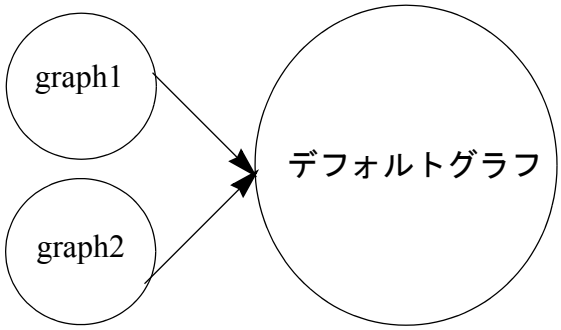
```
WHERE   { ?x foaf:name ?name }
LIMIT 10
```

GovTrack データセットについては、「例を実行する準備」（125 ページ）を参照してください。

デフォルトグラフは、1 つあるいは複数の FROM 節内で参照されているグラフの RDF マージ（グラフの和集合）の結果です。各 FROM 節には、デフォルトグラフを形成するために使用されているグラフを示す IRI が含まれます。

例えば、graph1 および graph2 がマージされて、デフォルトグラフを形成します。

```
FROM graph1
FROM graph2
```



注： この意味で、デフォルトグラフはデフォルトコレクション http://marklogic.com/semantics#default-graph と同じではありません。

次の例は、「Alice」が目的語の位置にあるすべてのトリプルを返す SPARQL の SELECT クエリを示したものです。RDF データセットには、単一のデフォルトグラフが含まれ、名前付きグラフは含まれません。

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

SELECT ?s ?p ?o
FROM <http://example.org/foaf/alice>
WHERE {?s foaf:name "Alice";
?p ?o .}
```

注： FROM キーワードは、WHERE 節よりも前に配置する必要があります。FROM キーワードを WHERE 節よりも後ろに配置すると、シンタックスエラーが発生します。

6.1.7.2 FROM NAMED キーワード

クエリでは、FROM NAMED 節を使用してデータセット内の名前付きグラフの IRI を指定できます。各 IRI は、データセット内の名前付きグラフ 1 つを指定するために使用されます。複数の FROM NAMED 節を記述すると、複数のグラフがデータセットに追加されます。FROM NAMED では、クエリで使用するグラフ名が、その節で提供されているグラフのみにマッチします。

named-graph は、ロード時に mlcp でコレクションパラメータ -output_collections http://www.example.org/my_graph を使用して設定できます。「コレクションおよびディレクトリの指定」（42 ページ）を参照してください。また、REST クライアントで PUT:/v1/graphs を使用して named-graph を設定することもできます。

> **注：** 名前付きグラフは、一般に RDF データをロードするときに作成されます。「トリプルのロード」（29 ページ）を参照してください。

クエリの FROM NAMED では、GRAPH キーワードを使用して WHERE 節からクエリされる名前付きグラフを識別します。

例えば以下のようになります。

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT ?who ?g ?mbox
FROM <http://example.org/foaf/aliceFoaf>
FROM NAMED <http://example.org/alice>
FROM NAMED <http://example.org/bob>
WHERE
{
   ?g dc:publisher ?who .
   GRAPH ?g { ?x foaf:mbox ?mbox }
}
```

この例では、FROM と FROM NAMED キーワードが同時に使用されています。FROM NAMED は、クエリの評価時に考慮されるグラフの範囲を設定するために使用されています。また、GRAPH コンストラクトでは、いずれかの名前付きグラフを指定しています。

> **注：** FROM または FROM NAMED キーワードを使用すると、GRAPH 節で使用できるグラフが制限される場合があります。

6.1.7.3 GRAPH キーワード

GRAPH キーワードは、データセット内の名前付きグラフに対するクエリの一部を評価するようにクエリエンジンに対して指示します。GRAPH 節で使用される変数は、別の GRAPH 節内、またはデータセットのデフォルトグラフにマッチするグラフパターンで使用することもできます。

例えば以下のようになります。

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc:   <http://purl.org/dc/elements/1.1/>

SELECT ?name ?mbox ?date
WHERE { ?g dc:publisher ?name ;
       dc:date ?date .
      GRAPH ?g { ?person foaf:name ?name ;
      foaf:mbox ?mbox }
      }
```

注： SPARQL クエリで GRAPH コンストラクトを使用する場合は、コレクションレキシコンをオンにする必要があります。コレクションレキシコンは、データベース設定ページまたは管理画面からオンにできます。

明示的な IRI を含む GRAPH 節（例えば GRAPH <....uri...> { ...graph pattern...}）は、GRAPH 節で指定された IRI を使用してデータセットにマッチします。

6.1.8 結果節

異なるタイプの SPARQL クエリでデータセットをクエリすると、返される結果のタイプも異なります。SPARQL クエリの各形式に応じて、次のような結果節が返されます。

- SELECT クエリ：変数バインドのシーケンスを返します。
- CONSTRUCT クエリ：トリプルテンプレート内で変数を置換することによって構築される RDF グラフを返します。
- DESCRIBE クエリ：見つかったリソースを記述する RDF グラフを返します。
- ASK クエリ：クエリパターンとマッチするかどうかを示すブール型の値を返します。

6.1.8.1 SELECT クエリ

SPARQL の SELECT キーワードは、データセット内のデータをリクエストしていることを示します。この SPARQL クエリは、クエリ形式の中で最もよく使用されるものです。SPARQL の SELECT クエリは、クエリを満たすバインドのシーケンスをソリューションとして返します。選択した変数は、カンマではなくスペースで区切られます。

クエリパターンで識別されるすべての変数を選択する場合の省略表記として、SPARQL の SELECT ではアスタリスクのワイルドカード記号（*）を使用できます。

例えば以下のようになります。

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

SELECT *
WHERE{?s foaf:givenName ?fn .
       ?s foaf:surname ?ln .
     }
```

注： 単一のトリプルパターンでは、末尾のピリオドはオプションです。また複数のトリプルパターンを持つクエリパターンでは、最後のトリプルの末尾のピリオドはオプションです。

この例では、SELECT クエリは主語変数（?s）の IRI と、データセット内のリソースの名（?fn）および姓（?ln）で構成されたシーケンスを返します。

SPARQL の SELECT クエリの結果は、XML または JSON でシリアライズされるか、別の関数にマップとして渡されます。SELECT クエリの結果は、必ずしもトリプルになるとは限りません。

6.1.8.2 CONSTRUCT クエリ

SPARQL の CONSTRUCT クエリを使用すると、既存のトリプルから新しいトリプルを作成できます。CONSTRUCT クエリを実行すると、結果はインメモリのトリプルとして sem:triple 値のシーケンスで返されます。

次の例は、データベース内の既存のトリプルから Albert Einstein に関するトリプルを作成します。

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

CONSTRUCT
{?person ?p ?o .}
WHERE {?person foaf:givenName "Albert"@en ;
       foaf:surname "Einstein"@en ;
       ?p ?o .}
```

CONSTRUCT クエリは、クエリパターン内の変数から作成された RDF グラフを返します。

次に示すトリプルは、Albert Einstein について、データセット内の既存のトリプルから作成されたものです。

```
@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/Albert_Einstein>
<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/Baden-Württemberg> .

@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/Albert_Einstein>
<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/German_Empire> .

@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/Albert_Einstein>
<http://dbpedia.org/ontology/deathPlace>
<http://dbpedia.org/resource/Princeton,_New_Jersey> .

@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/Albert_Einstein>
<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/Ulm> .

@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/Albert_Einstein>
<http://dbpedia.org/ontology/deathPlace>
<http://dbpedia.org/resource/United_States> .

@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/Albert_Einstein>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/0.1/Person> .

@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/Albert_Einstein>
<http://xmlns.com/foaf/0.1/givenName> "Albert"@en .

@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/Albert_Einstein>
<http://xmlns.com/foaf/0.1/name> "Albert Einstein"@en .

@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/Albert_Einstein>
<http://xmlns.com/foaf/0.1/surname> "Einstein"@en .
```

```
@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/Albert_Einstein>
<http://purl.org/dc/elements/1.1/description>
"Physicist"@en .
```

これらのトリプルは、インメモリで構築され、データベースには追加されません。

注： 「@en」言語タグは、それが英語の単語であり、他の言語タグとは異なるマッチになることを意味します。

6.1.8.3 DESCRIBE クエリ

SPARQL の `DESCRIBE` クエリは、`sem:triple` 値のシーケンスを返します。`DESCRIBE` クエリの結果は、1 つあるいは複数の指定されたリソースについて記述された RDF グラフを返します。W3C 規格では、詳細は実装依存とされています。MarkLogic では、識別された IRI の「Concise Bounded Desciption」を返します。これには、主語としてその IRI を持つすべてのトリプル、および（目的語として空白ノードを持つトリプルそれぞれについて）主語としてそれらの空白ノードを持つすべてのトリプルが含まれます。この実装では、具体化されたステートメントは提供されないため、最大で 9999 個のトリプルを返します。

例えば、次のクエリは「Pascal Bedrossian」を含んでいるトリプルを探します。

```
DESCRIBE <http://dbpedia.org/resource/Pascal_Bedrossian>
```

`DESCRIBE` クエリで探されたトリプルは、Turtle 形式で返されます。形式として、JSON やテキストを選択することもできます。

```
@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/Pascal_Bedrossian>
<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/France> .

@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/Pascal_Bedrossian>
<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/Marseille> .

@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/Pascal_Bedrossian>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/0.1/Person> .

@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/Pascal_Bedrossian>
<http://xmlns.com/foaf/0.1/surname> "Bedrossian"@en .
```

```
@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/Pascal_Bedrossian>
<http://xmlns.com/foaf/0.1/givenName> "Pascal"@en .

@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/Pascal_Bedrossian>
<http://xmlns.com/foaf/0.1/name> "Pascal Bedrossian"@en .

@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/Pascal_Bedrossian>
<http://purl.org/dc/elements/1.1/description>
"footballer"@en .

@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/Pascal_Bedrossian>
<http://dbpedia.org/ontology/birthDate> "1974-11-28"
^^xs:date .
```

注： サーバーでは、DESCRIBE 節に 9999 個までというトリプル数制限があります。つまり、クエリに 1 つまたは複数の IRI を持つ DESCRIBE 節が含まれ、これらの IRI を合計してトリプルが 9999 個を超える場合、トリプルが結果から切り詰められます。切り詰められた場合でも、サーバーからの警告やメッセージは表示されません。

6.1.8.4 ASK クエリ

SPARQL の ASK クエリは、単一の xs:boolean 値を返します。クエリパターンがデータセット内で何らかのマッチを持つ場合、ASK 節は true を返します。パターンマッチがない場合は、false を返します。

例えば、ケネディ家の 2 人、キャロリン・ベセット・ケネディとユーニス・ケネディ・シュライバーに関して、persondata データセットに次のファクトがあるとします。

- スペシャルオリンピックスの前身となるイベントの創設者であり、ジョン・F・ケネディの妹であるユーニス・ケネディ・シュライバーは、1921 年 7 月 10 日に誕生した。
- 広報担当者であり、JFK ジュニアの妻であるキャロリン・ベセット・ケネディは、1966 年 1 月 7 日に誕生した。

次のクエリは、キャロリンがユーニスの後に誕生したかどうかを尋ねています。

```
PREFIX db: <http://dbpedia.org/resource/>
PREFIX onto: <http://dbpedia.org/ontology/>

ASK
```

```
{
  db:Carolyn_Bessette-Kennedy onto:birthDate ?by .
  db:Eunice_Kennedy_Shriver onto:birthDate ?bd .
  FILTER (?by>?bd).
}
=>
true
```

応答は `true` です。

注： `ASK` クエリは、結果が少なくとも 1 つあるかどうかを確認します。

6.1.9 クエリ節

次のクエリ節を追加して、返される結果の数を増やしたり減らしたりできます。

- OPTIONAL キーワード
- UNION キーワード
- FILTER キーワード
- 比較演算子
- フィルタ式内での否定
- BIND キーワード
- Values セクション

6.1.9.1 OPTIONAL キーワード

`OPTIONAL` キーワードは、オプションのグラフパターンにマッチがある場合に追加の結果を返す目的で使用します。例えば次のクエリパターンは、データベース内で名（`?fn`）、姓（`?ln`）、およびメールアドレス（`?mb`）で構成されたトリプルを返します。

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

SELECT ?fn ?ln
WHERE{?x foaf:givenName ?fn .
      ?x foaf:surname ?ln .
      ?x foaf:email ?mb .
}
```

すべてのトリプルパターンにマッチするトリプルだけが返されます。`persondata` データセットには、メールアドレスがない人物が存在する可能性があります。このような場合、Query Console では、そのような人物が結果セットから暗黙で除外されます。

オプションのグラフパターン（左結合とも呼ばれます）を使用すると、共通する任意の変数でマッチする値を返すことができます（存在する場合）。OPTIONAL キーワードはグラフパターンでもあるため、（WHERE 節の波括弧内には）独自の波括弧の集まりを持ちます。

次の例では、前出の例を拡張して、1 つあるいは複数のメールアドレスを返し、メールアドレスがない場合は名と姓だけを返します。

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

SELECT ?fn ?ln
WHERE {?x foaf:givenName ?fn .
       ?x foaf:surname ?ln .
OPTIONAL{?x foaf:email ?mb .}
}
```

注： オプションパターンでは、バインドなしの変数が発生することがあります。バインドなしの変数の詳細については、「ORDER BY キーワード」（108 ページ）を参照してください。

6.1.9.2 UNION キーワード

UNION キーワードを使用すると、複数の異なるデータの集まりから複数のパターンをマッチし、それらをクエリ結果内で結合します。UNION キーワードは、WHERE 節の波括弧の内側に配置します。シンタックスは次のとおりです。

```
{ triple pattern } UNION { triple pattern }
```

UNION パターンは、グラフパターン同士を結合します。それぞれが別の可能性として、複数のトリプルパターンを含む場合があります（論理和）。

次の例では、「Authors」または「Novelists」と説明されている人々とその誕生日を検索します。

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX onto: <http://dbpedia.org/ontology/>

SELECT  ?person ?desc ?date
WHERE { ?person rdf:type foaf:Person .
      ?person dc:description ?desc .
      ?person onto:birthDate ?date .

      { ?person dc:description "Novelist"@en .}
```

```
UNION
      { ?person  dc:description "Author"@en .}
      }
```

グループグラフパターン構造を使用して、トリプルパターンを複数のグラフパターンにグループ化することもできます。

例えば以下のようになります。

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT ?person ?desc
WHERE {{?person rdf:type foaf:Person }
       {?person dc:description ?desc }

     {{?person dc:description "Author"@en }

 UNION
      { ?person  dc:description "Novelist"@en .} } }
```

波括弧の組ごとにトリプルが含まれていることに注意してください。これは、次に示すクエリとセマンティック的には同じであり、同じ結果が得られます。

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT ?person
WHERE {?person rdf:type foaf:Person ;
               dc:description ?desc .

   {?person dc:description "Author"@en }

UNION
   {?person  dc:description "Novelist"@en .}
      }
```

注： SPARQL クエリでは、複数の UNION パターンを使用できます。OPTIONAL クエリと UNION クエリからの結果は異なります。UNION クエリでは別のソリューションのサブグラフが許容されますが、OPTIONAL クエリでは明示的には許容されません。

6.1.9.3 FILTER キーワード

SPARQL クエリの結果は、複数の手法で制限できます。返されるマッチング結果の数を制限する場合は、FILTER、DISTINCT、または LIMIT キーワードを使用できます。

1 つあるいは複数の SPARQL の FILTER キーワードを使用すると、結果を制約する変数を指定できます。FILTER 制約は、WHERE 節の波括弧の内側に配置します。論理、算術、または比較演算子の記号（大なり（>）、小なり（<）、等価（=）など）を含めることができます。FILTER 制約では、マッチングクエリの結果を返すためにブール型条件が使用されます。また、isURI、isBlank など、使用できるビルトインの SPARQL テストも多数用意されています。

次の表は、FILTER 制約で使用される SPARQL 単項演算子の一部を示したものです。

演算子	型	結果の型
!	xsd:boolean	xsd:boolean
+	数値	数値
-	数値	数値
BOUND()	変数	xsd:boolean
isURI()	RDF ターム	xsd:boolean
isBLANK()	RDF ターム	xsd:boolean
isLITERAL	RDF ターム	xsd:boolean
STR()	リテラル /IRI	単純リテラル
LANG()	リテラル	単純リテラル
DATATYPE()	リテラル	IRI

注： 演算子の詳細なリストについては、「SPARQL Query Language for RDF」の「*Operator Mapping*」を参照してください。

次の例は、変数 ?bd（人の誕生日）に意味を与えるクエリパターンです。クエリパターンの FILTER 節では、この変数の値と 1999 年 1 月 1 日という日付とを比較し、この日付よりも後に生まれた人々を返します。

```
PREFIX onto: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
```

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>

SELECT ?s
WHERE {?s rdf:type foaf:Person .
       ?s onto:birthDate ?bd .
  FILTER (?bd > "1999-01-01"^^xsd:date)
}
```

SPARQL キーワード a は、共通の述語 rdf:type のショートカットであり、リソースのクラスを指定します。例えば、WHERE 節は次のように記述できます。

```
WHERE {?s a foaf:Person .
       ?s onto:birthDate ?bd .
```

regex 関数を使用すると、FILTER 節を正規表現パターンで表現できます。例えば以下のようになります。

```
SELECT ?s ?p ?o
WHERE {?s ?p ?o
  FILTER (regex (?o, "Lister", "i"))
}
```

この SPARQL クエリは、目的語の位置のテキストに、大文字 / 小文字の区別なしで文字列 Lister が含まれるすべてのマッチング結果を返します。i フラグを使用すると、正規表現のマッチは大文字 / 小文字の区別なしで実行されます。

注： このタイプの FILTER クエリは、fn:match XQuery 関数と同等です。SPARQL では正規表現が最適化されません。最適化された全文検索を実行するには cts:contains を使用してください。

正規表現言語は、「*XQuery 1.0 and XPath 2.0 Functions and Operators*」のセクション 7.6.1「Regular Expression Syntax」で定義されています。

SPARQL クエリでのビルトイン関数の使用

FILTER、BIND のほか、SELECT ステートメント内の式が含まれる、関数を使用できる SPARQL クエリでは、SPARQL 関数以外にも、XQuery や JavaScript のビルトイン関数が使用できます。例えば、プレフィックス fn、cts、math、xdmp を持つ関数が使用できます。

ビルトイン関数とは、XQuery の場合は「import module」、JavaScript の場合は「var <module> = require」を使用せずに呼び出すことができる関数です。SPARQL クエリで使用する場合、このような関数のことを拡張関数と呼びます。ビルトイン関数のリストは、http://docs.marklogic.com で「Server-Side JavaScript APIs」（または「Server-Side XQuery APIs」）を選択すると確認できます。また、ビルトインのリストは、「MarkLogic Built-In Functions」および「W3C-Standard Functions」にあります。詳細については、「セマンティック関数を使用したクエリ」（127 ページ）を参照してください。

SPARQL の拡張関数は、`http://www.w3.org/2005/xpath-functions#`*name* という形式の IRI（例えば `http://www.w3.org/2005/xpath-functions#starts-with`）で識別されます。「name」は関数のローカル名、# より前の文字列は関数のプレフィックス IRI です。一般的に `fn`、`cts`、`math`、`xdmp` と結合されたプレフィックス IRI（または「/」や「#」で終わらない他のあらゆるプレフィックス IRI）の場合は、プレフィックス IRI に # を付け、その後に関数のローカル名を続けます。例えば次のようになります。`http://marklogic.com/cts#contains`.

`cts` などのビルトイン関数にアクセスするには、SPARQL クエリで `PREFIX` を使用します。次の例では、`cts:contains` は `PREFIX` を使用することで追加され、その後は `FILTER` クエリの一部として含められています。

```
PREFIX cts: <http://marklogic.com/cts#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>

SELECT *
WHERE{ ?s ?p ?o .
  FILTER cts:contains(?o, cts:or-query(("Monarch",
"Sovereign")))
  FILTER(?p IN (dc:description, rdfs:type))
}
```

これは「Monarch」または「Sovereign」という語の全文検索であり、述語は説明（description）またはタイプ（type）です。2 番目の `FILTER` 節では `IN` を使用して、フィルタを適用する述語を指定しています。結果には、（王国、州、または領域で）「Monarch」（君主）の肩書を持つ人々と、Monarch butterfly（オオカバマダラ）、Monarch Islands（モナーク諸島）など「Monarch」という説明が含まれる項目が含まれます。

次の例では、SPARQL クエリで XPath 関数 `starts-with` を使用して、説明が「Chief」で始まる人々の役割または肩書を返します。この関数をインポートするには、`FILTER` クエリの一部として IRI を含めます。

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

SELECT ?desc
WHERE {?s dc:description ?desc
  FILTER (<http://www.w3.org/2005/xpath-functions#starts-
with>( ?desc,"Chief" ) )}
```

注： FILTER キーワードは、OPTIONAL および UNION キーワードと併用できます。

6.1.9.4 比較演算子

IN および NOT IN 比較演算子は FILTER 節で使用され、マッチするタームが式の集まりに含まれる場合はブール型の値 true を返し、それ以外の場合は false を返します。例えば以下のようになります。

```
ASK {
  FILTER(2 IN (1, 2, 3))
}

=>
true

ASK {
  FILTER(2 NOT IN (1, 2, 3))
}

=>
false
```

6.1.10 フィルタ式内での否定

否定を FILTER 式と併用すると、クエリの結果からソリューションを除外できます。否定には 2 タイプあります。一方のタイプは、フィルタリングされるクエリソリューションのコンテキストでグラフパターンがマッチするかどうかに応じて結果をフィルタリングします。もう一方のタイプは、他のパターンに関連するソリューションを削除することに基づきます。MarkLogic は、FILTER との併用には、（EXISTS、NOT EXISTS、および MINUS を使用する）SPARQL 1.1 Negation をサポートしています。

否定の使用例では、次のデータを使用します。

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

:alice  rdf:type   foaf:Person .
```

```
:alice  foaf:name  "Alice" .
:bob    rdf:type   foaf:Person .
```

このセクションでは、次の内容を取り上げます。

- EXISTS
- NOT EXISTS
- MINUS
- NOT EXISTS と MINUS の違い
- 否定を含む複合クエリ

6.1.10.1 EXISTS

フィルタ式 `EXISTS` は、データ内にクエリパターンが見つかるかどうかをチェックします。例えば、この例の `EXISTS` フィルタは、データ内にパターン `?person foaf:name ?name` がないかチェックします。

```
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?person
WHERE
{
    ?person rdf:type  foaf:Person .
    FILTER EXISTS { ?person foaf:name ?name }
}

=>
    person
<http://example.org/alice
```

このクエリの結果は、「Alice」です。`EXISTS` フィルタは、追加バインドは生成しません。

6.1.10.2 NOT EXISTS

`NOT EXISTS` フィルタ式を使用した場合、クエリは、フィルタが出現するグループグラフパターン内で変数の値が与えられると、グラフパターンがデータセットにマッチしないかどうかをテストします。次のクエリは、データ内に `?person foaf:name ?name` が出現しないかどうかをテストします。

```
PREFIX  rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>
```

```
SELECT ?person
WHERE
{
    ?person rdf:type  foaf:Person .
    FILTER NOT EXISTS { ?person foaf:name ?name }
}

=>
    person
<http://example.org/bob>
```

<http://example.org/bob> のグラフパターンには、?person に述語 foaf:name がないため、そのクエリは「Bob」をこのクエリの結果として返します。NOT EXISTS フィルタは、追加バインドは生成しません。

6.1.10.3 MINUS

SPARQL の否定の、他のタイプは、MINUS があります。これは、その両方の引数を評価してから、パターンの右側のソリューションとは互換性のない、左側のソリューションを計算します。

この例では、次のデータを追加します。

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

:alice  foaf:givenName "Alice" ;
        foaf:familyName "Smith" .

:bob    foaf:givenName "Bob" ;
        foaf:familyName "Jones" .

:carol  foaf:givenName "Carol" ;
        foaf:familyName "Smith" .
```

次のクエリは、データ内で ?s foaf:givenName "Bob" とはマッチしないパターンを探し、その結果を返します。

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

SELECT DISTINCT ?s
WHERE {
   ?s ?p ?o .
   MINUS {
      ?s foaf:givenName "Bob" .
```

```
    }
}

=>
<http://example/carol>
<http://example/alice>
```

結果は「Carol」と「Alice」です。

フィルタ NOT EXISTS と MINUS は、否定の 2 つのアプローチ方法を表します。NOT EXISTS によるアプローチでは、クエリパターンによって特定されるバインドに基づいて、データ内にパターンが存在するかどうかをテストします。MINUS によるアプローチでは、2 つのパターンの評価に基づいてマッチするものを削除します。場合によっては、これらのアプローチは異なる結果を生成することがあります。MINUS フィルタは、追加バインドは生成しません。

6.1.10.4 NOT EXISTS と MINUS の違い

フィルタ式 NOT EXISTS と MINUS は、否定の 2 つの使用方法を表します。NOT EXISTS フィルタは、クエリパターンによってすでに特定されたバインドが与えられると、データ内にパターンが存在するかどうかをテストします。MINUS フィルタは、クエリ内の 2 つのパターンの評価に基づいて、マッチするものを結果セットから削除します。場合によっては、これらの 2 つのアプローチは異なる答えを生成することがあります。

例：変数の共有

例えば、次のデータセットがあったとします。

```
@prefix : <http://example.com/> .
:a :b :c .
```

そして、次のクエリを使用するとします。

```
SELECT *
{
  ?s ?p ?o
  FILTER NOT EXISTS {?x ?y ?x}
}

=>
(This query has no results.)
```

結果セットは空になります。データ内に {?x ?y ?x} とマッチするものがなく、NOT EXISTS フィルタによってすべての結果が除外されるためです。

同じクエリで MINUS を使用すると、1 番目の部分 ?s ?p ?o と 2 番目の部分 ?x ?y ?z の間に共有変数がないため、除外されるバインドはありません。

```
SELECT *
{
  ?s ?p ?o
  FILTER MINUS {?x ?y ?x}
}

=>
s                     p                     o
<http://example.com/a> <http://example.com/b>
<http://example.com/c>
```

例：固定パターン

NOT EXISTS と MINUS で結果が異なる他の例としては、例のクエリ内に具体的なパターンがある（変数がない）場合があります。

次のクエリは、NOT EXISTS を否定のフィルタとして使用しています。

```
PREFIX : <http://example.com/>
SELECT *
{
 ?s ?p ?o
 FILTER NOT EXISTS {:a :b :c}
}

=>
(This query has no results.)
```

次のクエリは、MINUS を否定のフィルタとして使用しています。

```
PREFIX : <http://example.com/>
SELECT *
{
 ?s ?p ?o
 MINUS {:a :b :c}
}

=>
s                     p                     o
<http://example.com/a> <http://example.com/b>
<http://example.com/c>
```

バインドのマッチがないため、除外されるソリューションはなく、ソリューションには「a」、「b」および「c」が含まれます。

例：内部フィルタ

フィルタではグループからの変数がスコープに入ることが理由でも、結果の差異が発生します。この例では、NOT EXISTS 内の FILTER は、検討されるソリューションの ?n の値にアクセスできます。この例では、次のデータセットを使用します。

```
PREFIX : <http://example.com/>
:a :p 1 .
:a :q 1 .
:a :q 2 .

:b :p 3.0 .
:b :q 4.0 .
:b :q 5.0 .
```

FILTER NOT EXISTS を使用すると、次のクエリでは、?x :p ?n の考えられる各ソリューションに対してテストが行われます。

```
PREFIX : <http://example.com/>
SELECT * WHERE {
 ?x :p ?n
 FILTER NOT EXISTS {
     ?x :q :m .
     FILTER (?n = ?m)
     }
 }

=>
x                       n
<http://example.com/b> 3.0
```

MINUS を使用した場合、パターン内の FILTER は ?n の値を持たないため、常にバインドなしになります。

```
PREFIX : <http://example.com/>
SELECT * WHERE {
    ?x ?p  ?n
    MINUS {
          ?x :q ?m .
          FILTER (?n = ?m)
    }
 }

=>
x                       n
<http://example.com/b> 3.0
<http://example.com/a> 1
```

6.1.10.5 否定を含む複合クエリ

複合クエリは、ドキュメントに組み込まれたトリプルに対して動作します。クエリは、ドキュメントと、そのドキュメントに組み込まれたトリプル（存在する場合）の両方を検索します。FILTER キーワードとともに否定を追加して、クエリの結果を制約できます。

例えば以下のようになります。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

let $query := '
  PREFIX foaf: <http://xmlns.com/foaf/0.1/>
  SELECT ?country
  WHERE {
    <http://example.org/news/Nixon>
<http://example.org/wentTo> ?country
  FILTER NOT EXISTS {?country foaf:isIn ?location .
                ?location foaf:isIn "Europe"} .}'
let $store := sem:store((),cts:and-query( (
    cts:path-range-query( "//triples-context/confidence",
    ">=", 80) ,
    cts:path-range-query( "//triples-context/
    pub-date",    ">",
xs:date("1974-01-01")),
    cts:or-query( (
      cts:element-value-query( xs:QName("source"),
      "AP Newswire" ),
      cts:element-value-query( xs:QName("source"), "BBC" )
   )))))

let $result := sem:sparql($query, (), (), $store)
return <result>{$result}</result>
```

注： cts:path-range-query を使用するには、パスインデックスを正しく機能するように設定しておく必要があります。『*Administrator's Guide*』の「Understanding Range Indexes」を参照してください。

これは、前述のクエリを変更したものです。変更後の内容は次のとおりです。「ニクソンが訪問した国に関する情報を持つトリプルを含んでいるすべてのドキュメントを探してください。そのグループの中から、信頼度が 80% 以上で、公開日が 1974 年 1 月 1 日よりも後のトリプルのみを返してください。かつ、ソース要素が「AP Newswire」または「BBC」のトリプルのみを返してください」。MINUS フィルタは、ヨーロッパに位置する国を結果から削除します。

注： SPARQL Update は、ドキュメントに組み込まれたトリプルは変更しません。SPARQL Update を使用して行うことができるのは、新しいトリプルを複合クエリの一部としてグラフに挿入すること、または管理対象トリプルを変更することです。ドキュメント内のトリプルの詳細については、「非管理対象トリプル」（64 ページ）を参照してください。

6.1.10.6 BIND キーワード

BIND キーワードを使用すると、基本グラフパターンまたはプロパティパス式の変数に値を割り当てることができます。BIND を使用すると、先行する基本グラフパターンが終了します。BIND 節に導入されている変数は、BIND で使用される時点までにグループグラフパターンで使用されていてはなりません。パターン内の変数に計算値を割り当てると、その計算値を CONSTRUCT クエリなど他のパターンで使用できるようになります。シンタックスは (expression AS ?var) です。例えば以下のようになります。

```
PREFIX db: <http://dbpedia.org/resource/>
PREFIX onto: <http://dbpedia.org/ontology/>

SELECT ?person
  WHERE { BIND (db:London AS ?location)
          ?person onto:birthPlace ?location .
  }
LIMIT 10
```

6.1.10.7 Values セクション

SPARQL の VALUES セクションを使用すると、クエリ評価の結果と結合される、順番に並べられていないソリューションシーケンスとしてインラインデータを提供できます。VALUES セクションでは、複数の変数をデータブロックで指定できます。

例えば以下のようになります。

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

SELECT *
WHERE { ?a foaf:name ?n .
VALUES ?n { "John" "Jane" }}
```

このクエリの内容は、「foaf:name が John または Jane である主語を探してください」です。データセット内で ?n を検索する代わりに、?n が持つ可能性がある値を指定しています。これは、パラメータリストを丸括弧に入れる、より長い形式を使用したクエリと同じになります。

```
VALUES (?z) { ("John") ("Jane") }
```

注： データの VALUES ブロックは、クエリパターン内や、SELECT クエリまたはサブクエリの末尾に記述できます。

6.1.11 ソリューション修飾子

ソリューション修飾子は、SELECT クエリの結果セットを変更します。このセクションでは、次のソリューション修飾子を使用して、クエリから返される内容を変更する方法について説明します。

- DISTINCT キーワード
- LIMIT キーワード
- ORDER BY キーワード
- OFFSET キーワード
- サブクエリ
- 射影式

注： DISTINCT 以外の修飾子は、WHERE 節の後に記述します。

6.1.11.1 DISTINCT キーワード

結果セットから重複した結果を取り除くときは、DISTINCT キーワードを使用します。

例えば以下のようになります。

```
SELECT DISTINCT ?p
WHERE {?s ?p ?o}
```

このクエリは、persondata データセット内のすべてのトリプルについて、すべての述語を 1 回だけ返します。

p
<http://dbpedia.org/ontology/birthDate>
<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/ontology/deathDate>
<http://dbpedia.org/ontology/deathPlace>
<http://purl.org/dc/elements/1.1/description>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/0.1/givenName>
<http://xmlns.com/foaf/0.1/name>
<http://xmlns.com/foaf/0.1/surname>

6.1.11.2 LIMIT キーワード

表示される SPARQL クエリの結果をさらに制限する場合は、LIMIT キーワードを使用します。例えば、DBpedia データセットには、次のクエリにマッチする数千もの著者が含まれる可能性があります。

```
PREFIX dc:<http://purl.org/dc/elements/1.1/>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>

SELECT ?x ?fn ?ln
WHERE{?x dc:description "Author"@en ;
      foaf:name ?fn ;
      foaf:surname ?ln.}
```

表示するマッチング結果の数を指定するには、WHERE 節の波括弧の後に LIMIT キーワードと整数（変数は不可）を追加します。

例えば以下のようになります。

```
PREFIX dc:<http://purl.org/dc/elements/1.1/>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>

SELECT ?x ?fn ?ln
WHERE{?x dc:description "Author"@en ;
      foaf:name ?fn ;
      foaf:surname ?ln.}
LIMIT 10
```

クエリの結果は、最初の 10 件のマッチに制限されます。

x	fn	
<http://dbpedia.org/resource/A.S._King>	"A.S. King"@en	"King"@en
<http://dbpedia.org/resource/A._H._Almaas>	"A.H. Almaas"@en	"Almaas"@en
<http://dbpedia.org/resource/A._Lee_Martinez>	"A. Lee Martinez"@en	"Martinez"@en
<http://dbpedia.org/resource/A._M._Burrage>	"A. M. Burrage"@en	"Burrage"@en
<http://dbpedia.org/resource/A._Muthukrishnan>	"A. Muthukrishnan"@en	"Muthukrishnan"@
<http://dbpedia.org/resource/Abidemi_Sanusi>	"Abidemi Sanusi"@en	"Sanusi"@en
<http://dbpedia.org/resource/Ada_Albrecht>	"Ada Albrecht"@en	"Albrecht"@en
<http://dbpedia.org/resource/Adèle_Geras>	"Adele Geras"@en	"Geras"@en
<http://dbpedia.org/resource/Agnete_Friis>	"Agnete Friis"@en	"Friis"@en
<http://dbpedia.org/resource/Ahmad_Akbarpour>	"Ahmad Akbarpour"@en	"Akbarpour"@en

6.1.11.3 ORDER BY キーワード

クエリ結果をソートする基準となる 1 つあるいは複数の変数の値を指定する場合は、`ORDER BY` 節を使用します。「SPARQL 1.1 Query Language」勧告で説明されているように、SPARQL では、バインドされていない変数、空白ノード、IRI、または RDF リテラルの順序付けが用意されています。

http://www.w3.org/TR/sparql11-query/#modOrderBy

デフォルトの順序は昇順です。

例えば以下のようになります。

```
PREFIX dc:<http://purl.org/dc/elements/1.1/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?x ?fn ?ln
WHERE{?x dc:description "Author" ;
      foaf:name ?fn ;
      foaf:surname ?ln.}
ORDER BY ?ln ?fn
LIMIT 10
```

結果は、まず著者の姓（`?ln`）の順序、次に著者の名（`?fn`）の順序で並べられます。

x	fn	ln
<http://dbpedia.org/resource/Patience_Abbe>	"Patience Abbe"@en	"Abbe"@en
<http://dbpedia.org/resource/Lynn_Abbey>	"Lynn Abbey"@en	"Abbey"@en
<http://dbpedia.org/resource/George_Abbot_(author)>	"George Abbot"@en	"Abbot"@en
<http://dbpedia.org/resource/Eleanor_Hallowell_Abbott>	"Eleanor Hallowell Abbott"@en	"Abbott"@en
<http://dbpedia.org/resource/Hailey_Abbott>	"Hailey Abbott"@en	"Abbott"@en
<http://dbpedia.org/resource/Walid_Abdallah>	"Walid Abdallah"@en	"Abdallah"@en
<http://dbpedia.org/resource/Robert_Abernathy>	"Robert Abernathy"@en	"Abernathy"@en
<http://dbpedia.org/resource/Susan_Abulhawa>	"Susan Abulhawa"@en	"Abulhawa"@en
<http://dbpedia.org/resource/Rodolfo_Acevedo>	"Rodolfo Acevedo"@en	"Acevedo"@en
<http://dbpedia.org/resource/John_M._Ackerman>	"John M. Ackerman"@en	"Ackerman"@en

結果の順序を降順に変更するには、`DESC` キーワードを使用し、返される値の変数を括弧内に配置します。

例えば以下のようになります。

```
PREFIX dc:<http://purl.org/dc/elements/1.1/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?x ?fn ?ln
WHERE{?x dc:description "Author"@en ;
```

```
      foaf:name ?fn ;
      foaf:surname ?ln .}
ORDER BY DESC (?ln)
LIMIT 10
```

x	fn	ln
<http://dbpedia.org/resource/Joan_de_Hamel>	"Joan de Hamel"@en	"de Hamel"@en
<http://dbpedia.org/resource/Laila_al-Ouhaydib>	"Laila al-Ouhaydib"@en	"al-Ouhaydib"@en
<http://dbpedia.org/resource/Shahidul_Zahir>	"Shahidul Zahir"@en	"Zahir"@en
<http://dbpedia.org/resource/Anwer_Zahidi>	"Anwer Zahidi"@en	"Zahidi"@en
<http://dbpedia.org/resource/Helen_Zahavi>	"Helen Zahavi"@en	"Zahavi"@en
<http://dbpedia.org/resource/Rachel_Zadok>	"Rachel Zadok"@en	"Zadok"@en
<http://dbpedia.org/resource/Michele_Zackheim>	"Michele Zackheim"@en	"Zackheim"@en
<http://dbpedia.org/resource/Shan_Sa>	"Ni Yan"@en	"Yan"@en
<http://dbpedia.org/resource/Evie_Wyld>	"Evie Wyld"@en	"Wyld"@en
<http://dbpedia.org/resource/Patricia_Wrede>	"Patricia C. Wrede"@en	"Wrede"@en

6.1.11.4 OFFSET キーワード

OFFSET 修飾子は、ページネーション（マッチしたクエリ結果を指定された件数だけスキップして残りの結果を返す）を行う場合に使用します。このキーワードを LIMIT および ORDER BY キーワードと組み合わせて使用することで、データセットからデータのさまざまな部分を取得できます。例えば、さまざまな開始箇所から結果のページを作成できます。

次の例では、Author を昇順でクエリし、その結果のリストを位置 8 から開始した最初の 20 件に制限します。

```
PREFIX dc:<http://purl.org/dc/elements/1.1/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?x ?fn ?ln
WHERE{?x dc:description "Author"@en ;
      foaf:name ?fn ;
      foaf:surname ?ln.}
ORDER BY ?x
OFFSET 8
LIMIT 20
```

結果が返され、最初の 8 件のマッチはスキップされます。

x	fn	ln
<http://dbpedia.org/resource/Agnete_Friis>	"Agnete Friis"@en	"Friis"@en
<http://dbpedia.org/resource/Ahmad_Akbarpour>	"Ahmad Akbarpour"@en	"Akbarpour"@en
<http://dbpedia.org/resource/Ajip_Rosidi>	"Ajip Rosidi"@en	"Rosidi"@en
<http://dbpedia.org/resource/Alauddin_Masood>	"Alauddin Masood"@en	"Masood"@en
<http://dbpedia.org/resource/Alberto_Fortis>	"Abbe Alberto Fortis"@en	"Fortis"@en
<http://dbpedia.org/resource/Alexandra_Hawkins>	"Alexandra Hawkins"@en	"Hawkins"@en
<http://dbpedia.org/resource/Alexandre_Beljame>	"Alexandre Beljame"@en	"Beljame"@en
<http://dbpedia.org/resource/Alexis_Jenni>	"Alexis Jenni"@en	"Jenni"@en
<http://dbpedia.org/resource/Alexis_Lecaye>	"Alexis Lecaye"@en	"Lecaye"@en
<http://dbpedia.org/resource/Alfred_Leland_Crabb>	"Alfred Leland Crabb"@en	"Crabb"@en

注： SPARQL は、1 ベースのインデックスを使用します。つまり、1 番目の項目は 1 であり、0 ではありません。このため、オフセットを 8 とすると、1 から 8 までの項目がスキップされます。

6.1.11.5 サブクエリ

サブクエリを使用すると、複数のクエリの結果を結合できます。また、1 つあるいは複数のクエリを別のクエリの中で入れ子にできます。サブクエリは、それぞれが波括弧の対で囲まれています。通常、サブクエリは、ソリューション修飾子と併用します。次の例では、出生地が「London」の「Politician」（政治家）をクエリして探し、結果を最初の 10 件に制限します。

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX db: <http://dbpedia.org/resource/>
PREFIX onto: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT  ?name ?location ?date
WHERE
   { ?person dc:description "Politician"@en .

    {SELECT ?location
    WHERE{?person onto:birthPlace db:London .
          ?person onto:birthPlace ?location }
    }
    {SELECT ?date
    WHERE{?person onto:birthDate ?date .}
    }
    {SELECT ?name
    WHERE{ ?person foaf:name ?name }
    }
   }
LIMIT 10
```

6.1.11.6 射影式

SPARQL の SELECT クエリ内で射影式を使用すると、バインドされた変数だけではなく、任意の SPARQL 式を射影できます。これにより、クエリ内で新しい値を作成できます。

このタイプのクエリでは、クエリの結果セットのカラムとして、変数、定数 IRI、定数リテラル、関数呼び出し、またはその他の式から生成される値を SELECT で使用します。

> **注：** 関数には、SPARQL のビルトイン関数と実装でサポートされている拡張関数の両方を含めることができます。

射影式は、丸括弧で囲む必要があります。また、AS キーワードを使用してエイリアスを指定する必要があります。シンタックスは (expression AS ?var) です。

次に例を示します。

```
PREFIX ex: <http://example.org/>

SELECT ?Item (?price * ?qty AS ?total_price)
WHERE {
  ?Item ex:price ?price.
  ?Item ex:quantity ?qty
}
```

このクエリは、RDF データセットに含まれるグラフ内では発生していない ?total_price の値を返します。

6.1.12 SPARQL の結果の重複を解消

MarkLogic は、sem:sparql() に dedup-on オプションと dedup-off オプションを実装しています。シンプルな sem:sparql() の例に基づいて、重複の意味と動作について示します。

最初に、同じトリプルを 2 回挿入します。

```
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

(: load an rdf triple that will match the SPARQL query :)

sem:rdf-insert(
 sem:triple(sem:iri("http://www.example.org/dept/108/
 invoices/20963"),
```

```
  sem:iri("http://www.example.org/dept/108/invoices/paid"),
  "true") ,
  xdmp:default-permissions(),
  "test-dedup") ;

(: returns the URI of the document that contains the triple :)

import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

(: load an rdf triple that will match the SPARQL query :)

sem:rdf-insert(
  sem:triple(sem:iri("http://www.example.org/dept/108/
  invoices/20963"),
  sem:iri("http://www.example.org/dept/108/invoices/paid"),
  "true") ,
  xdmp:default-permissions(),
  "test-dedup") ;

(: returns the URI of the document that contains the triple :)
```

次に、dedup-off を含む SPARQL クエリを使用します。

```
sem:sparql('
PREFIX inv: <http://www.example.org/dept/108/invoices/>

SELECT ?predicate ?object
WHERE
{ inv:20963 ?predicate ?object }
' ,
(),
"dedup-off" )
=>
<http://www.example.org/dept/108/invoices/paid> "true"
<http://www.example.org/dept/108/invoices/paid> "true"
```

まったく同一の 2 つのトリプルが返されます。次の SPARQL クエリは、dedup-on を使用しています。これがデフォルトです。

```
sem:sparql('
PREFIX inv: <http://www.example.org/dept/108/invoices/>

SELECT ?predicate ?object
WHERE { inv:20963 ?predicate ?object }
' ,
```

```
(),
 "dedup-on" )
=>
<http://www.example.org/dept/108/invoices/paid> "true"
```

1 つのトリプルインスタンスのみが返されます。

dedup-on オプションがデフォルトで、標準準拠の動作になります。重複するトリプルを挿入することがない場合、sem:sparql の dedup-off オプションで同じ結果が返されるのは当然ですが、検索でフィルタリングする場合などかなりのパフォーマンスオーバーヘッドが伴うため、このオプションの使用については慎重に検討してください。

6.1.13 プロパティパス式

プロパティパスを使用すると、RDF グラフをトラバースできます。2 つのグラフノード間の可能なルートを使ってグラフをたどることができます。プロパティパスを使用すると、例えば「ジョンに繋がる人全員とジョンを知っている人を知っている人全員を表示する」といった質問に答えることができます。プロパティパスでは、XPath に似たシンタックスを使用して、データセットグラフ内の任意の長さのパスをクエリできます。プロパティパスのクエリは、ノードをリンクするパスがプロパティパスにマッチしている、接続するノードのペアを取得します。このため、トリプルとして表現されている関係性をたどること、および使用することが容易になります。

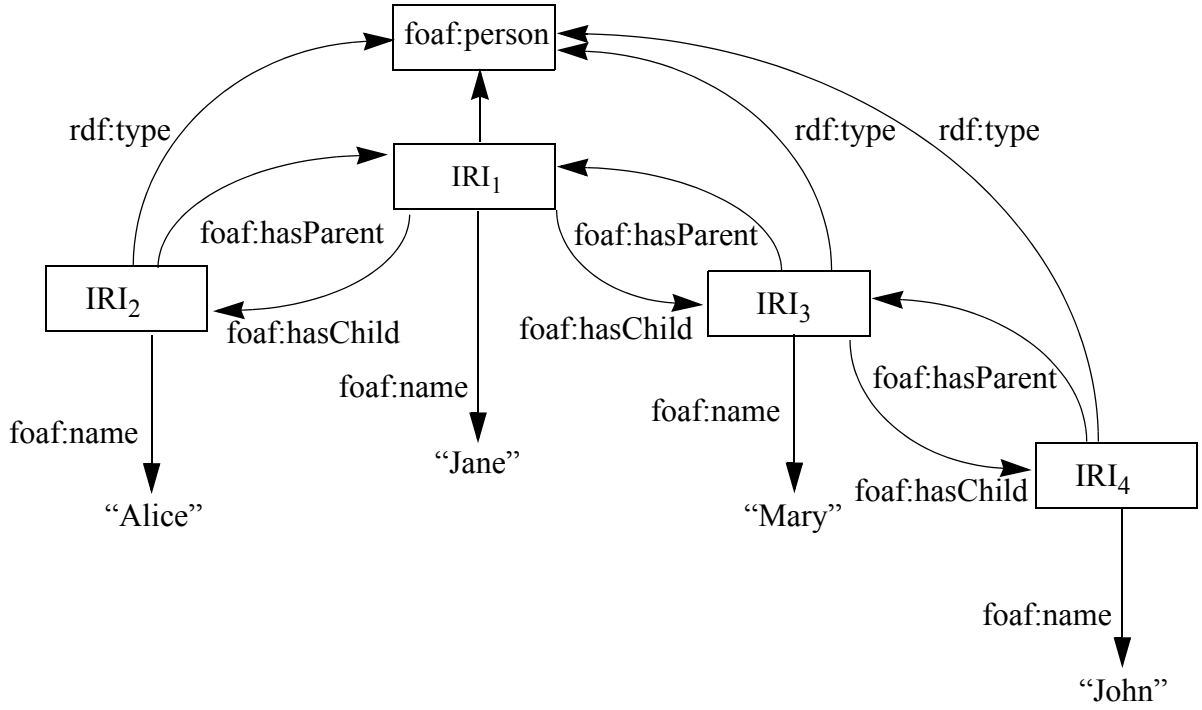
クエリを評価すると、パス式のすべてのマッチが特定され、主語または目的語が適宜バインドされます。記録されるマッチは、グラフを通じたルートごとに 1 つだけです。そのため、任意のパス式に対する重複はありません。

6.1.13.1 列挙プロパティパス

次の表は、サポートされている列挙パス演算子（|、^、および /）の中で、プロパティパスの述語と結合できるものについて説明したものです。

プロパティパス	構成	説明
シーケンス	path1/path2	path1 から path2 へパスを前進させます。
反転	^path	目的語から主語へパスを後進させます。
代替	path1|path2	path1 または path2 のいずれか
グループ	(path)	グループパス path。括弧で優先順位を制御します。

次の例は、このシンプルなグラフモデルを使用したプロパティパスを示しています。



同じグラフモデルを Turtle 形式のトリプルとして表現すると、次のようになります。

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns> .
@prefix p0: <http://marklogic.com/semantics/> .

p0:alice foaf:hasParent p0:jane ;
  a foaf:Person ;
  foaf:name "Alice" .

p0:jane foaf:hasChild p0:alice,
    p0:mary;
  a foaf:Person ;
  foaf:name "Jane" .

p0:mary foaf:hasParent p0:jane ;
  a foaf:Person ;
  foaf:hasChild p0:john ;
  foaf:name "Mary" .

p0:john foaf:hasParent p0:mary ;
  a foaf:Person ;
  foaf:name "John".
```

次のクエリ例では、パス（/ 演算子）を使用して、アリス（Alice）の親の名前を調べます。

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

SELECT ?s ?name
  WHERE {?s foaf:name "Alice".
    ?s foaf:hasParent/foaf:name ?name .
  }

=>
  s       name
<http://marklogic.com/semantics/alice> "Jane"
```

次のクエリでは、ジョン（John）からリンク 2 つ分離れた人々の名前（ジョンの祖父母）を調べます。

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

SELECT ?s ?name
  WHERE {?s foaf:name "John".
    ?s foaf:hasParent/foaf:hasParent/foaf:name ?name .
  }

=>
  s     name
<http://marklogic.com/semantics/john> "Jane"
```

次のクエリでは、プロパティパスの方向を反転して（^ 演算子を使用して主語と目的語の役割を入れ替えて）、メアリー（Mary）の母の名前を調べます。

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

SELECT ?s
  WHERE { <http://marklogic.com/semantics/mary>
^foaf:hasChild ?s }

=>
  s
<http://marklogic.com/semantics/Jane>
```

6.1.13.2 非列挙プロパティパス

非列挙パスを使用すると、トリプルパスをクエリして、関係性とシンプルなファクトを発見できます。次の表は、プロパティパスの述語と結合できる非列挙パス演算子（+、*、または ?）について説明したものです。

プロパティパス	構成	説明
1 つあるいは複数	path+	パス要素の 1 つあるいは複数のマッチによって、パスの主語と目的語を接続するパス。
ゼロあるいは複数	path*	パス要素のゼロあるいは複数のマッチによって、パスの主語と目的語を接続するパス。
ゼロあるいは 1 つ	path?	パス要素のゼロあるいは 1 つのマッチによって、パスの主語と目的語を接続するパス。

注： パス要素自体がパス構造で構成されている場合があります。

反転演算子（^）は、列挙パス演算子と併用できます。このような演算子の優先順位は、グループ内で左から右です。

以降の各例では、sem:rdf-insert を使用して次のトリプルを追加し、foaf:knows の概念を表しています。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

let $string := '
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix p0: <http://marklogic.com/semantics/> .

p0:alice foaf:knows p0:jane .

p0:jane foaf:knows p0:mary,
  p0:alice .

p0:mary foaf:knows p0:john,
  p0:jane .

p0:john foaf:knows p0:mary .'

return sem:rdf-insert(sem:rdf-parse($string, "turtle"))
```

「Mary」とつながっている全員の名前を調べるには、foaf:knows に「+」パス演算子を付けて使用します。

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

SELECT ?s ?name
  WHERE {?s foaf:name "Mary" .
    ?s foaf:knows+/foaf:name ?name .}

=>
  s                                          name
<http://marklogic.com/semantics/mary>    "Jane"
<http://marklogic.com/semantics/mary>    "John"
<http://marklogic.com/semantics/mary>    "Mary"
<http://marklogic.com/semantics/mary>    "Alice"
```

このクエリは、1 個以上のパスが存在する foaf:knows で「Mary」とつながっているすべてのトリプルにマッチします。foaf:knows に「*」演算子を付けると、0 個以上のパスで「Mary」とつながっている全員（Mary を含む）の名前を調べることができます。

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

SELECT ?s ?name
  WHERE {?s foaf:name "Mary" .
    ?s foaf:knows*/foaf:name ?name .}
```

この場合、結果は前の例と同じになります。0 個以上のパス（「*」パス演算子）で「Mary」とつながっている人の数は、1 個以上のパスでつながっている人の数と同じためです。

「?」演算子を使用すると、1 個のパス要素で「Mary」とつながっているトリプルが探されます。

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

SELECT ?s ?name
  WHERE {?s foaf:name "Mary" .
    ?s foaf:knows?/foaf:name ?name .}

=>
  s                                        name
<http://marklogic.com/semantics/mary>  "Jane"
<http://marklogic.com/semantics/mary>  "John"
<http://marklogic.com/semantics/mary>  "Mary"
```

また、プロパティパスのシーケンスを使用してトリプル間のつながりを発見することもできます。

例えば、次のクエリは 3 つのパス要素で「Mary」とつながっているトリプルを探します。

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

SELECT ?s ?name
WHERE {
?s foaf:name "Mary" .
?s foaf:knows/foaf:knows/foaf:knows/foaf:name ?name .
}

 s                                                 name
<http://marklogic.com/semantics/mary>    "John"
<http://marklogic.com/semantics/mary>    "Jane"
<http://marklogic.com/semantics/mary>    "John"
<http://marklogic.com/semantics/mary>    "Jane"
```

結果に重複があるのは、異なるパスをクエリがトラバースしたためです。SELECT 節で DISTINCT キーワードを追加すると、それぞれの結果に対してインスタンスが 1 つのみ返され、重複が除外されます。

プロパティパスを使用する SPARQL クエリと cts:query パラメータを結合して、一部のドキュメントのみに結果を制限できます（複合クエリ）。

次の複合クエリは、「Alice」につながっているすべての人の中から、子供がいる人を探します。

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

SELECT ?s ?name
WHERE {
  ?s foaf:name "Mary" .
  ?s foaf:knows+/foaf:name ?name .
    ?s ?p ?o .
FILTER cts:contains(?p, cts:word-
query("http://xmlns.com/foaf/0.1/hasChild"))
       }
=>
<http://marklogic.com/semantics/mary> "Alice"
<http://marklogic.com/semantics/mary> "Jane"
```

```
<http://marklogic.com/semantics/mary> "John"
<http://marklogic.com/semantics/mary> "Mary"
```

また、cts:query パラメータを使用して、クエリをコレクションまたはディレクトリに制限することもできます。

6.1.13.3 推論

非列挙パスを使用すると、シソーラス関係を使用したシンプルな推論を実行できます。（シソーラス関係（thesaural relationship）は、シンプルなオントロジーです）。例えば、次のパターンを使用すると、リソースのスーパータイプなど、可能性があるすべてのリソースタイプを推論できます。

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?x ?type
{
   ?x rdf:type/rdfs:subClassOf* ?type
}
```

例えば、次のクエリはサブクラスが「shirt」の製品を探します。

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex: <http://example.com>

SELECT ?product
WHERE
  {
    ?product rdf:type/rdfs:subClassOf* ex:Shirt ;
}
```

推論の詳細については、「推論」（147 ページ）を参照してください。

6.1.14 SPARQL 集約

SPARQL 集計関数を使用すると、トリプルに対してシンプルな分析的クエリを実行できます。集計関数は、トリプル内の値または値共起に対して操作を実行します。

例えば、集計関数を使用して、値の合計を計算できます。次の SPARQL クエリは、SUM を使用して総売上を調べます。

```
PREFIX demov: <http://demo/verb/>
PREFIX vcard: <http://www.w3.org/2006/vcard/ns/>
```

```
SELECT (SUM (?sales) as ?sum_sales)
FROM <http://marklogic.com/semantics/COMPANIES100/>
WHERE {
?company a vcard:Organization .
?company demov:sales ?sales
}
```

サポートされている SPARQL 集計関数は次に示すとおりです。

集計関数	例
COUNT	SELECT (COUNT (?company) as ?count_companies) Count of “companies”
SUM	SELECT (SUM (?sales) as ?sum_sales)
MIN	SELECT (MIN (?sales) as ?min_sales)
MAX	SELECT ?country (MAX (?sales) AS ?max_sales)
AVG	SELECT ?industry (ROUND(AVG (?employees)) AS ?avg_employees)
グループ化操作：	GROUP BY では、すべての集計関数がサポートされています。
GROUP BY	GROUP BY ?industry または GROUP BY ?country ?industry
GROUP BY <some_aggregate_ variable>	GROUP BY AVG
GROUP BY..HAVING <some_aggregate_ variable>	GROUP BY ?industry HAVING (?sum_sales > 3000000000)
GROUP CONCAT<more_than_one _item>	SELECT ?region (GROUP_CONCAT(DISTINCT ?industry ; separator=" + ") AS ?industries)

集計関数	例
SAMPLE	SELECT ?country (SAMPLE(?industry) AS ?sample_industry) (SUM (?sales) AS ?sum_sales) 非集約変数を適切に評価するには、SAMPLE が必要です。

次の例は、大量のトリプルに対して集計関数 COUNT を使用する SPARQL クエリを示したものです。

```
PREFIX demor: <http://demo/resource/>
PREFIX demov: <http://demo/verb/>
PREFIX vcard: <http://www.w3.org/2006/vcard/ns/>

# count the companies
# (more precisely, count things of type organization)

(SELECT ( COUNT (?company) AS ?count_companies )

FROM <http://marklogic.com/semantics/test/COMPANIES100/>
WHERE {
  ?company a vcard:Organization .

}=>
100
```

次に、COUNT および ORDER BY DESC を使用する別の例を示します。

```
PREFIX demor: <http://demo/resource/>
PREFIX demov: <http://demo/verb/>
PREFIX vcard: <http://www.w3.org/2006/vcard/ns/>

SELECT DISTINCT ?object (COUNT(?subject) AS ?count)
WHERE {
  ?subject <http://www.w3.org/1999/02/22-rdf-syntax-
ns#type/> ?object
}
ORDER BY DESC (?count)
LIMIT 10
```

次のクエリは、集約（MAX）を使用して、背番号が最も大きい野球選手を調べ、その選手に関するすべてのトリプルを取得します。データセット内のすべての選手が持っていることが判明している任意のトリプル（bb:number）を使用し、主語を ?key に格納します。その後、すべてのトリプルをクエリし、外側のクエリの主語が ?key 値にマッチするものをフィルタリングします。

```
PREFIX bb:  <http://marklogic.com/baseball/players/>
PEFIX bbr:  <http://marklogic.com/baseball/rules/>
PREFIX xs: <http://www.w3.org/2001/XMLSchema#>

SELECT *
FROM <Athletics>
{
  ?s ?p ?o .
  {
     SELECT(MAX(?s1) as ?key)
     WHERE
     {
       ?s1 bb:number ?o1 .
     }
  }
  FILTER (?s = ?key)
}
ORDER BY ?p
```

次のクエリは複雑な入れ子になっており、COUNT AVG を使用して、特定の製品タイプのベンダーのうち、価格の低いほうから 10 件を調べます。製品タイプは、平均コストを下回る製品のうち割合が最も高いものが選択されます。その後、「name1」または「name2」が含まれるベンダーをフィルタリングします。

```
PREFIX bsbm: <http://www4.wiwiss.fu-
berlin.de/bizer/bsbm/v01/vocabulary/>
PREFIX bsbm-inst: <http://www4.wiwiss.fu-berlin.de/
bizer/bsbm/v01/instances/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX cts: <http://marklogic.com/cts#>

SELECT ?vendor (xsd:float(?belowAvg)/?offerCount As
?cheapExpensiveRatio)
 {
  { SELECT ?vendor (count(?offer) As ?belowAvg)

     {
    { ?product a <http://www4.wiwiss.fu-berlin.de/bizer/
bsbm/v01/instances/ProductType459> .
    ?offer bsbm:product ?product .
```

```
    ?offer bsbm:vendor ?vendor .
    ?offer bsbm:price ?price .
    { SELECT ?product (avg(xsd:float(xsd:string(?price)))
As ?avgPrice)
     {
      ?product a <http://www4.wiwiss.fu-berlin.de/
bizer/bsbm/v01/
        instances/ProductType459> .
      ?offer bsbm:product ?product .
      ?offer bsbm:vendor ?vendor .
      ?offer bsbm:price ?price .
     }
     GROUP BB ?product
    }
   } .
  FILTER (xsd:float(xsd:string(?price)) < ?avgPrice)
  }
   GROUP BY ?vendor
  }
  { SELECT ?vendor (count(?offer) As ?offerCount)
   {
    ?product a <http://www4.wiwiss.fu-berlin.de/
bizer/bsbm/v01/
      instances/ProductType459> .
    ?offer bsbm:product ?product .
    ?offer bsbm:vendor ?vendor .
   }
   GROUP BY ?vendor
  }
  FILTER cts:contains(?vendor, cts:or-query(("name1",
"name2")))
}
ORDER BY desc(xsd:float(?belowAvg)/?offerCount) ?vendor
LIMIT 10
```

6.1.15 sem:sparql の結果の使い方

クエリ内での `sem:sparql` の結果の使い方の例を示します。

```
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

cts:search(
  fn:doc(),
  cts:triple-range-query(
    (), "takenIn",
```

```
    (: Use sem:sparql to run a query, then use the !
operator to
     : convert the solution to a sequence of strings
     :)
    sem:sparql(
      'select ?countryIRI
       {
         ?continentIRI
<http://www.w3.org/2004/02/skos/core#prefLabel>
?continentLabel .
         ?countryIRI
<http://dbpedia.org/property/continent> ?continentIRI .
       }',
     map:entry("continentLabel", rdf:langString("Countries
in South America", "en"))
    ) ! map:get(., "countryIRI")
  ))
```

6.1.16 SPARQL のリソース

SPARQL 勧告は、次の各規格と密接に関連しています。

- SPARQL Protocol for RDF（SPROT）規格では、SPARQL クエリの発行と結果の受け取りに関するリモートプロトコルを定義しています。
  http://www.w3.org/TR/rdf-sparql-protocol/
- MarkLogic では、W3C 勧告で説明されている単純含意をサポートしています。
  http://www.w3.org/TR/rdf-mt/#entail
- SPARQL Query Results XML Format 規格では、SPARQL の `SELECT` および `ASK` クエリの結果を表現するための XML ドキュメント形式を定義しています。
  http://www.w3.org/TR/rdf-sparql-XMLres/
- SPARQL 1.1 Graph Store HTTP Protocol：
  http://www.w3.org/TR/2012/CR-sparql11-http-rdf-update-20121108/

SPARQL クエリ言語を詳しく学習する場合に利用できる各種チュートリアルが用意されています。例えば以下のようになります。

- http://www.cambridgesemantics.com/semantic-university

次の書籍を読むことをお勧めします。

- Bob DuCharme 著『*Learning SPARQL*』（発行元：O'Reilly）
- Dean Allemang および Jim Hendler 著『*Semantic Web for the Working Ontologist*』（発行元：Morgan Kaufmann）

その他に、次のリソースが役立ちます。

- 「SPARQL Implementations」: http://www.w3.org/wiki/SparqlImplementations
- 「SPARQL Working Group」: http://www.w3.org/2009/sparql/wiki/Main_Page
- 「SPARQL 1.1 Query Results JSON Format」: http://www.w3.org/TR/2012/PR-sparql11-results-json-20121108/
- 「SPARQL Frequently Asked Questions」: http://thefigtrees.net/lee/sw/sparql-faq

6.2 XQuery または JavaScript によるトリプルのクエリ

このセクションでは、XQuery または JavaScript とセマンティックデータを組み合わせた使用例について説明します。JavaScript または XQuery を使用して MarkLogic 内のトリプルをクエリする場合は、セマンティック API ライブラリ、ビルトイン関数、search API ビルトイン関数、またはこれらの組み合わせを使用できます。

このセクションでは、次の内容を取り上げます。

- 例を実行する準備
- セマンティック関数を使用したクエリ
- 変数に対するバインドの使用
- 結果の XML および RDF 表示
- CURIE を扱う
- cts 検索を使用したセマンティックの使用

6.2.1 例を実行する準備

XQuery または JavaScript によるトリプルのクエリ例では、GovTrack データセットを入手していることを前提にしています。独自のデータセットを使用する場合や http://www.govtrack.us/data/rdf/ にアクセスできない場合は、このセクションをスキップしても構いません。

`GovTrack.us` のデータは無料で、米国議会の法案、議員、および投票記録に関する立法府の情報を公的に利用できます。この情報は、政府機関の公式 Web サイトに由来しています。`Govtrack.us` は、オープンデータの原則に基づいて立法府の透明性を実現しています。

GovTrack データセットをインストールする前に、次のものを用意しておく必要があります。

- MarkLogic サーバー 8.0-4 以降

- mlcp（MarkLogic Content Pump）。『*mlcp User Guide*』の「Installation and Configuration」を参照してください。
- GovTrack データセット、および http://www.govtrack.us/data/rdf/ へのアクセス

次の手順に従って、GovTrack データセットをダウンロードし、MarkLogic サーバーにロードします。

1. 次の各ファイルをローカルのファイルシステムのディレクトリにダウンロードします。

   - http://www.govtrack.us/data/rdf/bills.108.cosponsors.rdf.gz
   - http://www.govtrack.us/data/rdf/bills.108.rdf.gz
   - http://www.govtrack.us/data/rdf/people.rdf.gz
   - http://www.govtrack.us/data/rdf/people.roles.rdf.gz

2. `govtrack` データベースおよびフォレストを作成します。ここで説明する各例では、GovTrack データ用にアプリケーションサーバーをポート 8000 で使用できます。このデフォルトサーバーは、XDBC サーバーおよび REST インスタンスとして機能します。

   独自の XDBC サーバーおよび REST インスタンスを作成する場合は、詳細については、このガイドの「追加サーバーの設定」、および『*REST Application Developer's Guide*』の「Administering REST Client API Instances」を参照してください。

3. `govtrack` データベースのトリプルインデックスおよびコレクションレキシコンがオンになっていることを確認します。（これらはデフォルトでオンになっているはずです。）「トリプルインデックスの有効化」（58 ページ）を参照してください。

4. mlcp で `govtrack` にデータをインポートします。このとき、`info:govtrack/people` および `info:govtrack/bills` のコレクションを指定します。「mlcp を使用したトリプルのロード」（37 ページ）を参照してください。Windows でのインポートコマンドは次のようになります。

   ```
   mlcp.bat import -host localhost -port 8000 -username admin ^
     -password password -database govtrack -input_file_type rdf ^
     -input_file_path c:\space\GovTrack -input_compressed true^
     -input_compression_codec gzip ^
     -output_collections "info:govtrack/people,info:govtrack/bills"
   ```

   各自の環境に合わせて `host`、`port`、`username`、`password`、および `-input_file_path` オプションを修正します。この例では、読みやすいように長い行が分割され、Windows の連続文字（「^」）が追加されています。

注： コマンドには、必ず `-database` パラメータを追加してください。このパラメータを省略すると、データはデフォルトの Documents データベースに格納されます。

UNIX での同等なコマンドは次のようになります。

```
mlcp.sh import -host localhost -port 8000 -username admin \
-password password -database govtrack -input_file_type RDF \
-input_file_path /space/GovTrack -input_compressed true \
-input_compression_codec gzip \
-output_collections 'info:govtrack/people,
info:govtrack/bills'
```

上記の例では、長い行が分割され、Unix の連続文字（「\」）が追加されています。

注： 適切なパーサーを呼び出すには、`-input_file_type` に RDF を指定することが重要です。

6.2.2 セマンティック関数を使用したクエリ

SPARQL の `SELECT`、`ASK`、および `CONSTRUCT` クエリは、XQuery の `sem:sparql` および `sem:sparql-values` 関数や、JavaScript の `sem.sparql` および `sem.sparqlValues` 関数を使用して実行できます。関数のシグネチャと説明の詳細については、『*MarkLogic XQuery and XSLT Function Reference*』の「XQuery Library Modules」に掲載されているセマンティックに関するドキュメントを参照してください。

次の例では、`govtrack` データベースのトリプルインデックスに対して SPARQL クエリを実行します。「例を実行する準備」（125 ページ）を参照してください。

注： セマンティック関数にはビルトインのものもそうでないものもあるため、セマンティック API を使用する XQuery モジュールまたは JavaScript モジュールごとにセマンティック API ライブラリをインポートすることをお勧めします。

XQuery を使用する場合、import ステートメントは次のようになります。

```
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";
```

JavaScript では、import ステートメントは次のようになります。

```
var sem = require("/MarkLogic/semantics.xqy");
```

6.2.2.1 sem:sparql

データベース内の RDF データをクエリするには、SPARQL 言語の場合と同様の方法で `sem:sparql` 関数を使用します。`sem:sparql` を使用するには、SPARQL クエリを文字列として関数に渡します。

XQuery を使用する場合、クエリは次のようになります。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

sem:sparql('
PREFIX bill: <http://www.rdfabout.com/rdf/usgov/congress/
108/bills/>
SELECT ?predicate ?object
WHERE { bill:h963 ?predicate ?object }
')
```

JavaScript を使用する場合、クエリは次のようになります。

```
var sem = require("/MarkLogic/semantics.xqy");

sem.sparql( +
'PREFIX bill:
<http://www.rdfabout.com/rdf/usgov/congress/108/bills/>' +
'SELECT ?predicate ?object' +
'WHERE { bill:h963 ?predicate ?object }' )
```

注： JavaScript では、文字列リテラルを複数の行に分けることができません。「+」または「/」を使用して連結する必要があります。

XQuery コードは、シーケンスとして配列を返します。一方、JavaScript コードは、Sequence を返します。詳細については、『*JavaScript Reference Guide*』の「Sequence」を参照してください。

主語が法案番号「h963」であるすべてのトリプルをクエリする例の結果は、次のようになります。

predicate	object
<http://www.rdfabout.com/rdf/schema/usbill/hasTitle>	<-276ac564:14008b22ed1:-3313>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.rdfabout.com/rdf/schema/usbill/HouseBill>
<http://www.rdfabout.com/rdf/schema/usbill/inCommittee>	<http://www.rdfabout.com/rdf/usgov/congress/committees/HouseGovernmentReform>
<http://www.rdfabout.com/rdf/schema/usbill/cosponsor>	<http://www.rdfabout.com/rdf/usgov/congress/people/D000598>
<http://www.rdfabout.com/rdf/schema/usbill/sponsor>	<http://www.rdfabout.com/rdf/usgov/congress/people/F000116>
<http://www.rdfabout.com/rdf/schema/usbill/congress>	"108"
<http://purl.org/dc/terms/created>	"2003-02-27"
<http://www.rdfabout.com/rdf/schema/usbill/number>	"963"
<http://purl.org/dc/elements/1.1/title>	"H.R. 108/963: To redesignate the facility of the United States Postal Service located at 2777 Loga \"Cesar E. Chavez Post Office\"."
<http://purl.org/ontology/bibo/shortTitle>	"H.R. 963: To redesignate the facility of the United States Postal Service located at 2777 Logan A \"Cesar E. Chavez Post Office\"."
<http://www.rdfabout.com/rdf/schema/usbill/title>	"H.R. 963: To redesignate the facility of the United States Postal Service located at 2777 Logan A \"Cesar E. Chavez Post Office\"."
<http://www.w3.org/2000/01/rdf-schema#label>	"H.R. 963: To redesignate the facility of the United States Postal Service located at 2777 Logan A \"Cesar E. Chavez Post Office\". (108th Congress)"
<http://www.rdfabout.com/rdf/schema/usbill/type>	"h"
<http://www.rdfabout.com/rdf/schema/usbill/status>	"introduced"
<http://www.rdfabout.com/rdf/schema/usbill/introduced>	"2003-02-27"^^xs:date

SPARQL クエリの構築の詳細については、「SPARQL クエリの構築」（78 ページ）を参照してください。

SPARQL クエリは FLWOR ステートメント（FLWOR statement）の入力文字列として構築することもできます。次の例では、`let` ステートメントに SPARQL クエリが含まれています。これは SPARQL の `ASK` クエリです。モルモン教（Latter Day Saints）の教徒である男性政治家がいるかどうかを調べます。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

let $sparql := '
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX politico:
<http://www.rdfabout.com/rdf/schema/politico/>
PREFIX govtrack:
<http://www.rdfabout.com/rdf/schema/usgovt/>
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0/>

ASK  { ?x rdf:type  politico:Politician ;
   foaf:religion "Latter Day Saints" ; foaf:gender "male".}
```

```
'
return sem:sparql($sparql)

=>
true
```

6.2.2.2 sem:sparql-values

`sem:sparql-values` 関数を使用すると、バインドのシーケンスで SPARQL クエリが返す内容を制限できます。次の例では、議員 2 人を表現する主語 IRI に値のシーケンスがバインドされています。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

let $bindings := ( map:entry("s",
sem:iri("http://www.rdfabout.com/rdf/usgov/congress/people
/A000069")),
map:entry("s",
sem:iri("http://www.rdfabout.com/rdf/usgov/congress/people
/G000359"))
)
return
sem:sparql-values("select * { ?s ?p ?o }",$bindings)
```

結果は、議員 2 人の値のシーケンスとして返されます。

s	p	o
<http://www.rdfabout.com/rdf/usgov/congress/people/A000069>	<http://www.w3.org/2001/vcard-rdf/3.0#N>	<-276ac564:14008b2
<http://www.rdfabout.com/rdf/usgov/congress/people/A000069>	<http://www.rdfabout.com/rdf/schema/politico/hasRole>	<-7059da6e:1403c27
<http://www.rdfabout.com/rdf/usgov/congress/people/A000069>	<http://www.rdfabout.com/rdf/schema/politico/hasRole>	<-7059da6e:1403c27
<http://www.rdfabout.com/rdf/usgov/congress/people/A000069>	<http://www.rdfabout.com/rdf/schema/politico/hasRole>	<-7059da6e:1403c27
<http://www.rdfabout.com/rdf/usgov/congress/people/A000069>	<http://xmlns.com/foaf/0.1/img>	<http://www.govtrack
<http://www.rdfabout.com/rdf/usgov/congress/people/A000069>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://www.rdfabout
<http://www.rdfabout.com/rdf/usgov/congress/people/A000069>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://xmlns.com/foa
<http://www.rdfabout.com/rdf/usgov/congress/people/A000069>	<http://www.w3.org/2001/vcard-rdf/3.0#BDAY>	"1924-09-11"
<http://www.rdfabout.com/rdf/usgov/congress/people/A000069>	<http://www.rdfabout.com/rdf/schema/usgovt/congressBioGuideID>	"A000069"
<http://www.rdfabout.com/rdf/usgov/congress/people/A000069>	<http://xmlns.com/foaf/0.1/religion>	"Congregationalist"
<http://www.rdfabout.com/rdf/usgov/congress/people/A000069>	<http://www.rdfabout.com/rdf/schema/usgovt/name>	"Daniel Akaka"
<http://www.rdfabout.com/rdf/usgov/congress/people/A000069>	<http://xmlns.com/foaf/0.1/name>	"Daniel Akaka"
<http://www.rdfabout.com/rdf/usgov/congress/people/A000069>	<http://xmlns.com/foaf/0.1/title>	"Sen."

sem:sparql-values 関数は、最も外側にある VALUES ブロックの SPARQL 1.1 機能と同等であると考えることができます。詳細については、「Values セクション」（105 ページ）を参照してください。

SPARQL value クエリで変数を使用する箇所では、sem:sparql-values の引数として外部バインドを渡すことで、変数を固定値に設定できます。「変数に対するバインドの使用」（132 ページ）を参照してください。

6.2.2.3 sem:store

sem:store 関数には、基準のセットが格納されます。この基準のセットは、クエリの一部として sem:sparql や sem:sparql-values、sem:sparql-update に渡して評価を行うトリプルのセットを選択するために使用されます。sem:store に含まれるトリプルは、現在のデータベースのトリプルインデックスからのものであり、sem:store 内のオプションおよび cts:query 引数によって制約されます（例えば、「このクエリとマッチする、ドキュメント内のすべてのトリプル」）。複数の sem:store コンストラクタを指定した場合、すべてのソースからのトリプルがマージされ、一体となってクエリされます。

sem:sparql、sem:sparql-values、または sem:sparql-update のオプションとして sem:store コンストラクタを指定しなかった場合、そのクエリのデフォルトの sem:store（デフォルトデータベースのトリプルインデックス）が使用されます。

6.2.2.4 インメモリでのトリプルの検索

sem:in-memory-store を使用すると、インメモリでトリプルをクエリできます。

例えば以下のようになります。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

let $triples := sem:rdf-parse($string, ("turtle",
"myGraph"))
let $query := '
PREFIX ad: <http://marklogic.com/addressbook/>
PREFIX d:  <http://marklogic.com/id/>

CONSTRUCT{ ?person ?p ?o .}
FROM <myOtherGraph>
WHERE
{
  ?person ad:firstName "Elvis" ;
  ad:lastName  "Presley" ;
  ?p ?o .
```

```
}
'
for $result in sem:sparql($query, (), (), sem:in-memory-
store($triples))
order by sem:triple-object($result)
return <result>{$result}</result>
```

このクエリは、「myGraph」という名前のトリプルのグラフをインメモリで構築します。このグラフには、名が Elvis で姓が Presley という人物が含まれます。これらのトリプルのソースは「myOtherGraph」であり、結果は順序付けられて返されます。

6.2.3 変数に対するバインドの使用

標準の SPARQL の拡張機能により、クエリステートメントのボディで変数に対するバインドを使用できます。SPARQL クエリで変数を使用する箇所では、`sem:sparql` の引数として外部バインドを渡すことで、変数を固定値に設定できます。

変数に対するバインドは、（従来は許容されていなかったシンタックスで）`OFFSET` および `LIMIT` 節の値として使用することもできます。次のクエリ例では、変数に対するバインドを `LIMIT` と `OFFSET` の両方で使用します。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";
import module namespace json =
"http://marklogic.com/xdmp/json"
  at "MarkLogic/json/json.xqy";
declare namespace jbasic =
"http://marklogic.com/xdmp/json/basic";

let $query := '
PREFIX bb: <http://marklogic.com/baseball/players/>

SELECT ?firstname ?lastname ?team
FROM <SportsTeams>
{
 {
  ?s bb:firstname ?firstname .
  ?s bb:lastname ?lastname .
  ?s bb:team ?team .
  ?s bb:position ?position .
FILTER (?position = ?pos)
 }
}
ORDER BY ?lastname
```

```
LIMIT ?lmt
'
let $mymap := map:map()
let $put := map:put($mymap, "pos", "pitcher")
let $put := map:put($mymap, "lmt", "3")
let $triples := sem:sparql($query, $mymap)
let $triples-xml := sem:query-results-serialize($triples,
"xml")
return <results>{$triples-xml}</results>

=>
<results>
 <sparql xmlns="http://www.w3.org/2005/sparql-results/">
  <head>
    <variable name="firstname"></variable>
    <variable name="lastname"></variable>
    <variable name="team"></variable>
  </head>
 <results>
  <result>
   <binding name="firstname">
    <literal datatype="http://www.w3.org/2001/
    XMLSchema#string">
    Fernando</literal>
   </binding>
   <binding name="lastname">
    <literal datatype="http://www.w3.org/2001/
    XMLSchema#string">
    Abad</literal>
   </binding>
   <binding name="team">
    <literal datatype="http://www.w3.org/2001/
    XMLSchema#string">
    Athletics</literal>
   </binding>
  </result>
  <result>
   <binding name="firstname">
    <literal datatype="http://www.w3.org/2001/
    XMLSchema#string">
    Jesse</literal>
   </binding>
   <binding name="lastname">
    <literal datatype="http://www.w3.org/2001/
    XMLSchema#string">
   Chavez</literal>
```

```
    </binding>
    <binding name="team">
     <literal datatype="http://www.w3.org/2001/
     XMLSchema#string">
     Athletics</literal>
    </binding>
   </result>
   <result>
    <binding name="firstname">
     <literal datatype="http://www.w3.org/2001/
     XMLSchema#string">
     Ryan</literal>
    </binding>
    <binding name="lastname">
     <literal datatype="http://www.w3.org/2001/
     XMLSchema#string">
     Cook</literal>
    </binding>
    <binding name="team">
     <literal datatype="http://www.w3.org/2001/
     XMLSchema#string">
     Athletics</literal>
    </binding>
   </result>
  </results>
  </sparql>
 </results>
```

バインドは、SPARQL（sem:sparql）、SPARQL value（sem:sparql-values）、および SPARQL Update（sem:sparql-update）で使用できます。SPARQL Update で使用される変数に対するバインドの例については、「変数のバインド」（189 ページ）を参照してください。

6.2.4 結果の XML および RDF 表示

sem:query-results-serialize および sem:rdf-serialize 関数を使用すると、結果を XML、JSON、または RDF シリアライゼーションで表示できます。

次の例では、sem:sparql クエリは法案番号「1024」の共同提出者を調べ、値シーケンスを sem:query-results-serialize に渡して結果をデフォルトの XML 形式の変数バインドとして返します。

```
xquery version "1.0-ml";
import module namespace sem =
"http://marklogic.com/semantics"
```

```
  at "/MarkLogic/semantics.xqy";

sem:query-results-serialize(sem:sparql('
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX bill: <http://www.rdfabout.com/rdf/schema/usbill/>

SELECT ?bill ?person ?name
WHERE {?bill rdf:type bill:SenateBill ;
             bill:congress "108" ;
             bill:number "1024" ;
             bill:cosponsor ?person .
       ?person foaf:name ?name .}
'))
```

結果は、W3C SPARQL Query Results 形式で返されます。

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
 <head>
  <variable name="bill"/>
  <variable name="person"/>
  <variable name="name"/>
 </head>
 <results>
  <result>
   <binding name="bill">
    <uri>http://www.rdfabout.com/rdf/usgov/congress/108/bills/s1024</uri>
   </binding>
   <binding name="person">
    <uri>http://www.rdfabout.com/rdf/usgov/congress/people/C000880</uri>
   </binding>
   <binding name="name">
    <literal datatype="http://www.w3.org/2001/XMLSchema#string">Michael Crapo</literal>
   </binding>
  </result>
  <result>
   <binding name="bill">
    <uri>http://www.rdfabout.com/rdf/usgov/congress/108/bills/s1024</uri>
   </binding>
   <binding name="person">
    <uri>http://www.rdfabout.com/rdf/usgov/congress/people/C001041</uri>
   </binding>
   <binding name="name">
    <literal datatype="http://www.w3.org/2001/XMLSchema#string">Hillary Clinton</literal>
   </binding>
  </result>
```

同じ結果を JSON シリアライゼーションで表示するには、このクエリの後に形式オプションを追加します。

例えば以下のようになります。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

sem:query-results-serialize(sem:sparql('
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX bill: <http://www.rdfabout.com/rdf/schema/usbill/>

SELECT ?bill ?person ?name
WHERE {?bill rdf:type bill:SenateBill ;
             bill:congress "108" ;
             bill:number "1024" ;
             bill:cosponsor ?person .
      ?person foaf:name ?name .}
'), "json")
```

sem:rdf-serialize 関数を使用する場合は、文字列として返されるトリプルを渡すか、オプションでパーシングのシリアライゼーションオプションを指定できます。

例えば以下のようになります。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

sem:rdf-serialize(
   sem:triple(
      sem:iri(
   "http://www.rdfabout.com/rdf/usgov/congress/people/
D000060"),
      sem:iri("http://www.rdfabout.com/rdf/schema/usgovt/
name"),
            "Archibald Darragh"), "rdfxml")
```

次の表では、出力に使用できるシリアライゼーションのオプションについて説明します。

シリアライゼーション	出力形式
ntriple	`xs:string`
nquad	`xs:string`
turtle	`xs:string`
rdfxml	要素
rdfjson	`json:object`
triplexml	`sem:triple` 要素のシーケンス

結果を表示する別の方法を選択することもできます。「結果のレンダリングの選択」（77 ページ）を参照してください。

6.2.5 CURIE を扱う

CURIE (Compact URI Expression) は、特定のリソースを表す URI の短縮バージョンです。MarkLogic では、SPARQL 言語に組み込まれているものと同様のメカニズムを使用して長い IRI を短縮できます。このセクションの例で示すように、便宜的によく使用されるプレフィックスの定義がビルトインで用意されています。

CURIE は、プレフィックスとリファレンスの 2 つのコンポーネントで構成されています。プレフィックスはコロン（:）でリファレンスと区切られます。例えば、`dc:description` はダブリンコアのプレフィックスであり、リファレンス `http://purl.org/dc/elements/1.1/` は説明です。

最もよく使用されるプレフィックスとそのマッピングは、次に示すとおりです。

```
map:entry("atom", "http://www.w3.org/2005/Atom/"),
map:entry("cc", "http://creativecommons.org/ns/"),
map:entry("dc", "http://purl.org/dc/elements/1.1/"),
map:entry("dcterms", "http://purl.org/dc/terms/"),
map:entry("doap", "http://usefulinc.com/ns/doap/"),
map:entry("foaf", "http://xmlns.com/foaf/0.1/"),
map:entry("media", "http://search.yahoo.com/searchmonkey/
media/"),
map:entry("og", "http://ogp.me/ns/"),
map:entry("owl", "http://www.w3.org/2002/07/owl/"),
map:entry("prov", "http://www.w3.org/ns/prov/"),
map:entry("rdf", "http://www.w3.org/1999/02/22-rdf-syntax-
```

```
ns"),
map:entry("rdfs", "http://www.w3.org/2000/01/rdf-schema/"),
map:entry("result-set",
"http://www.w3.org/2001/sw/DataAccess/tests/result-set/"),
map:entry("rss", "http://purl.org/rss/1.0/"),
map:entry("skos", "http://www.w3.org/2004/02/skos/core/"),
map:entry("vcard", "http://www.w3.org/2006/vcard/ns/"),
map:entry("void", "http://rdfs.org/ns/void/"),
map:entry("xhtml", "http://www.w3.org/1999/xhtml/"),
map:entry("xs","http://www.w3.org/2001/XMLSchema#")
```

sem:curie-expand および sem:curie-shorten 関数を使用することで、MarkLogic で CURIE を扱うことができます。sem:curie-expand を使用すると、よく使用されるプレフィックスを宣言する必要がなくなります。

例えば以下のようになります。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

sem:curie-expand("foaf:name")

=>

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

この例では、cts:triple-range-query は「Lamar Alexander」という名前の人物を探します。結果は cts:search から返され、foaf:name が「Lamar Alexander」に等しい sem:triple 要素が検索される点に注意してください。述語 CURIE は foaf:name について完全に展開された IRI として表示されます。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

let $query := cts:triple-range-query((), sem:curie-
expand("foaf:name"), "Lamar Alexander", "sameTerm")

return cts:search(fn:collection()//sem:triple, $query)

=>
<sem:triples xmlns="http://marklogic.com/semantics">
   <sem:subject>
   http://www.rdfabout.com/rdf/usgov/congress/people/A000360/
```

```
   </sem:subject>
   <sem:predicate>
      http://xmlns.com/foaf/0.1/name
   </sem:predicate>
   <sem:object
datatype="http://www.w3.org/2001/XMLSchema#string">
      Lamar Alexander
   </sem:object>
</sem:triples>
```

次の例では、クエリには一連の `cts:triples` 関数呼び出しおよび `sem:curie-expand` が含まれ、1917 年 11 月 20 に誕生した議員の名前を調べます。該当する人物の名前は、返されるトリプルステートメントの目的語位置（`sem:triple-object`）から RDF リテラル文字列として返されます。

```
xquery version "1.0-ml";

import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

let $person-triples := cts:triples((), sem:curie-
expand("vcard3:BDAY",
map:entry("vcard3", "http://www.w3.org/2001/
vcard-rdf/3.0/")),
"1917-11-20")
let $subject := sem:triple-subject($person-triples)
let $name-triples := cts:triples($subject,
sem:curie-expand("foaf:name"), ())
let $name := sem:triple-object($name-triples)
return ($name)

=>

Robert Byrd
```

IRI を CURIE に短縮するには `sem:curie-shorten` を使用します。この関数の評価では、プレフィックスで表現される値とコロンよりも後の部分（リファレンス）を連結したもので CURIE を置換します。

例えば以下のようになります。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";
```

```
sem:curie-shorten(sem:iri("http://www.w3.org/1999/02/
    22-rdf-syntax-ns#resource/"))

=>
rdf:resource
```

注： CURIE は IRI にマッピングされますが、属性の値や、IRI のみが含まれるように指定されているその他のコンテンツの値として使用しないでください。

例えば次のクエリでは、`cts:triple-range-query` はその位置に文字列である `sem:curie-shorten` ではなく IRI（`sem:iri`）があることを前提にしているため、空のシーケンスを返します。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

let $query := cts:triple-range-query((),
   sem:curie-shorten(sem:iri("http://xmlns.com/
   foaf/0.1/name")), "Lamar Alexander", "sameTerm")

return cts:search(fn:collection()//sem:triple, $query)
```

代わりに、次のいずれかを使用できます。

```
let $query := cts:triple-range-query((),
   sem:curie-expand("foaf:name"), "Lamar Alexander",
   "sameTerm")
```

または、プレフィックスを完全な IRI に展開します。

```
let $query := cts:triple-range-query((),
   sem:iri("http://xmlns.com/foaf/0.1/name/"), "Lamar
    Alexander","sameTerm")
```

注： SPARQL で定義されている `sameTerm` 関数は、値等価性演算を実行します。これは、型の処理方法が等価演算子（`=`）と異なります。MarkLogic では、`sameTerm` が `=` と異なるのは、型およびタイムゾーンだけです。例えば、`sameTerm(A,B)` は A=B を意味します。SPARQL 用語では `sameTerm` セマンティックを使用して SPARQL クエリ内のグラフパターンにグラフをマッチさせることを単純含意と呼びます。詳細については、「トリプル値とタイプ情報」（55 ページ）を参照してください。

6.2.6 cts 検索を使用したセマンティックの使用

このセクションでは、cts 検索を使用して MarkLogic トリプルストアからの RDF データを返す方法について説明します。次のトピックから構成されます。

- cts:triples
- cts:triple-range-query
- cts:search
- cts:contains

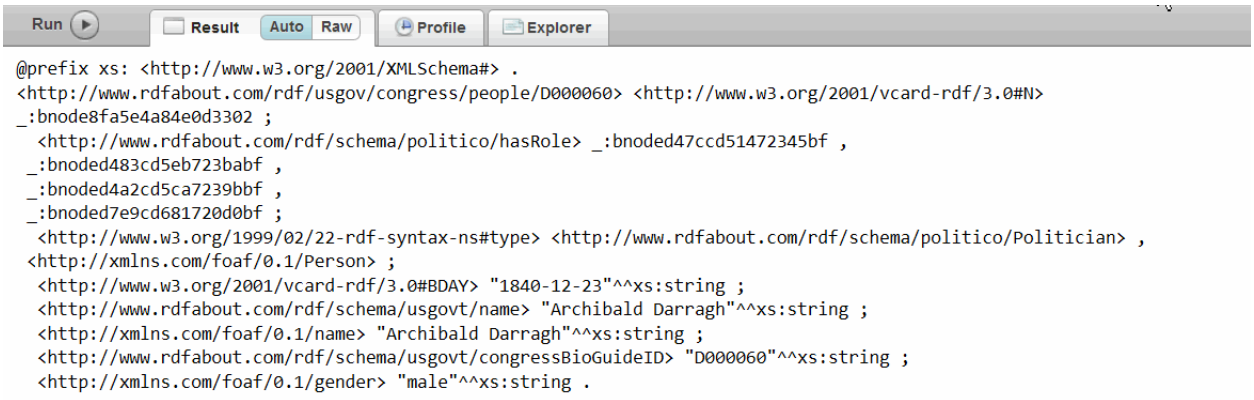
6.2.6.1 cts:triples

`cts:triples` 関数は、トリプルインデックスからパラメータ値を取得します。トリプルは、トリプルインデックス内に存在する任意のソート順で返すことができます。

この例では、議員の主語 IRI が主語 IRI の 1 番目のパラメータとして渡されます。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

let $r := cts:triples
  (sem:iri(

"http://www.rdfabout.com/rdf/usgov/congress/people/D000060
"),
   )

return ($r)
```

マッチング結果は、その議員（Archibald Darragh）のトリプルを返します。



6.2.6.2 cts:triple-range-query

トリプルインデックスへのアクセスは、cts:triple-range-query 関数を通じて提供されます。この例の 1 番目のパラメータは、主語の空のシーケンスです。「Lamar Alexander」という名前の人物を検索するため、述語および目的語の各パラメータが sameTerm 演算子とともに指定されています。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

let $query := cts:triple-range-query((),
  sem:iri("http://xmlns.com/foaf/0.1/name"), "Lamar
Alexander", "sameTerm")

return cts:search(fn:collection()//sem:triple, $query)
```

6.2.6.3 cts:search

ビルトインの cts 検索関数は、テキスト検索を実行するために使用する XQuery 関数です。次の例では、cts:search は XML ドキュメントの info:govtrack/bills コレクションに対してクエリを実行し、ドキュメントに「Guam」という語が含まれる法案の数を特定しています（指定された文字列の cts:word-query）。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

let $search := cts:search(//sem:triple,
  cts:and-query((cts:collection-query("info:govtrack/bills"),
cts:word-query("Guam"))
  )
 )[1]

return cts:remainder($search)

=>
16
```

cts:query と比較演算子の組み合わせを使用できます。次の例の cts:triple-range-query 関数は、cts:search 内で使用されています。これにより、foaf:name が「Lamar Alexander」に等しいか、Alexander の主語 IRI に画像 IRI を提供する foaf:img プロパティが含まれる sem:triple 要素を探します。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

declare namespace dc = "http://purl.org/dc/elements/1.1/";

cts:search(collection()//sem:triple, cts:or-query((
  cts:triple-range-query((), sem:curie-expand("foaf:name"),
   "Lamar Alexander", "sameTerm"),
  cts:triple-range-query(
   sem:iri

("http://www.rdfabout.com/rdf/usgov/congress/people/
A000360"),
   sem:curie-expand("foaf:img"), (), "="
 )
)))
```

SPARQL 式、および SPARQL 1.1 の `IN/NOT IN` 演算子でシーケンスを構築すると、`cts:query` 値のシーケンスを 1 番目の引数として想定する `cts:and-query` など、ビルトイン cts 関数を効果的に活用できます。

また、順序を指定するために、`cts:order` コンストラクタを `cts:search` のオプションとして使用することもできます。これにより、指定したインデックスを使用して cts 検索結果を並べることができるため、パフォーマンスの向上と予測可能性が実現されます。『*Query Performance and Tuning Guide*』の「Creating a cts:order Specification」を参照してください。

6.2.6.4 cts:contains

`cts:contains` 関数は、`FILTER` および `BIND` 節内に出現する SPARQL 式で使用できます。例については、「FILTER キーワード」（95 ページ）を参照してください。

`cts:contains` では任意の値を 1 番目の引数にできるため、クエリのトリプルパターンでバインドされた変数を 1 番目の引数として渡すことができます。トリプルパターンでは、トリプルインデックスの参照時に返される結果の数を減らすため、全文インデックスを使用します。

例えば以下のようになります。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

fn:count(sem:sparql('
```

```
PREFIX cts: <http://marklogic.com/cts#>

SELECT DISTINCT *
WHERE
{ ?s ?p ?o .
     FILTER cts:contains(?o, cts:word-query("Environment")) }
   ')
  )
=>
53
```

次の例は、法案番号「hr543」があるかどうかを検証するシンプルなクエリです。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

cts:contains(collection("info:govtrack/bills")//sem:subject,
  cts:word-query('hr543'))

=>
true
```

6.3 オプティック API を使用したトリプルのクエリ

サーバーサイドのトリプルクエリには、オプティック API も使用できます。オプティックを使用した次のクエリの例は、ブルックリンで出生した人のリストを、`person` と `name` の 2 列がある表の形式で返します。

```
xquery version "1.0-ml";
import module namespace op="http://marklogic.com/optic"
  at "/MarkLogic/optic.xqy";

let $resource :=
op:prefixer("http://dbpedia.org/resource/")
let $foaf     := op:prefixer("http://xmlns.com/foaf/0.1/")
let $onto     :=
op:prefixer("http://dbpedia.org/ontology/")
let $person   := op:col("person")

return op:from-triples((
   op:pattern($person, $onto("birthPlace"),
$resource("Brooklyn")),
   op:pattern($person, $foaf("name"), op:col("name"))))
   => op:result()
```

このクエリは、この章の前の方にあるクエリと同じデータセットを使用しています（「SPARQL を使用したトリプルのクエリ」（72 ページ）を参照）。結果は次のようになります。

sql as Table

person	name
http://dbpedia.org/resource/A.E._Coleby	A.E. Coleby
http://dbpedia.org/resource/A.S._Douglas	Alexander Shafto Douglas
http://dbpedia.org/resource/A._A._Pearson	A. A. Pearson
http://dbpedia.org/resource/A._B._Campbell	A. B. Campbell
http://dbpedia.org/resource/A._E._W._Mason	A E W Mason
http://dbpedia.org/resource/A._Follett_Osler	A. Follett Osler
http://dbpedia.org/resource/A._J._Ayer	Alfred Ayer
http://dbpedia.org/resource/A._L._Lloyd	Albert Lancaster Lloyd
http://dbpedia.org/resource/A._M._W._Stirling	A.M.W. Stirling
http://dbpedia.org/resource/A._R._Rawlinson	A.R Rawlinson
http://dbpedia.org/resource/Aaron_Hart_(businessman)	Aaron Hart
http://dbpedia.org/resource/Aaron_Patton	Aaron Patton

オプティック API の詳細については、『*Application Developer's Guide*』の「Optic API for Relational Operations」および「Data Access Functions」を参照してください。オプティックを使用したサーバーサイドクエリの詳細については、『Optic API』の「`op:from-triples`」または「`op.fromTriples`」を参照してください。

6.4 シリアライゼーション

さまざまな方法で、結果の出力シリアライゼーションを設定できます。これらのオプションは、JSON または XQuery 関数の一部としてクエリレベルで設定すると、デフォルトのオプションよりも優先させることができます。また、XQuery 宣言でメソッドを設定したり、アプリケーションサーバーでメソッドを設定したりすることができます。これらの出力オプションは、アプリケーションサーバーから返されるデータや、REST 経由で送信されるデータのシリアライズ方法に影響します。

6.4.1 出力メソッドの設定

クエリの結果の出力メソッドは次の方法で設定できます。優先順位が高いメソッドから記載しています。

- オプションを `xdmp:quote()` に設定する
- `xdmp:set-response-output-method()` を設定する
- XSLT 出力メソッドを設定する
- XQuery（または JavaScript）で静的宣言を使用する
- アプリケーションサーバーで出力を設定する

つまり、アプリケーションサーバーで設定したあらゆる設定は、XQuery または Javascript での静的宣言による設定によって上書きされます。

XQuery 宣言で出力メソッドを設定するには、次を使用します。

```
declare option xdmp:output "method = sparql-results-json"
```

XQuery 関数の一部として出力メソッドを設定するには、次を使用します。

```
set-response-output-method("sparql-results-json")
```

サーバーサイド JavaScript 関数の一部として出力メソッドを設定するには、次を使用します。

```
setResponseOutputMethod("sparql-results-json")
```

6.5 セキュリティ

トリプルインデックス、`cts:triples`、および `sem:sparql` の各クエリは、データベースユーザーが読み取りパーミッションを持つドキュメントからのトリプルだけを返します。

名前付きグラフは、コレクションで使用可能な書き込み保護設定を継承します。

タスク	権限
`sem:sparql` の実行	http://marklogic.com/xdmp/privileges/sem-sparql

MarkLogic のセキュリティの詳細については、『*Security Guide*』の「Document Permissions」を参照してください。

7.0 推論

MarkLogic のセマンティック、あるいは一般的なセマンティック技術では、データを理解するため、そのデータとルールの組み合わせに基づいて新しいファクトを自動的に発見する「推論」のプロセスが必要になります。推論とは、ルールの集まりに基づいてデータに関する新しいファクトを「推論」つまり発見するプロセスのことです。セマンティックのトリプルを使用した推論では、自動的な手続きによって、新しい関係性（新しいファクト）を既存のトリプルから生成できることになります。

inference query とは、自動推論の影響を受けるあらゆる SPARQL クエリを指します。推論を記述する W3C 規格、および関連する標準へのリンクについては、次を参照してください。http://www.w3.org/standards/semanticweb/inference

新しいファクトは、実装に応じて、データベースに追加することも（フォワードチェーン推論）、クエリ時に推論することも（バックワードチェーン推論）できます。MarkLogic では、自動バックワードチェーン推論に対応しています。

この章は、次のセクションで構成されています。

- 自動推論
- 推論を実現するその他の方法
- パフォーマンスに関する考慮事項
- REST API による推論の使用
- 推論で使用する API のまとめ

7.1 自動推論

自動推論は、ルールセットおよびオントロジーを使用して実行されます。その名前が示すように、自動推論は自動的に実行されますが、集中的に管理することもできます。MarkLogic のセマンティックではバックワードチェーン推論（backward-chaining inference）を使用します。つまり、推論はクエリ時に実行されます。これは、非常に柔軟性が高くなっています。つまり、クエリごとにいずれのルールセットとオントロジー（それぞれ複数可）を使用するかを、データベースごとのデフォルトルールセットとともに指定できます。

このセクションでは、次の内容を取り上げます。

- オントロジー
- ルールセット
- 推論に使用できるメモリ
- より複雑な使用例

7.1.1 オントロジー

オントロジーは、データのセマンティックを定義するために使用され、データに関する新しいファクトの推論に使用できる、データ内の関係を記述します。セマンティックでは、ontology とはトリプルの集まりを指し、この集まりにより、世界の一部を構成するセマンティックモデルを提供します。このモデルにより、特定の分野（人同士、出版物のタイプ同士、または薬のタクソノミー同士の関係）に関する知識を表現できるようになります。この知識モデルは、データ内の関係を記述するトリプルのコレクションから構成されています。さまざまな vocabularies で、ファクトを表現する概念および関係を定義するタームのセットを与えることができます。

オントロジーは、ドメイン内にどのタイプのものが存在するかと、それらにどのような関連があるかを記述します。語彙は、何らかの内部または外部機関によって管理されている明確な定義を持つ用語から構成されます。例えば、オントロジートリプル `ex:dog skos:broader ex:mammal` は「dog is part of the broader concept mammal（犬は、より広い概念「哺乳類」の一部です）」と述べています。

次の SPARQL の例では、そのオントロジートリプルをグラフに挿入します。

```
PREFIX skos:
<http://www.w3.org/2004/02/skos/core#Concept/>
PREFIX ex: <http://example.org/>

INSERT DATA
{
GRAPH <http://marklogic.com/semantics/animals/vertebrates>
{
ex:dog skos:broader ex:mammal .
}}
```

ビジネスまたはリサーチ分野をモデル化するために自分でオントロジーを作成して利用できます。またこれを、データに関する情報をさらに発見するために 1 つあるいは複数のルールセットとともに使用できます。

ルールセットは、オントロジートリプルも含め、クエリのスコープ内のトリプルすべてにわたって適用されます。オントロジートリプルは、使用するにはクエリのスコープに入れる必要があります。これを行うには、次のようにいくつかの方法があります。

- クエリで `FROM` または `FROM NAMED/GRAPH` を使用して、アクセスするデータを指定します。オントロジーは、コレクション / 名前付きグラフ別に編成されます。
- `sem:sparql` または `sem:sparql-update` に対して `default-graph=` および `named-graph=` オプションを使用します。

- `cts:query` を使用して、データをクエリ対象から除します。オントロジーは、ディレクトリ別など、`cts:query` で探すことができるあらゆるものを基準として編成できます。
- オントロジーをインメモリストアに追加し、データベースとインメモリストアの両方でクエリします。この場合、オントロジーはデータベースに格納されず、クエリごとに操作および変更できます。
- オントロジーを公理トリプルとしてルールセットに追加します。公理トリプルとは、ルールセットの記述が常に true であるトリプルのことで、ルール内で空の `WHERE` 節を使用することで示します。これにより、クエリ時に特定のルールセットファイルにオントロジーを含めるかどうかを選択できるようになります。

7.1.2 ルールセット

ルールセット（ruleset）とは、推論規則（データから追加のファクトを推論するために使用できる規則）の集まりのことです。ルールセットは、既存のトリプルから新しいトリプルをクエリ時に推論するために、MarkLogic の推論エンジンで使用されます。ルールセットは、他のルールセットをインポートして構築することもできます。推論ルールを使用すると、アサーテッドトリプル（asserted triples）と推論トリプル（inferred triples）の両方で検索できます。セマンティック推論エンジンでは、ルールセットを使用して、クエリ時に既存のトリプルから新しいトリプルを作成します。

例えば、ジョンがロンドンに住んでいること、そしてロンドンがイングランドにあることを「私」が知っている場合、私（人間）はジョンがイングランドに住んでいることを知っていることになります。つまり、私はそのファクトを「推論」したのです。同様に、ジョンがロンドンに住んでいるというトリプルとロンドンがイングランドにあるというトリプルの両方がデータベース内にあり、さらに、「〜に住んでいる」（lives in）と「〜にある」（is in）という意味をオントロジーの一部として表しているトリプルもある場合、MarkLogic はジョンがイングランドに住んでいると推論できます。また、イングランドに住んでいるすべての人に関するデータをクエリすると、ジョンが結果に含まれることになります。

「住んでいる」（lives in）という概念を表すシンプルなカスタムルール（ルールセット）を次に示します。

```
# geographic rules for inference
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema/>
PREFIX ex: <http://example.com/>
```

```
PREFIX gn: <http://www.geonames.org/ontology/>

RULE "livesIn" CONSTRUCT {
  ?person ex:livesIn ?place2
} {
  ?person ex:livesIn ?place1 .
  ?place1 gn:parentFeature ?place2
}
```

このルールは（下から読んでいくと）、place1 が place2 にあり（parentFeature を持ち）、person が place1 に住んでいる場合、person は place2 にも住んでいる、ということを記述しています。

ルールセットを使用してクエリ時に行われる推論は、「バックワードチェーン」推論と呼ばれます。各 SPARQL クエリは、指定されたルールセットを参照し、結果に基づいて新しいトリプルを作成します。この種の推論は、読み込みとインデックス付けを高速化しますが、クエリ時は少し低速になる可能性があります。一般に、推論は多くの（そして複雑な）ルールを追加するにつれて負荷が高く（速度が遅く）なります。

MarkLogic では、クエリごとに必要なルールセットだけを適用できます。データベースのデフォルトルールセット（複数可）を指定するのが便利ですが、クエリによってはデフォルトを無視することもできます。デフォルトのルールセットの関係性を利用せずに、推論を使用せずにクエリしたり、代替ルールセットを使用してクエリすることもできます。

このセクションでは、次の内容を取り上げます。

- 事前定義されたルールセット
- クエリのルールセットの指定
- Admin UI を使用してデータベースのデフォルトルールセットを指定する
- デフォルトルールセットよりも優先させる
- 新しいルールセットの作成
- ルールセットの文法
- ルールセットの例

7.1.2.1 事前定義されたルールセット

MarkLogic のセマンティックには、推論用に事前定義された標準ベースのルールセット（RDFS、RDFS-Plus、OWL Horst）が同梱されています。これらのルールセットは、MarkLogic 固有の言語で記述されています。この言語は、SPARQL の CONSTRUCT ク

エリと同じシンタックスを持ちます。各ルールセットには、完全なルールセット（xxx-full.rules）と最適化されたバージョン（xxx.rules）の 2 つのバージョンがあります。

クエリできめ細やかな推論を行うことができるように、これらの各ルールセットのコンポーネントは個別に使用できます。また、これらのルールセットをいくつかインポートしたり、独自のルールを記述したりすることで、独自のルールセットを作成することもできます。詳細については、「新しいルールセットの作成」（159 ページ）を参照してください。

（Linux で）これらの事前定義されたルールセットを確認するには、MarkLogic インストールディレクトリの下にある Config ディレクトリに移動します（/<MarkLogic_install_dir>/Config/*.rules）。例えば以下のようになります。

```
/opt/MarkLogic/Config/*.rules
```

.rules 拡張子を持つ一連のファイルが表示されます。.rules ファイルのそれぞれがルールセットです。Windows インストールの場合、これらのファイルは通常、C:\Program Files\MarkLogic\Config にあります。

次に、ルール domain.rules の例を示します。

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema/>

tbox {
  ?p rdfs:domain ?c .
}

RULE "domain axioms" construct {
  rdfs:domain rdfs:domain rdf:Property .
  rdfs:domain rdfs:range rdfs:Class .
} {}

RULE "domain rdfs2" CONSTRUCT {
  ?x a ?c
} {
  ?x ?p ?y .
  ?p rdfs:domain ?c
}
```

この例で、a は「〜のタイプ」（type of）（rdf:type または rdfs:type）を意味します。「domain rdfs2」ルールは、2 番目の波括弧内のすべてが真である（p がドメイン c を持つ。つまり、述語 p を持つすべてのトリプルについて、目的語がドメイン c

内にある必要があります）場合、1 番目の波括弧内のトリプルを構築する（x p y がある場合、x は c である）ということを記述しています。

ルールをテキストエディタで開くと、一部のルールセットがコンポーネント化されていることがわかります。つまり、ルールセットが小さなコンポーネントルールセット内に定義されていて、それらが積み重なることでより大きなルールセットになっています。例えば、rdfs.rules は、次のように他の 4 つのルールをインポートして、rdfs ルールの最適化されたセットを作成します。

```
# RDFS 1.1 optimized rules
prefix rdf:     <http://www.w3.org/1999/02/
22-rdf-syntax-ns#>
prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#>

import "domain.rules"
import "range.rules"
import "subPropertyOf.rules"
import "subClassOf.rules"

RULE "rdf classes" construct {
...
...
```

ルールセットの使用（および作成）の際にビルディングブロックアプローチを使用することで、本当に必要なルールだけを有効にできるため、クエリが可能な限り効率的になります。ルールセットのシンタックスは、SPARQL の CONSTRUCT のシンタックスに似ています。

7.1.2.2 クエリのルールセットの指定

sem:ruleset-store を使用すると、SPARQL クエリに使用するルールセットを選択できます。そのルールセットが関数の一部として指定されます。sem:ruleset-store 関数は、$store で提供される sem:store 関数によって定義されているトリプルにルールセットを適用した結果となるトリプルのセットを返します（例えば、「このルールから推論できるすべてのトリプル」）。

このステートメントでは、rdfs.rules ルールセットを sem:ruleset-store の一部として指定します。

```
let $rdfs-store := sem:ruleset-store("rdfs.rules",
sem:store() )
```

したがって上記は、$rdfs-store に、sem:store に対して rdfs.rules を使用した推論によって生成されるトリプルを格納することを述べています。sem:store に指定されている値がない場合、クエリは現在のデータベースのトリプルインデックスにある

トリプルを使用します。ビルトイン関数 sem:store および sem:ruleset-store は、クエリ対象とするトリプルや、クエリで使用するルールセット（ある場合）を定義するために使用します。$store 定義には、ルールセットに加え、クエリの定義域を制限するその他の方法（cts:query）が含まれます。

次の例では、事前定義された rdfs.rules ルールセットを使用しています。

```
import module namespace sem =
"http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";
let $sup :=
'PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>

INSERT DATA
{ <someMedicalCondition> rdf:type <osteoarthritis> .
  <osteoarthritis> rdfs:subClassOf <bonedisease> .}'
return sem:sparql-update($sup)
; (: transaction separator :)

let $sq :=
'PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX d: <http://diagnoses#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?diagnosis
WHERE { ?diagnosis rdf:type <bonedisease>.} '

(: rdfs.rules is a predefined ruleset :)
let $rs := sem:ruleset-store("rdfs.rules", sem:store())
return sem:sparql($sq, (), (), $rs)
(: the rules specify that query for <bonedisease> will
return the subclass <osteoarthritis> :)
```

注： sem:store や sem:ruleset-store を含む SPARQL クエリの一部としてグラフ URI が含まれる場合、クエリにはストア内のトリプルのほか、グラフ内のトリプルも含まれます。

次の例は、$triples 内のデータに対する SPARQL クエリです。ルールセット rdfs:subClassOf および rdfs:subPropertyOf を使用しています。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/
  semantics" at "/MarkLogic/semantics.xqy";
```

```
let $triples := sem:store((), cts:word-query("henley"))
return
sem:sparql("
PREFIX skos: <http://www.w3.org/2004/02/skos/
  core#Concept/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT * { ?c a skos:Concept; rdfs:label ?l }",(),(),
sem:ruleset-store(("subClassOf.rules",
  "subPropertyOf.rules"),($triples))
)
```

データベースのデフォルトルールセットの管理には、Admin UI、REST Mangement API、または XQuery Admin API を使用できます。Admin UI でルールセットを指定する方法については、「Admin UI を使用してデータベースのデフォルトルールセットを指定する」（154 ページ）を参照してください。REST Management API の詳細については、`PUT:/manage/v2/databases/{id|name}/properties` の `default-ruleset` プロパティを参照してください。Admin API の詳細については、`admin:database-add-default-ruleset` を参照してください。

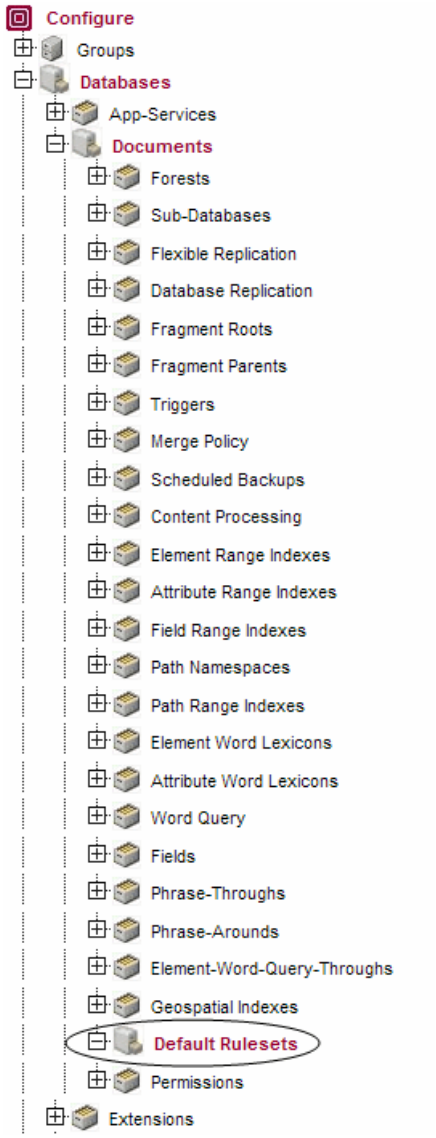
7.1.2.3 Admin UI を使用してデータベースのデフォルトルールセットを指定する

Admin UI を使用すると、特定のデータベースに対してクエリに使用するデフォルトルールセットを設定できます（このデータベースを使用するときは、クエリにこのルールセットを使用するなど）。

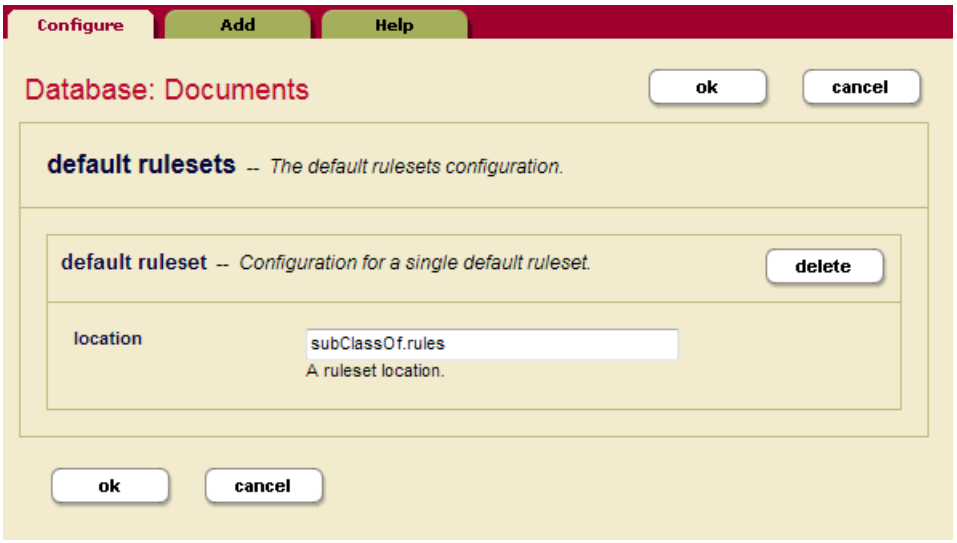
データベースに 1 つまたは複数のルールセットを指定するには、次の手順に従います。

1. Admin UI の左側のツリーメニューで、[Databases] をクリックします。

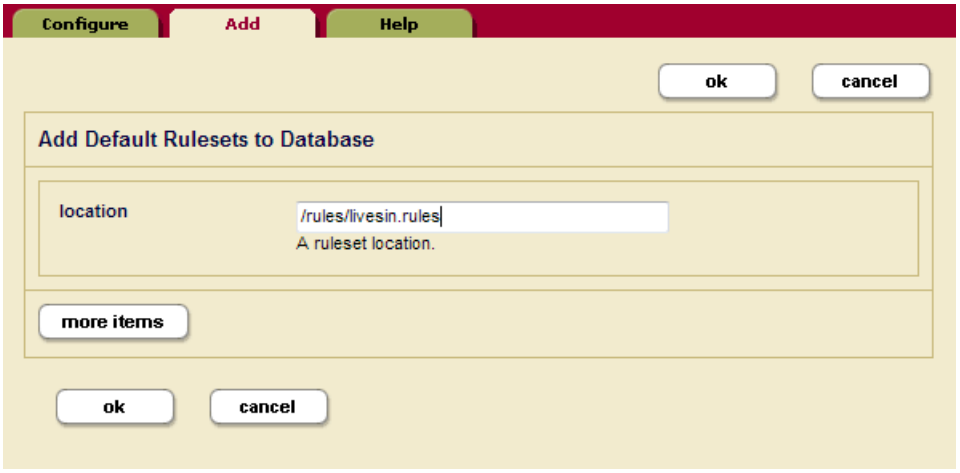
2. データベース名をクリックして、リストを展開し、[Default Rulesets]までスクロールします。



3. ［Default Rulesets］をクリックすると、現在 Documents データベースに関連付けられているルールセットが表示されます。



4. 独自のルールセットを追加するには、［Add］をクリックしてルールセットの名前と場所を入力します。



5. カスタムルールセットは Schemas データベース内にあります。

   MarkLogic から提供されているルールセットは、MarkLogic インストールディレクトリの下にある Config ディレクトリ内に配置されています (`/<MarkLogic_install_dir>/Config/*.rules`)。

6. このデータベースに追加のルールセットを関連付けるには、さらに項目をクリックします。

注： ルールセットのセキュリティは、MarkLogic スキーマのセキュリティと同じ方法で管理されます。

Query Console では、`admin:database-get-default-rulesets` 関数を使用して、データベースに現在関連付けられているデフォルトルールセットを調べることができます。

この例では、Documents データベースのデフォルトルールセットの名前と場所が返されます。

```
xquery version "1.0-ml";
import module namespace admin =
"http://marklogic.com/xdmp/admin" at "/MarkLogic/
  admin.xqy";

let $config := admin:get-configuration()
let $dbid := admin:database-get-id($config, "Documents")
return admin:database-get-default-rulesets($config, $dbid)

=>

<default-ruleset
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://marklogic.com/xdmp/database">
    <location>/rules/livesin.rules</location>
</default-ruleset>
```

注： デフォルトのルールセットがデータベースに関連付けられている状態で、他のルールセットをクエリの一部で指定した場合、両方のルールセットが使用されます。ルールセットは累積的です。デフォルトルールセットを無視するには、`sem:store` で `no-default-ruleset` オプションを使用します。

7.1.2.4 デフォルトルールセットよりも優先させる

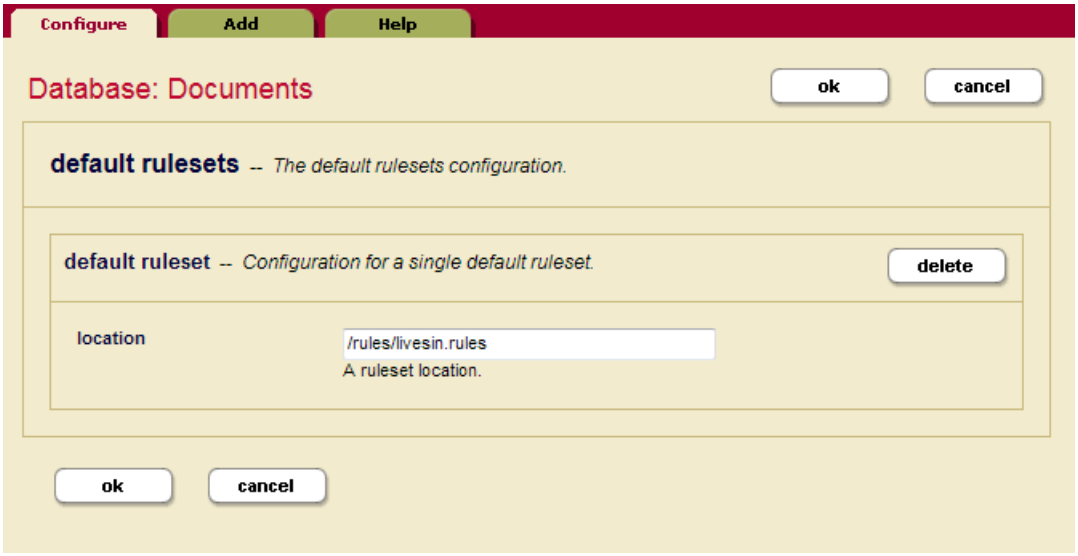
データベースでデフォルトとして設定されたルールセットは、オフにしたり無視したりできます。この例では、SPARQL クエリがデータベースに対して実行されますが、デフォルトルールセットは無視され、クエリには `rdfs:subClassOf` 推論ルールセットが使用されます。

```
xquery version "1.0-ml";
import module namespace sem =
"http://marklogic.com/semantics" at "/MarkLogic/
  semantics.xqy";
```

```
sem:sparql("
PREFIX skos:
<http://www.w3.org/2004/02/skos/core#Concept/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT * {
  ?c a skos:Concept;
  rdfs:label ?l }",(),(),
sem:ruleset-store("subClassOf.rules",sem:store("no-
default-rulesets"))
)
```

また、クエリの一部であるルールセットもオフにしたり無視したりできます。それには Admin UI を使用するか、XQuery または JavaScript を使用してルールセットを指定します。

また、Admin UI でデフォルトルールセットをデータベースから「削除」して、データベースのデフォルトルールセットを変更することもできます。Admin UI で、左側にあるナビゲーションパネルからデータベース名を選択し、データベース名をクリックします。[Default Rulesets] をクリックします。



[Database: Documents] パネルで、削除するデフォルトルールセットを選択して [delete] をクリックします。完了したら [ok] をクリックします。削除されたルールセットは、そのデータベースのデフォルトルールセットではなくなります。

注： このアクションを実行してもルールセット自体は削除されません。デフォルトルールセットではなくなるだけです。

XQuery で `admin:database-delete-default-ruleset` を使用して、データベースのデフォルトルールセットを変更することもできます。この例では、Documents データベースのデフォルトルールセットとしての `subClassOf.rules` を取り除きます。

```
xquery version "1.0-ml";
import module namespace admin =
"http://marklogic.com/xdmp/admin"
  at "/MarkLogic/admin.xqy";

let $config := admin:get-configuration()
let $dbid := admin:database-get-id($config, "Documents")
let $rules := admin:database-ruleset("subClassOf.rules")
let $c := admin:database-delete-default-ruleset($config,
$dbid, $rules)

return admin:save-configuration($c)
```

7.1.2.5 新しいルールセットの作成

MarkLogic では、推論に使用する独自のルールセットを作成できます。MarkLogic のルールセットは、MarkLogic 固有の言語で記述されており、SPARQL の `CONSTRUCT` クエリに基づいています。

推論ルールは、複数の推論トリプルを構築するための方法として考えることもできます。構築後は、新しいデータセット（`sem:store` によって定義されるデータベースの部分と推論トリプルとを含むデータセット）に対して検索を実行できます。

MarkLogic 提供のルールセットは、インストールディレクトリにあります。

```
/<MarkLogic_install_dir>/Config/*.rules
```

カスタムルールセットを作成する場合は、Schemas データベースに挿入し、Schemas データベース内の URI として参照します。ルールセットの場所は、Schemas データベース内で使用しているデータベースの URI、または `<MarkLogic Install Directory>/Config` あるファイル名のいずれかになります。

> **注：** MarkLogic は、まず、`/Config` にある MarkLogic 提供のルールセットを検索してから、Schemas データベース内で他のあらゆるルールセットを検索します。

7.1.2.6 ルールセットの文法

MarkLogic ルールセットは、MarkLogic 固有の言語で記述されています。この言語は、SPARQL 1.1 の文法に基づいています。推論ルールのシンタックスは、SPARQL の `CONSTRUCT` の文法に似ています。ただし `WHERE` 節は、トリプルパターン、結合、およびフィルタの組み合わせに制限されます。ルールセットには、一意の名前が必要です。

次の文法は、MarkLogic ルールセット言語を指定しています。

```
Rules ::= RulePrologue Rule*
Rule ::= 'RULE' RuleName 'CONSTRUCT' ConstructTemplate
'WHERE'?
    RuleGroupGraphPattern
RuleName ::= String
RuleGroupGraphPattern ::= '{' TriplesBlock? ( ( Filter
    RuleGroupGraphPattern ) '.'? TriplesBlock? )* '}'
RulePrologue ::= ( BaseDecl | PrefixDecl | RuleImportDecl )*
RuleImportDecl ::= 'IMPORT' RuleImportLocation
RuleImportLocation ::= String
```

RuleImportLocation の String は、インポートするルールの場所の URI にする必要があります。ここで定義されていない（BaseDecl など）非終端記号は、SPARQL 1.1 文法にある生成規則への参照です。

- この文法により、ルールの WHERE 節のコンテンツが制限され、静的分析中はさらにトリプルパターン、結合、およびフィルタの組み合わせに制限されます。
- 標準の SPARQL コメントシンタックスを使用したコメントは使用できます（「#」形式のコメントで、IRI または文字列の外側にあり、行の末尾まで続きます）。
- MarkLogic のルールセットは、拡張子「.rules」を使用し、MIME タイプが「application/vnd.marklogic-ruleset」になります。
- ルールセットの一部として、特定の種類のプロパティパス演算子（「/」、「^」など）が使用できます。ただし、次の演算子は、ルールセット内のプロパティパスの一部としては使用できません：「|」、「?」、「*」、「+」。

プロローグの import ステートメントには、指定された場所で見つかったルールセットからのルールのほか、推移的にインポートされた他のすべてのルールセットからのルールも、すべて含まれます。特定の場所にある 1 つのルールセットを複数回インポートしても、インポートの効果は 1 回だけインポートした場合と変わりません。1 つのルールセットを異なる場所から複数回インポートすると、MarkLogic では、それらは異なるルールセットと見なされ、それらが含んでいる重複したルール名のためにエラーが生成されます（XDMP-DUPRULE）。

7.1.2.7 ルールセットの例

/<MarkLogic Install>/Config ディレクトリにあるこのルールセット（subClassOf.rules）には、プレフィックスが含まれ、ルール名および CONSTRUCT 節があります。subClassOf rdfs9 ルールが、役目を果たすルールです。

```
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:    <http://www.w3.org/2000/01/rdf-schema#>
```

```
tbox {
  ?c1 rdfs:subClassOf ?c2 .
}

RULE "subClassOf axioms" CONSTRUCT {
  rdfs:subClassOf rdfs:domain rdfs:Class .
  rdfs:subClassOf rdfs:range rdfs:Class .
} {}

RULE "subClassOf rdfs9" CONSTRUCT {
  ?x a ?c2
} {
  ?x a ?c1 .
  ?c1 rdfs:subClassOf ?c2 .
  FILTER(?c1!=?c2)
}
```

subClassOf rdfs9 ルールには FILTER 節も含まれることに注意してください。

同じディレクトリにある次のルールセット（rdfs.rules）は、より小さなルールセットをインポートして、完全な RDFS ルールセットに近づけるルールセットを作成します。

```
# RDFS 1.1 optimized rules
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-
ns#>
PREFIX rdfs:    <http://www.w3.org/2000/01/rdf-schema#>

import "domain.rules"
import "range.rules"
import "subPropertyOf.rules"
import "subClassOf.rules"

RULE "rdf classes" CONSTRUCT {
  rdf:type a rdf:Property .
  rdf:subject a rdf:Property .
  rdf:predicate a rdf:Property .
  rdf:object a rdf:Property .
  rdf:first a rdf:Property .
  rdf:rest a rdf:Property .
  rdf:value a rdf:Property .
  rdf:nil a rdf:List .
} {}

RULE "rdfs properties" CONSTRUCT {
  rdf:type rdfs:range rdfs:Class .
```

```
  rdf:subject rdfs:domain rdf:Statement .
  rdf:predicate rdfs:domain rdf:Statement .
  rdf:object rdfs:domain rdf:Statement .

  rdf:first rdfs:domain rdf:List .
  rdf:rest rdfs:domain rdf:List .
  rdf:rest rdfs:range rdf:List .

  rdfs:isDefinedBy rdfs:subPropertyOf rdfs:seeAlso .
} {}

RULE "rdfs classes" CONSTRUCT {
  rdf:Alt rdfs:subClassOf rdfs:Container .
  rdf:Bag rdfs:subClassOf rdfs:Container .
  rdf:Seq rdfs:subClassOf rdfs:Container .
  rdfs:ContainerMembershipProperty rdfs:subClassOf
rdf:Property .
} {}

RULE "datatypes" CONSTRUCT {
  rdf:XMLLiteral a rdfs:Datatype .
  rdf:HTML a rdfs:Datatype .
  rdf:langString a rdfs:Datatype .
} {}

RULE "rdfs12" CONSTRUCT {
  ?p rdfs:subPropertyOf rdfs:member
} {
  ?p a rdfs:ContainerMembershipProperty
}
```

次に示すのは、前述したカスタムルールです。地理的な場所に関する情報の推論に使用するために作成できます。

```
# geographic rules for inference
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema/>
PREFIX ex: <http://example.com/>
PREFIX gn: <http://www.geonames.org/ontology/>

RULE "lives in" CONSTRUCT {
  ?person ex:livesIn ?place2
} {
  ?person ex:livesIn ?place1 .
  ?place1 gn:parentFeature ?place2
}
```

xdmp:document-insert および Query コンソールを使用して、livesIn ルールを Schemas データベースに追加します。Schemas データベースがコンテンツソースとして選択されていることを必ず確認してから、次のコードを実行します。

```
xquery version "1.0-ml";

xdmp:document-insert(
'/rules/livesin.rules',
text{'
# geographic rules for inference
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema/>
PREFIX ex: <http://example.com/>
PREFIX gn: <http://www.geonames.org/ontology/>

RULE "lives in" CONSTRUCT {
  ?person ex:livesIn ?place2
} {
  ?person ex:livesIn ?place1 .
  ?place1 gn:parentFeature ?place2
  }'
})
```

この例により、livesin.rule が Schemas データベース、rules ディレクトリ（/rules/livesin.rules）に格納されます。作成したルールセットは、付属のルールセットを推論の一部として含める場合と同様の方法で含めることができます。MarkLogic では、Schemas データベースでルールの場所が確認されてから、付属のルールセットの場所が確認されます。

7.1.3 推論に使用できるメモリ

デフォルト、最大、および最小推論サイズの値はいずれも、システムごとではなくクエリごとになります。最大推論サイズは、推論用のメモリ上限になります。appserver-max-inference-size 関数を使用すると、管理者が推論のメモリ上限を設定できます。この量を超えることはできません。

デフォルト推論サイズは、推論に使用できるメモリの量です。デフォルトでは、推論に使用できるメモリの量は 100MB です（size=100）。メモリが足りなくなり、推論フルのエラー（INFFULL）が発生した場合は、admin:appserver-set-default-inference-size を使用してデフォルトのメモリを増やすか、Admin UI の HTTP Server 設定ページでデフォルト推論サイズを変更する必要があります。

また、クエリ内で sem:ruleset-store の一部として推論メモリサイズを設定することもできます。次のクエリは、推論のメモリサイズを 300MB（size=300）に設定しています。

```
Let $store := sem:ruleset-store(("baseball.rules",
"rdfs-plus-full.rules"),
sem:store(), ("size=300"))
```

クエリが INFFULL 例外を返す場合は、ruleset-store でサイズを変更できます。

7.1.4 より複雑な使用例

推論はより複雑なクエリで使用できます。次に、SPARQL クエリの JavaScript の例を示します。このクエリでは、オントロジーがインメモリのストアに追加されます。インメモリのストアは、推論を使用して、山羊のソフトチーズを使用するレシピを発見します。このクエリにより、可能性のあるレシピタイトルのリストが返されます。

```
var sem = require("/MarkLogic/semantics.xqy");

var inmem = sem.inMemoryStore(
  sem.rdfParse(`
    prefix ch: <http://marklogic.com/semantics/cheeses/>
    prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    prefix owl: <http://www.w3.org/2002/07/owl#>
    prefix dcterms: <http://purl.org/dc/terms/>

    ch:FreshGoatsCheese owl:intersectionOf (
      ch:SoftFreshCheese
      [ owl:hasValue ch:goatsMilk ;
      owl:onProperty ch:milkSource ]
    ) .`,"turtle"));
var rules = sem.rulesetStore(
  ["intersectionOf.rules","hasValue.rules"],
  [inmem,sem.store()])

sem.sparql(`
    prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    prefix dcterms: <http://purl.org/dc/terms/>
    prefix f: <http://linkedrecipes.org/schema/>
    prefix ch: <http://marklogic.com/semantics/cheeses/>

select ?title ?ingredient WHERE {
  ?recipe dcterms:title ?title ;
    f:ingredient [
      a ch:FreshGoatsCheese ;
      rdfs:label ?ingredient]
    }`,[],[],rules)
```

このクエリの結果を得るには、ヤギの乳から作られているチーズを記述しているトリプルも含む、レシピのトリプルストアも用意する必要があります。

7.2 推論を実現するその他の方法

自動推論を進める前に、各自のユースケースにより適している可能性がある、その他の推論手法を確認しておきましょう。

このセクションでは、次の内容を取り上げます。

- パスの使用
- マテリアライズ

7.2.1 パスの使用

多くの場合、推論を実行するにはクエリを書き換えます。例えば、非列挙プロパティパス（property path）を使用すると、シンプルな推論を実行できます。プロパティパス（「プロパティパス式」（113 ページ）で説明）が、シンプルな種類の推論を可能にしています。

SPARQL クエリで RDFS 語彙および「/」プロパティパスを使用すると、リソースのスーパータイプなど、リソースの、可能性のあるタイプすべてを探すことができます。

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2001/01/rdf-schema#>
SELECT ?type
{
   <http://example/thing> rdf:type/rdfs:subClassOf* ?type
}
```

結果は、すべてのリソースおよびそれらの推論タイプになります。非列挙プロパティパスの式に星印（*）を付けると、主語と目的語が `rdf:type` でつながれ、その後に `rdfs:subClassOf` が 0 回以上出現するトリプルが探されます。

例えば、次のクエリを使用して、subClasses が「shirt」である製品を階層のすべての深さで探すことができます。

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ex: <http://example.com>

SELECT ?product
WHERE
  {
    ?product rdf:type/rdfs:subClassOf* ex:Shirt ;
}
```

また、プロパティパスを使用して、イングランド（England）に住んでいる人を探すことができます。

```
PREFIX gn: <http://www.geonames.org/ontology/>
PREFIX ex: <http://www.example.org>

SELECT ?p
{
  ?p ex:livesIn/gn:parentFeature "England"
}
```

プロパティパスの詳細、およびセマンティックでのプロパティパスの使用方法の詳細については、「プロパティパス式」（113 ページ）を参照してください。

7.2.2 マテリアライズ

クエリ時の自動推論に代わる方法（backward-chaining inference）の 1 つがマテリアライズで、forward-chaining inference とも呼ばれます。この方法では、クエリの一部としてではなく、データの部分に対して推論を実行します。推論されたトリプルは、後でクエリできるように格納されます。マテリアライズは、ほぼ静的なトリプルデータに最適です。あまり変更されないルールやオントロジーを使用して推論を実行します。

読み込み時や更新時に行われるこのマテリアライズプロセスは、時間がかかることがあり、格納に大量のディスク領域を必要とします。コードやスクリプトを記述して、トランザクションやセキュリティを処理したり、データやオントロジーの変更を処理したりする必要があります。

> **注：** 自動推論を選択した場合は、このようなタスクがすべて自動で処理されます。

マテリアライズが非常に有用なのは、非常に高速なクエリが必要で、事前処理作業が可能であり、推論トリプル用の追加のディスク領域が確保できる場合です。このタイプの推論は、データ、ルールセット、オントロジー、データの一部があまり変更されない場合の使用に適しています。

もちろん、組み合わせることもできます。つまり、あまり変更が発生しない推論トリプル（オントロジートリプルなど。例えば、「カスタマーは人である」は法的なエンティティです）はマテリアライズする一方で、頻繁に変更や追加が発生するトリプルには自動推論を使用できます。また、推論が持つスコープがより広い場合に自動推論を使用できます（`new-order-111` が `line-item-222` を含み、`line-item-222` が `product-333` を含み、`product-333` が `accessory-444` に関連する場合など）。

7.3 パフォーマンスに関する考慮事項

SPARQL クエリの実行を高速化するために重要なのは、（返される結果の数が小さくなるように十分にリッチなクエリを記述して）データを「パーティション化」することです。クエリ対象のトリプルが少ないほど、バックワードチェーン推論は、利用可能なメモリで高速に実行されます。これを実現するには、`FILTER` を使用したり、`cts:query`（例えば、collection-query）を通じてクエリのスコープを制約したりして、推論クエリの選択条件を厳しくします。

7.3.1 部分的なマテリアライズ

使用頻度は高いが、変更される頻度はあまり高くないデータ、ルールセット、オントロジーを部分的にマテリアライズできます。データの一部に対して推論を実行して、推論トリプルをマテリアライズし、マテリアライズされたこのトリプルを推論クエリで使用できます。

このトリプルをマテリアライズするには、推論に使用するルールの SPARQL クエリを構築し、パイプラインの読み込みまたは更新の一部としてそのクエリをデータに対して実行します。

7.4 REST API による推論の使用

REST Client API のメソッド `POST:/v1/graphs/sparql` または `GET:/v1/graphs/sparql` を使用して SPARQL クエリを実行または更新するときは、リクエストパラメータ `default-rulesets` および `rulesets` を通じてルールセットを指定できます。いずれのパラメータも省略すると、データベースのデフォルトルールセットが適用されます。

`rdfs.rules` および `equivalentProperties.rules` をデータベースのデフォルトルールセットに設定したら、Query Console から REST を使用して次の SPARQL クエリを実行できます。

```
xquery version "1.0-ml";
import module namespace sem =
"http://marklogic.com/semantics" at "/MarkLogic/
  semantics.xqy";

let $uri := "http://localhost:8000/v1/graphs/sparql"
return
let $sparql :='
PREFIX rdf:    <http://www.w3.org/1999/02/
22-rdf-syntax-ns#>
PREFIX rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
PREFIX prod:   <http://example.com/products/>
PREFIX ex:     <http://example.com/>
```

```
SELECT ?product
FROM <http://marklogic.com/semantics/products/inf-1>
WHERE
 {
  ?product  rdf:type  ex:Shirt ;
  ex:color  "blue"
}
'
let $response :=
xdmp:http-post($uri,
<options xmlns="xdmp:http">
  <authentication method="digest">
    <username>admin</username>
    <password>admin</password>
  </authentication>
  <headers>
    <content-type>application/sparql-query</content-type>
    <accept>application/sparql-results+xml</accept>
  </headers>
</options>
text {$sparql})
  return
  ($response[1]/http:code, $response[2] /node())

=>

    product
<http://example.com/products/1001>
<http://example.com/products/1002>
<http://example.com/products/1003>
```

（データベースのデフォルトルールセットが同じ場合に）REST エンドポイントと `curl` を使用すると、このクエリは次のようになります。

```
curl --anyauth --user Admin:janem-3 -i -X POST \
-H "Content-type:application/x-www-form-urlencoded" \
-H "Accept:application/sparql-results+xml" \
--data-urlencode query='PREFIX rdf:
<http://www.w3.org/1999/02/22-rdf-syntax-ns#> \
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX prod: <http://example.com/products/> \
PREFIX ex: <http://example.com/> \
SELECT ?product FROM
<http://marklogic.com/semantics/products/inf-1> \
WHERE {?product rdf:type ex:Shirt ; ex:color  "blue"}' \
http://localhost:8000/v1/graphs/sparql
```

この curl の例を実行するには、「\」文字を削除し、コマンドを 1 行にしてください。詳細については、「REST クライアント API でセマンティックを使用する」（194 ページ）、および『*REST Application Developer's Guide*』の「Querying Triples」を参照してください。

7.5 推論で使用する API のまとめ

MarkLogic には、セマンティック推論で使用できる複数の API が用意されています。セマンティック API は、実際の推論クエリの一部として使用できます（クエリするトリプルおよび適用するルールを指定します）。データベース API は、特定のデータベースによって推論で使用されるルールセットを選択する目的で使用できます。管理 API は、アプリケーションサーバーまたはタスクサーバーで推論に使用されるメモリを制御できます。

- セマンティック API
- データベースルールセット API
- 管理 API

7.5.1 セマンティック API

MarkLogic のセマンティック API は、トリプル管理、推論、および個々のクエリで（またはデータベースのデフォルトで）使用されるルールセットの指定に使用できます。ストアは、クエリで評価するトリプルのサブセットを識別するために使用されます。

セマンティック API	説明
sem:store	sem:sparql のクエリ引数は、sem:store を利用して、クエリの一部として評価するトリプルのソースを示すことができます。複数の sem:store コンストラクタを指定した場合、すべてのソースからのトリプルがマージされ、一体となってクエリされます。 sem:store には、cts:query とともに 1 つあるいは複数のオプションを含ませることで、sem:sparql クエリの一部として評価するトリプルのスコープを制限できます。sem:store パラメータは、sem:sparql-update および sem:sparql-values と併用することもできます。

セマンティック API	説明
sem:in-memory-store	sem:store を返します。これは、引数として渡される sem:triple 値からのトリプルの集まりを表現します。現在のデータベースで設定されたデフォルト ルールセットは、sem:in-memory-store で作成された sem:store には影響しません。
sem:ruleset-store	元のトリプルに加え、sem:store 内のトリプルにルールセットを適用することで生成されるトリプルの集まりを表現する新しい sem:store を返します。

注： sem:in-memory-store 関数は、推奨されない sem:sparql-triples 関数（MarkLogic 7 で使用可能）ではなく sem:sparql との併用を優先してください。現在は、sem:sparql の cts:query 引数も推奨されていません。

インメモリのトリプルに基づくストア（つまり、sem:in-memory-store によって作成されたストア）で sem:sparql-update を呼び出すと、ディスク上にはないインメモリのトリプルを更新できないため、エラーが発生します。同様に、複数のストアを sem:sparql-update に渡す場合も、それらのうちのいずれかがインメモリのトリプルに基づく場合は、エラーが発生します。

7.5.2 データベースルールセット API

次のデータベースルールセット API は、データベースに関連付けられたルールセットを管理する目的で使用します。

ルールセット API	説明
admin:database-ruleset	データベースで推論に使用するルールセット要素。1 つあるいは複数のルールセットを推論に使用できます。デフォルトでは、ルールセットは設定されていません。
admin:database-get-default-rulesets	データベースのデフォルトルールセットを返します。
admin:database-add-default-ruleset	データベースで推論に使用するルールセットを追加します。1 つあるいは複数のルールセットを推論に使用できます。デフォルトでは、ルールセットは設定されていません。

ルールセット API	説明
admin:database-delete-default-ruleset	データベースが推論に使用するデフォルトルールセットを削除します。

7.5.3 管理 API

次の管理 API は、推論に割り当てるメモリサイズ（デフォルト、最小、および最大）を管理する目的で使用します。

管理 API（admin:）	説明
admin:appserver-set-default-inference-size	このアプリケーションサーバーにおけるリクエストの推論サイズのデフォルト値を指定します。
admin:appserver-get-default-inference-size	アプリケーションサーバーによる推論で、sem:store が使用できるデフォルトのメモリ量（MB 単位）を返します。
admin:taskserver-set-default-inference-size	このタスクサーバーにおけるリクエストの推論サイズのデフォルト値を指定します。
admin:taskserver-get-default-inference-size	タスクサーバーによる推論で、sem:store が使用できるデフォルトのメモリ量（MB 単位）を返します。
admin:appserver-set-max-inference-size	リクエストの推論サイズの上限を指定します。推論サイズは、このアプリケーションサーバーで sem:store が推論を実行するために使用できる最大のメモリ量（MB 単位）です。
admin:appserver-get-max-inference-size	アプリケーションサーバーによる推論で、sem:store が使用できる最大のメモリ量（MB 単位）を返します。
admin:taskserver-set-max-inference-size	リクエストの推論サイズの上限を指定します。推論サイズは、このタスクサーバーで sem:store が推論を実行するために使用できる最大のメモリ量（MB 単位）です。
admin:taskserver-get-max-inference-size	タスクサーバーによる推論で、sem:store が使用できる最大のメモリ量（MB 単位）を返します。

8.0 SPARQL Update

SPARQL1.1 Update 言語は、トリプルおよびグラフを削除、挿入、および更新（削除 / 挿入）する目的で使用します。更新は、実際にはデータベースでの `INSERT/DELETE` 操作です。

SPARQL Update は、SPARQL 1.1 規格スイートの一部です（http://www.w3.org/TR/2013/REC-sparql11-update-20130321）。ただし、SPARQL Query とは別の言語です。SPARQL Update を使用するとトリプルやトリプルの集まりを操作できますが、SPARQL クエリ言語で実行できるのはトリプルデータの検索やクエリです。

SPARQL Update を使用すると、セキュリティレベルを管理できます。管理対象トリプルに対するすべての SPARQL クエリは、グラフパーミッションによって管理されます。トリプルドキュメントは、読み込み時のパーミッションを継承します。

SPARQL Update の影響を受けるのは、MarkLogic で管理されているトリプル（ドキュメントルートが `sem:triples` であるトリプル）だけです。管理対象トリプルは、次の関数を使用してデータベースにロードされたトリプルです。

- mlcp と `-input_file_type RDF`
- `sem:rdf-load`
- `sem:rdf-insert`
- `sem:sparql-update`

埋め込みトリプルは、XML または JSON ドキュメントの一部です。埋め込みトリプルを作成、削除、または更新する場合は、適切なドキュメント更新関数を使用してください。ドキュメントに埋め込まれたトリプルの詳細については、「非管理対象トリプル」（64 ページ）を参照してください。非管理対象トリプルも、ドキュメント管理関数を使用して修正および更新できます。詳細については、「XQuery およびサーバーサイド JavaScript でのトリプルの挿入、削除、および修正」（267 ページ）を参照してください。

このセクションは、次のように構成されています。

8.1 SPARQL Update の使用

SPARQL Update を使用すると、グラフストア（Graph Store）で管理対象トリプルを挿入および削除できます。SPARQL Update の操作は、グラフデータ操作とグラフ管理操作の 2 種類で構成されています。

SPARQL Update は、次に示す複数の方法で使用できます。

- Query Console から：ドロップダウンリストからクエリタイプとして SPARQL Update を選択します。「Query Console での SPARQL Update の使用」（191 ページ）を参照してください。
- XQuery または JavaScript を使用：SPARQL Update を XQuery（`sem:sparql-update`）または JavaScript（`sem.sparqlUpdate`）から呼び出します。「XQuery またはサーバーサイド JavaScript での SPARQL Update の使用」（191 ページ）を参照してください。
- REST API（`GET:/v1/graphs/sparql` または `POST:/v1/graphs/sparql`）経由で使用します。「REST での SPARQL Update の使用」（193 ページ）を参照してください。

SPARQL Update は管理対象トリプル（managed triples）に使用します。「埋め込まれた」トリプルまたは非管理対象トリプル（unmanaged triples）を修正するには、XQuery または JavaScript で適切なドキュメント更新関数を使用します。「XQuery およびサーバーサイド JavaScript でのトリプルの挿入、削除、および修正」（267 ページ）を参照してください。

SPARQL Update には、新しいロール `sparql-update-user` が追加されています。グラフに対してトリプルを挿入（`INSERT`）、削除（`DELETE`）、または更新（`DELETE/INSERT`）するには、ユーザーが `sparql-update-user` の各権限を持っている必要があります。詳細については、『*Security Guide*』の「Role-Based Security Model」を参照してください。

8.2 SPARQL Update によるグラフの操作

SPARQL Update を使用すると、RDF グラフを操作できます。グラフ管理操作は、`CREATE`、`DROP`、`COPY`、`MOVE`、および `ADD` で構成されています。

SPARQL Update の規格には、RDF グラフを操作するために次の各コマンドとオプションが用意されています。

コマンド	オプション	説明
CREATE	SILENT、GRAPH IRIref	新しいグラフを作成します。グラフの名前を指定するには GRAPH を使用します。SILENT は、グラフをサイレントに作成します。つまり、グラフがすでに存在する場合でも、エラーを返しません。
DROP	SILENT、GRAPH IRIref、DEFAULT、NAMED、ALL	グラフとそのコンテンツを削除します。削除するグラフの名前を指定するには GRAPH を使用します。デフォルトグラフを削除するには DEFAULT を使用します。
COPY	SILENT、GRAPH、IRIref_from、DEFAULT、TO、IRIref_to	ソースグラフを宛先グラフにコピーします。宛先グラフ内のすべてのコンテンツが上書き（削除）されます。
MOVE	SILENT、GRAPH、IRIref_from、DEFAULT、TO、IRIref_to	ソースグラフのコンテンツを宛先グラフに移動し、ソースグラフからコンテンツを削除します。宛先グラフ内のすべてのコンテンツが上書き（削除）されます。
ADD	SILENT、GRAPH、IRIref_from、DEFAULT、TO、IRIref_to	ソースグラフのコンテンツを宛先グラフに追加します。ADD 操作では、ソースグラフと宛先グラフのどちらのコンテンツも変更されません。

複数のステートメントをセミコロン（;）で区切って 1 つの SPARQL Update 操作内に記述すると、同一トランザクションとして実行されます。また、同じリクエストに含めることができます。各ステートメントは、前のステートメントの結果を参照できることに注意してください。

例えば、次のクエリの COPY 操作では、最初のステートメントで作成されたグラフ <TEST> を参照できます。

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

INSERT DATA
{
  <http://example/book0> dc:title "A default book"
  GRAPH <TEST> {<http://example/book1> dc:title "A new
   book" }
  GRAPH <TEST> {<http://example/book2> dc:title "A second
   book" }
  GRAPH <TEST> {<http://example/book3> dc:title "A third
   book" }
};
COPY <TEST> TO <BOOKS1>
```

注： SPARQL Update の操作は、空のシーケンスを返します。

8.2.1 CREATE

この操作を実行すると、グラフが作成されます。グラフがすでに存在する場合は、SILENT オプションを使用していない限り、エラーが返されます。すでに存在するグラフのコンテンツは変更されません。グラフを作成するパーミッションがない場合は、エラーが返されます（SILENT オプションを使用している場合を除く）。

CREATE 操作のシンタックスは、次のとおりです。

```
CREATE ( SILENT )? GRAPH IRIref
```

SILENT オプションを使用すると、エラーは返されません。GRAPH IRIref オプションでは、新しいグラフの IRI を指定します。

例えば以下のようになります。

```
CREATE GRAPH
<http://marklogic.com/semantics/tutorial/update> ;
```

指定した宛先グラフが存在しない場合は、そのグラフが作成されます。CREATE 操作によって作成されるグラフのパーミッションは、sem:sparql-update で指定したものになります。また、パーミッションを指定していない場合は、そのユーザーのデフォルトパーミッションになります。

8.2.2 DROP

DROP 操作を実行すると、指定したグラフがグラフストアから削除されます。DROP 操作のシンタックスは、次のとおりです。

```
DROP ( SILENT )? GRAPH IRIref | DEFAULT |NAMED | ALL )
```

GRAPH キーワードは、IRIref で指定したグラフを削除する場合に使用し、DEFAULT キーワードオプションは、デフォルトグラフをグラフストアから削除する場合に使用します。また、NAMED キーワードオプションは、すべての名前付きグラフをグラフストアから削除する場合に使用します。ALL キーワードを指定すると、すべてのグラフがグラフストアから削除されます。これは、グラフストアをリセットした場合と同じ結果です。

例えば以下のようになります。

```
DROP SILENT GRAPH
<http://marklogic.com/semantics/tutorial/intro> ;
```

この操作が正常に完了すると、指定したグラフは以降のグラフ更新操作で使用できなくなります。指定した名前付きグラフが存在しない場合は、エラーが返されます。SILENT を指定している場合、操作の結果は常に成功になります。

> **注**： グラフストアのデフォルトグラフを削除すると、ユーザーのデフォルトパーミッションを持つ空のデフォルトグラフが新規に作成されます。

8.2.3 COPY

COPY 操作は、ソースグラフのすべてのトリプルを宛先グラフに挿入する目的で使用します。ソースグラフのトリプルは影響を受けませんが、宛先グラフにトリプルが存在する場合は、新しいトリプルが挿入される前に削除されます。

COPY 操作のシンタックスは、次のとおりです。

```
COPY ( SILENT )? ( ( GRAPH )? IRIref_from | DEFAULT)
   TO ( ( GRAPH )? IRIref_to | DEFAULT )
```

COPY 操作では、パーミッションがソースグラフから宛先グラフにコピーされます。パーミッション情報はソースグラフが持っているため、COPY 操作では sem:sparql-update の $perm パラメータは適用されません。

例えば以下のようになります。

```
COPY <http://marklogic.com/semantics/tutorial/intro> TO
  <http://marklogic.com/semantics/tutorial/start> ;
```

宛先グラフが存在しない場合は作成されます。ソースグラフが存在しない場合は、エラーが返されます。SILENT オプションを使用すると、操作の結果は常に成功になります。

COPY 操作は、グラフを削除して新しいグラフを挿入する操作と同様の操作です。

```
DROP SILENT (GRAPH IRIref_to | DEFAULT); INSERT
   { ( GRAPH IRIref_to )?{ ?s ?p ?o } } WHERE
   { ( GRAPH IRIref_from )? { ?s ?p ?o } }
```

COPY を使用してグラフをそのグラフ自身にコピーしても操作は実行されず、データはそのままの状態になります。

宛先グラフが存在しない場合に更新が失敗となるようにする場合は、sem:sparql-update の existing-graph オプションを指定する必要があります。新しいグラフにコピーすると、その新しいグラフではコピー元のグラフのパーミッションが使用されます。既存のグラフにコピーした場合、そのグラフのパーミッションは変更されません。

8.2.4 MOVE

MOVE 操作は、ソースグラフのすべてのトリプルを宛先グラフへ移動する目的で使用します。MOVE 操作のシンタックスは、次のとおりです。

```
MOVE (SILENT)? ( ( GRAPH )? IRIref_from | DEFAULT)
   TO ( ( GRAPH )? IRIref_to | DEFAULT)
```

ソースグラフは、挿入後に削除されます。宛先グラフにトリプルが存在する場合は、宛先トリプルが挿入される前に削除されます。

例えば以下のようになります。

```
MOVE <http://marklogic.com/semantics/tutorial/queries> TO
  <http://marklogic.com/semantics/tutorialSearches> ;
```

グラフ <http://marklogic.com/semantics/queries> は、そのトリプルが <http://marklogic.com/semantics/searches> に挿入された後で削除されます。グラフ <http://marklogic.com/semantics/searches> 内のすべてのトリプルは、その他のトリプルが挿入される前に削除されます。

注： MOVE を使用してグラフをそのグラフ自身に移動しても操作は実行されず、データはそのままの状態になります。

宛先グラフが存在しない場合は作成されます。ソースグラフが存在しない場合、MOVE 操作からはエラーが返されます。SILENT オプションを使用すると、操作の結果は常に成功になります。

MOVE 操作は、次のようになります。

```
DROP SILENT (GRAPH IRIref_to | DEFAULT); INSERT
   { ( GRAPH IRIref_to )?{ ?s ?p ?o } } WHERE
   { ( GRAPH IRIref_from )? { ?s ?p ?o } };
    DROP ( GRAPH IRIref_from | DEFAULT)
```

MOVE 操作では、パーミッションがソースグラフから宛先グラフに移動されます。パーミッション情報はソースグラフが持っているため、この操作には sem:sparql-update の $perm パラメータは適用されません。

宛先グラフが存在しない場合に更新が失敗となるようにする場合は、sem:sparql-update の existing-graph オプションを指定する必要があります。新しいグラフにコピーすると、その新しいグラフではコピー元のグラフのパーミッションが使用されます。既存のグラフにコピーした場合、そのグラフのパーミッションは変更されません。

8.2.5 ADD

ADD 操作は、ソースグラフのすべてのトリプルを宛先グラフへ挿入する目的で使用します。ソースグラフのトリプルは影響を受けず、宛先グラフに既存のトリプルが存在する場合も変更されません。

ADD 操作のシンタックスは、次のとおりです。

```
ADD ( SILENT )? ( ( GRAPH )? IRIref_from | DEFAULT) TO
( ( GRAPH )? IRIref_to | DEFAULT)
```

例えば以下のようになります。

```
ADD <http://marklogic.com/semantics/tutorial/queries> TO
  <http://marklogic.com/semantics/searches> ;
```

ADD を使用してグラフをそのグラフ自身に追加しても操作は実行されず、データはそのままの状態になります。宛先グラフが存在しない場合は作成されます。ソースグラフが存在しない場合、ADD 操作からはエラーが返されます。SILENT オプションを使用すると、操作の結果は常に成功になります。

トリプルを新しいグラフに追加する場合は、sem:sparql-update を使用して新しいグラフのパーミッションを設定できます。パーミッションを指定していない場合は、デフォルトグラフのパーミッションがそのグラフに適用されます。

ADD 操作は、次のようになります。

```
INSERT { ( GRAPH IRIref_to )? { ?s ?p ?o } } WHERE
    { ( GRAPH IRIref_from )? { ?s ?p ?o } }
```

宛先グラフが存在しない場合に更新が失敗となるようにする場合は、`sem:sparql-update` の `existing-graph` オプションを指定する必要があります。新しいグラフにコピーすると、その新しいグラフではコピー元のグラフのパーミッションが使用されます。既存のグラフにコピーした場合、そのグラフのパーミッションは変更されません。

8.3 グラフレベルのセキュリティ

SPARQL Update を使用すると、グラフレベルでセキュリティを管理できます。グラフレベルのセキュリティとは、グラフに設定されたパーミッションに対応するロールを持つユーザーだけがグラフの表示、グラフコンテンツの変更、または新規グラフの作成を実行できるという意味です。ユーザーは、読み取りパーミッションを（ロール経由で）持つトリプルやグラフだけを表示できます。『*Security Guide*』の「Role-Based Security Model」を参照してください。

デフォルトでは、グラフを作成するユーザーのデフォルトパーミッションでグラフが作成されます。`sem:sparql-update` を呼び出すときの引数としてパーミッションを渡し、グラフパーミッションを指定すると、新規グラフが作成されるグラフ操作およびグラフ管理操作では、その指定されたパーミッションが使用されます。これは、`CREATE GRAPH` を使用して明示的にグラフを作成する場合も、存在しないグラフにトリプルを挿入またはコピー（`INSERT`、またはコピー、移動、追加が発生するグラフ操作）する操作によって暗黙的にグラフが作成される場合も当てはまります。

既存のグラフにトリプルを挿入する際に、`sem:sparql-update` でパーミッションを指定しても無視されます。グラフを別のグラフにコピーする場合、元のデータとパーミッションをグラフ間で「引き継ぐ」ことが優先されるためです。

> **注：** グラフレベルのセキュリティは、XQuery または JavaScript を介して SPARQL または SPARQL Update を使用するすべてのセマンティック操作に適用され、セマンティック REST 関数が含まれます。

デフォルトのユーザーパーミッションは、MarkLogic 管理者が設定します。これは、『*Security Guide*』の「Default Permissions」で説明するドキュメント作成のデフォルトパーミッションと同じです。

グラフの現在のパーミッションを確認するには、次のように `sem:graph-get-permissions` を使用します。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "MarkLogic/semantics.xqy";
```

```
sem:graph-get-permissions(
  sem:iri("MyGraph"))

=>

<sec:permission
xmlns:sec="http://marklogic.com/xdmp/security">
  <sec:capability>read</sec:capability>
  <sec:role-id>5995163769635647336</sec:role-id>
</sec:permission>

<sec:permission
xmlns:sec="http://marklogic.com/xdmp/security">
<sec:capability>update</sec:capability>
<sec:role-id>5995163769635647336</sec:role-id>
</sec:permission>
```

これにより、このグラフに対して設定されている 2 つの権限（read および update）が返されます。ID `5995163769635647336` のロールを持っている場合は、このグラフの情報を読み取ることができ、グラフおよびグラフ内のトリプルを確認できます。また、ID `5995163769635647336` のロールを持っている場合は、グラフを更新できます。`sem:graph-get-permissions` を使用するには、対象のグラフに対して読み取り権限を持っている必要があります。

注： 新しいデータベースには、デフォルトグラフのグラフドキュメントはまだ存在していません。このデータベースにトリプルを挿入すると、デフォルトグラフが作成されます。

グラフのパーミッションを設定するには、`sem:graph-set-permissions` を使用します。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "MarkLogic/semantics.xqy";

declare namespace sec =
"http://marklogic.com/xdmp/security";

sem:graph-set-permissions(sem:iri("MyGraph"),
  (xdmp:permission("sparql-update-role", "update"))
```

これにより、グラフおよびグラフ内のトリプルのパーミッションが設定されます。指定した IRI が存在しない場合、そのグラフが作成されます。パーミッションを設定するには、グラフの更新パーミッションが必要です。

グラフにパーミッションを追加するには、sem:graph-add-permissions を使用します。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "MarkLogic/semantics.xqy";

declare namespace sec = "http://marklogic.com/xdmp/
security";

sem:graph-add-permissions(sem:iri("MyGraph"),
  (xdmp:permission("sparql-read-role", "read"),))
```

これにより、sparql-read-role の読み取りパーミッションが、グラフおよびグラフ内のトリプルに追加されます。IRI で指定されているグラフが存在しない場合、そのグラフが自動的に作成されます。既存のグラフにパーミッションを追加するには、グラフの更新パーミッションが必要です。

注： 管理者以外のユーザー（つまり、admin ロールを持たないユーザー）によって作成されるグラフには、グラフの更新パーミッションが少なくとも 1 つ必要です。更新パーミッションがない場合は、グラフの作成時に XDMP-MUSTHAVEUPDATE 例外が返されます。

パーミッションを削除するには、次のように sem:graph-remove-permissions を使用します。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "MarkLogic/semantics.xqy";

declare namespace sec = "http://marklogic.com/xdmp/
security";

sem:graph-remove-permissions(sem:iri("MyGraph"),
  (xdmp:permission("sparql-read-role", "read"),))
```

これにより、「MyGraph」における sparql-read-role の読み取りパーミッションが削除されます。パーミッションを削除するには、グラフの更新パーミッションが必要です。グラフが存在しない場合、エラーが返されます。

8.4 SPARQL Update によるデータの操作

SPARQL Update では、データ操作として INSERT DATA、DELETE DATA、DELETE/INSERT、LOAD、および CLEAR などを実行できます。データ操作では、グラフに含まれるトリプルデータを操作します。

SPARQL Update には、RDF グラフ内のデータを操作するために、次に示すコマンドとオプションが用意されています。

コマンド	オプション	説明
INSERT DATA	QuadData、WITH、GRAPH	トリプルをグラフに挿入します。グラフを指定していない場合は、デフォルトグラフが使用されます。
DELETE DATA	QuadData	QuadData の指定に従って、トリプルをグラフから削除します。グラフを指定していない場合は、スコープ内のすべてのグラフから削除されます。
DELETE..INSERT WHERE	WITH、IRIref、USING、NAMED、WHERE、DELETE、INSERT	WHERE 節で指定したクエリパターンのバインドに基づき、グラフストアに対してトリプルの削除または追加を実行します。
DELETE WHERE	WITH、IRIref、USING、NAMED、WHERE、DELETE、INSERT	WHERE 節で指定したクエリパターンのバインドに基づき、グラフストアからトリプルを削除します。DELETE WHERE とは、INSERT（オプション）がない DELETE..INSERT WHERE です。
INSERT WHERE	WITH、IRIref、USING、NAMED、WHERE、DELETE、INSERT	WHERE 節で指定したクエリパターンのバインドに基づいて、グラフストアにトリプルを追加します。INSERT WHERE とは、DELETE（オプション）がない DELETE..INSERT WHERE です。
CLEAR	SILENT、（GRAPH IRIref | DEFAULT | NAMED | ALL）	指定したグラフのすべてのトリプルを削除します。

複数のステートメントをセミコロン（;）で区切って 1 つの SPARQL Update 操作内に記述すると、同一トランザクションとして実行されます。また、同じリクエストに含めることができます。各ステートメントは、前のステートメントの結果を参照できることに注意してください。

注： SPARQL Update の操作は、空のシーケンスを返します。

MarkLogic にトリプルをロードするその他の方法としては、mlcp、sem:rdf-load、HTTP REST エンドポイントなどが挙げられます。「mlcp を使用したトリプルのロード」（37 ページ）、「sem:rdf-load」（46 ページ）、「グラフストアの処理」（49 ページ）、および「セマンティックトリプルの読み込み」（29 ページ）を参照してください。一括読み込みでは、mlcp が推奨される手法です。

8.4.1 INSERT DATA

INSERT DATA を実行すると、リクエストで指定されたトリプルがグラフに追加されます。INSERT DATA のシンタックスは、次のとおりです。

```
INSERT DATA QuadData
(GRAPH VarOrIri) ? {TriplesTemplates}
```

「QuadData」パラメータはトリプルパターンの集まり（「TriplesTemplates」）で構成されており、GRAPH ブロックにラップすることもできます。

注： トリプルドキュメントのすべての管理対象トリプルは、SPARQL Update を使用して挿入するときに宛先グラフを指定していない限り、デフォルトグラフに配置されます。

トリプルを新しいグラフに挿入する場合は、sem:sparql-update を使用して指定した目的のパーミッションでグラフを作成します。グラフのパーミッションを指定していない場合、グラフはユーザーのデフォルトパーミッションで作成されます。グラフのパーミッションを管理するには、sem:graph-add-permissions を使用します。パーミッションおよびセキュリティの詳細については、「グラフレベルのセキュリティ」（179 ページ）を参照してください。

例えば次に示す更新操作では、<My Graph> を更新（update）できるように sem:graph-add-permissions を使用して更新パーミッションを sparql-update-role に追加し、3 つのトリプルをそのグラフに挿入しています。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

sem:graph-add-permissions(sem:iri("MyGraph"),
xdmp:permission("sparql-update-role", "update")),
sem:sparql-update('

PREFIX exp: <http://example.org/marklogic/people>
PREFIX pre: <http://example.org/marklogic/predicate>

INSERT DATA {
  GRAPH <MyGraph>{
```

```
    exp:John_Smith pre:livesIn "London" .
    exp:Jane_Smith pre:livesIn "London" .
    exp:Jack_Smith pre:livesIn "Glasgow" .
    }}
')
```

グラフが「QuadData」で記述されていない場合は、デフォルトグラフ（http://marklogic.com/semantics#default-graph）が使用されます。グラフストアに存在しないグラフにデータが挿入されると、そのデータ用の新しいグラフが、ユーザーのパーミッションを使って作成されます。

この例では、INSERT DATA を使用してトリプルをデフォルトグラフに挿入してから、3 つのトリプルを「BOOKS」という名前のグラフに挿入します。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

sem:sparql-update('
PREFIX dc: <http://marklogic.com/dc/elements/1.1/>
INSERT DATA
{
  <http://example/book0> dc:title "A default book"
  GRAPH <BOOKS> {<http://example/book1> dc:title "A new
  book" }
  GRAPH <BOOKS> {<http://example/book2> dc:title "A second
  book" }
  GRAPH <BOOKS> {<http://example/book3> dc:title "A third
  book" }
 }
');
```

この例では、グラフ「BOOKS」内の「A new book」というタイトルの書籍を削除し、その位置に「Inside MarkLogic Server」というタイトルを挿入します。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

sem:sparql-update('
PREFIX dc: <http://marklogic.com/dc/elements/1.1/>
WITH <BOOKS>
DELETE {?b dc:title "A new book"}
INSERT {?b dc:title "Inside MarkLogic Server" }
WHERE {
```

```
?b dc:title "A new book".
}')
```

WITH キーワードは、このグラフ（<BOOKS>）を各節で使用することを意味します。

8.4.2 DELETE DATA

DELETE DATA 操作を実行すると、リクエストで指定したトリプルがグラフストアの各グラフに含まれている場合、それらのトリプルが削除されます。DELETE DATA のシンタックスは、次のとおりです。

```
DELETE DATA QuadData
```

「QuadData」パラメータでは、削除するトリプルを指定します。グラフが「QuadData」で記述されていない場合は、デフォルトグラフが使用されます。

MarkLogic で管理されているトリプルのうち、「QuadData」で指定されたいずれかのトリプルの主語、述語、および目的語にマッチするトリプルが削除されます。グラフを「QuadData」で指定している場合、削除の範囲はそのグラフ内のトリプルに制限されます。指定していない場合、削除の範囲はデフォルトグラフ内のトリプルに制限されます。

この例では、<http://marklogic.com/semantics/COMPANIES100A/> から「true」および「Retail/Wholesale」にマッチするトリプルを削除します。

```
PREFIX demov: <http:demo/verb#>
PREFIX demor: <http:demo/resource#>

DELETE DATA
{
  GRAPH <http://marklogic.com/semantics/COMPANIES100A/>
    {
    demor:COMPANY100 demor:listed "true" .
    demor:COMPANY100 demov:industry "Retail/Wholesale" .
  }
}
```

DELETE DATA 操作に変数または空白ノードが含まれている場合は、エラーがスローされます。存在しないトリプルを削除しても、何も起こりません。グラフストア内に存在しないトリプルを「QuadData」で指定しても無視されます。

8.4.3 DELETE..INSERT WHERE

DELETE..INSERT WHERE 操作は、WHERE 節で指定したクエリパターンのバインドに基づいて、グラフストアに対してトリプルの削除または追加を実行する目的で使用します。DELETE..INSERT WHERE を使用すると、マッチさせるパターンを指定してトリプルを削除または挿入できます。

詳細については、http://www.w3.org/TR/sparql11-update/#updateLanguage を参照してください。

特定のパターンに従ってトリプルを削除するには、オプションの INSERT 節を付けないで DELETE..INSERT WHERE コンストラクトを使用します。グラフを指定していない場合、デフォルトグラフ（http://marklogic.com/semantics#default-graph。名前なしグラフとも呼ばれます）に対してトリプルの挿入または削除が実行されます。DELETE..INSERT WHERE のシンタックスは、次のとおりです。

```
( WITH IRIref )?
( ( DeleteClause InsertClause? ) | InsertClause )
( USING ( NAMED )? IRIref )*
WHERE GroupGraphPattern
DeleteClause ::= DELETE  QuadPattern
InsertClause ::= INSERT  QuadPattern
```

この操作では、変数の集まりについてバインドのマッチングシーケンスを判断するためのデータを、WHERE 節を使用して識別しています。各ソリューションのバインドは、トリプルを削除する場合は DELETE テンプレートに代入され、新しいトリプルを作成する場合は INSERT テンプレートに代入されます。

存在しないグラフに挿入しようとすると、そのグラフが作成されます。DELETE および INSERT は同じトランザクション内で実行され、MarkLogic のセキュリティルールに従います。

この例では、「Healthcare/Life Sciences」を含む各トリプルが削除され、2 つのトリプル（「Healthcare」に 1 つ、「Life Sciences」に 1 つ）が挿入されます。

```
PREFIX demov: <http:demo/verb#>

WITH <http://marklogic.com/semantics/COMPANIES100A/>
DELETE
{
  ?company demov:industry "Healthcare/Life Sciences" .
}
INSERT
{
  ?company demov:industry "Healthcare" .
```

```
    ?company demov:industry "Life Sciences" .
  }
  WHERE {
    ?company demov:industry "Healthcare/Life Sciences" .}
```

DELETE と INSERT は相互に独立しています。

8.4.4 DELETE WHERE

DELETE/INSERT 操作は、WHERE 節で指定したクエリパターンのバインドに基づいて、グラフストアに対してトリプルの削除または追加を実行する目的で使用します。DELETE WHERE とは、INSERT（オプション）がない DELETE..INSERT WHERE です。DELETE WHERE を使用すると、マッチさせるパターンを指定して、マッチしたトリプルを削除できます。

特定のパターンに従ってトリプルを削除するには、オプションの INSERT 節を付けないで DELETE WHERE コンストラクトを使用します。グラフを指定していない場合、デフォルトグラフ（http://marklogic.com/semantics#default-graph。名前なしグラフとも呼ばれます）からトリプルが削除されます。

DELETE WHERE のシンタックスは、次のとおりです。

```
( WITH IRIref )?
( ( DeleteClause InsertClause? ) | InsertClause )
( USING ( NAMED )? IRIref )*
WHERE GroupGraphPattern
DeleteClause ::= DELETE  QuadPattern
InsertClause ::= INSERT  QuadPattern
```

この例の DELETE WHERE では、従業員が 100 人未満の企業に関する売上データをグラフ <http://marklogic.com/semantics/COMPANIES100A/> から削除します。

```
PREFIX demov: <http:demo/verb#>
PREFIX vcard: <http:www.w3c.org/2006/vcard/ns#>

WITH <http://marklogic.com/semantics/COMPANIES100A/>
DELETE
{
  ?company demov:sales ?sales .
}
WHERE {
  ?company a vcard:Organization .
  ?company demov:employees ?employees .
}
  FILTER ( ?employees < 100 )
```

8.4.5 INSERT WHERE

INSERT WHERE 操作は、WHERE 節で指定したクエリパターンのバインドに基づいて、グラフストアにトリプルを追加する目的で使用します。INSERT WHERE は、DELETE（オプション）がない DELETE..INSERT WHERE です。INSERT WHERE を使用すると、マッチさせるパターンを指定して、マッチしたトリプルを挿入できます。

特定のパターンに従ってトリプルを挿入するには、オプションの DELETE 節を付けないで INSERT WHERE コンストラクトを使用します。グラフを指定していない場合は、デフォルトグラフ（http://marklogic.com/semantics#default-graph。名前なしグラフとも呼ばれます）にトリプルが挿入されます。

INSERT WHERE のシンタックスは、次のとおりです。

```
( WITH IRIref )?
( ( DeleteClause InsertClause? ) | InsertClause )
( USING ( NAMED )? IRIref )*
WHERE GroupGraphPattern
DeleteClause ::= DELETE  QuadPattern
InsertClause ::= INSERT  QuadPattern
```

次の INSERT WHERE の例は、米国のニューヨークにある会社を探し、State="NY" と deliveryRegion="East Coast" を追加します。

```
PREFIX demov: <http:demo/verb#>
PREFIX vcard: <http:www.w3c.org/2006/vcard/ns#>

WITH <http://marklogic.com/semantics/sb/COMPANIES100A/>
INSERT
{
  ?company demov:State "NY" .
  ?company demov:deliveryRegion "East Coast"
}
WHERE {
  ?company a vcard:Organization .
  ?company vcard:hasAddress [
    vcard:region "New York" ;
    vcard:country-name "USA"]
}
```

8.4.6 CLEAR

CLEAR 操作を実行すると、指定したグラフのすべてのトリプルが削除されます。CLEAR のシンタックスは、次のとおりです。

```
CLEAR (SILENT)? (GRAPH IRIref | DEFAULT | NAMED | ALL )
```

GRAPH IRIref オプションは、GRAPH IRIref で指定したグラフからすべてのトリプルを削除する目的で使用します。また、DEFAULT オプションは、グラフストアのデフォルトグラフからすべてのトリプルを削除する目的で使用します。NAMED オプションは、グラフストアのすべての名前付きグラフ内にあるすべてのトリプルを削除するために使用し、ALL オプションは、グラフストアのすべてのグラフからすべてのトリプルを削除するために使用します。

例えば以下のようになります。

```
CLEAR GRAPH
<http://marklogic.com/semantics/COMPANIES100A/> ;
```

この操作では、グラフからすべてのトリプルが削除されます。グラフおよびそのメタデータ（パーミッションなど）はグラフの消去後も維持されます。指定したグラフが存在しない場合は、CLEAR が失敗します。SILENT オプションを使用すると、エラーは返されません。

8.5 変数のバインド

変数のバインドは、INSERT DATA および DELETE DATA での値として使用できる他、引数として sem:sparql-update に渡されます。

SPARQL Update で使用する変数のバインドを作成するには、バインドをマッピングする XQuery または JavaScript 関数を作成し、そのバインドを sem:sparql-update の呼び出しの一部として渡します。

この例では、関数を作成して変数を割り当て、バインドの集まりを構築し、そのバインドと INSERT DATA を使用して反復的なデータをトリプルストアに挿入します。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/
  semantics"
  at "/MarkLogic/semantics.xqy";

declare function
local:add-player-triples($id as xs:integer,
    $lastname as xs:string,
    $firstname as xs:string,
    $position as xs:string,
    $team as xs:string,
    $number as xs:integer
  )
{
```

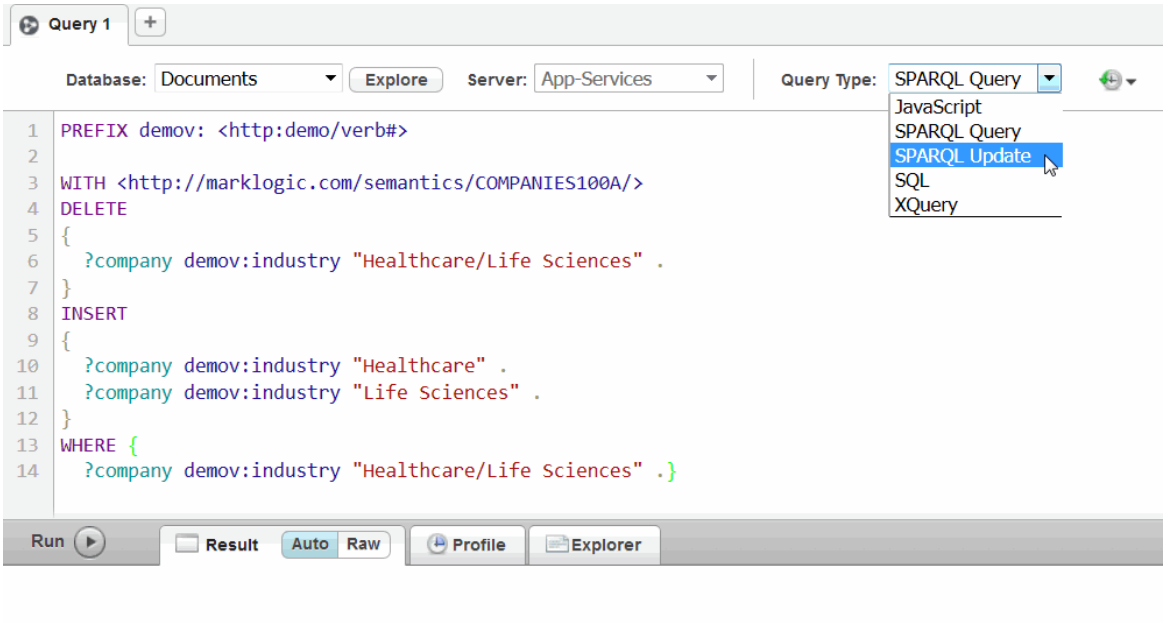
```
let $query := '
PREFIX bb: <http://marklogic.com/baseball/players/>
PREFIX xs: <http://www.w3.org/2001/XMLSchema#>

INSERT DATA
{
GRAPH <PlayerGraph>
{
?playertoken bb:playerid ?id .
?playertoken bb:lastname ?lastname .
?playertoken bb:firstname ?firstname .
?playertoken bb:position ?position .
?playertoken bb:number ?number .
?playertoken bb:team ?team .
}
}

'
let $playertoken := fn:concat("bb:", $id)
let $player-map := map:entry("id", $id)
let $put := map:put($player-map, "playertoken",
$playertoken)
let $put := map:put($player-map, "lastname", $lastname)
let $put := map:put($player-map, "firstname", $firstname)
let $put := map:put($player-map, "position", $position)
let $put := map:put($player-map, "number", $number)
let $put := map:put($player-map, "team", $team)

return sem:sparql-update($update, $player-map)
};

local:add-player-triples(417, "Doolittle", "Sean",
"pitcher", 62, "Athletics"),
local:add-player-triples(215, "Abad", "Fernando",
"pitcher", 56, "Athletics"),
local:add-player-triples(109, "Kazmir", "Scott", "pitcher",
26, "Athletics"),
```

マッピングを使用しているため、変数の順序は重要ではありません。

この同じパターンを `LIMIT` 節および `OFFSET` 節で使用できます。変数のバインドは、SPARQL Update（`sem:sparql-update`）、SPARQL（`sem:sparql`）、および SPARQL value（`sem:sparql-values`）で使用できます。SPARQL で `LIMIT` 節および `OFFSET` 節を使用して変数のバインドを使用する例については、「変数に対するバインドの使用」（132 ページ）を参照してください。

8.6 Query Console での SPARQL Update の使用

SPARQL Update は Query Console で使用でき、Query Console のドロップダウンメニューでクエリタイプとして表示されます。



Query Console では、クエリを作成するときに SPARQL Update キーワードが強調表示されます。SPARQL Update のクエリは、SPARQL クエリを実行するときと同じ方法で、Query Console で実行できます。

8.7 XQuery またはサーバーサイド JavaScript での SPARQL Update の使用

SPARQL Update は、XQuery では `sem:sparql-update`、JavaScript では `sem.sparqlUpdate` を使用して実行できます。関数のシグネチャと説明の詳細については、『*MarkLogic XQuery and XSLT Function Reference*』および『*MarkLogic Server-Side JavaScript Function Reference*』の「XQuery Library Modules」に掲載されているセマンティックに関するドキュメントを参照してください。

> **注：** セマンティック関数にはビルトインのものもそうでないものもあるため、セマンティック API を使用する XQuery モジュールまたは JavaScript モジュールごとにセマンティック API ライブラリをインポートすることをお勧めします。

XQuery を使用する場合、import ステートメントは次のようになります。

```
import module namespace sem = "http://marklogic.com/
  semantics" at "/MarkLogic/semantics.xqy";
```

JavaScript では、import ステートメントは次のようになります。

```
var sem = require("/MarkLogic/semantics.xqy");
```

XQuery で SPARQL Update を使用する複雑なクエリの例を次に示します。このクエリでは、アスレチックス（Athletics）という野球チームの選手のうち、以前はツインズ（Twins）に在籍していた選手をランダムに選択し、その選手をアスレチックスから削除します。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/
semantics"
  at "/MarkLogic/semantics.xqy";

let $query := '
PREFIX bb:  <http://marklogic.com/baseball/players#>
PREFIX bbr:  <http://marklogic.com/baseball/rules#>
PREFIX xs: <http://www.w3.org/2001/XMLSchema#>

WITH <Athletics>
DELETE
{
  ?s ?p ?o .
}
INSERT
{
  ?s ?p bbr:Twins .
}
WHERE
  {
    ?s ?p ?o .
    {
      SELECT (max(?s1) as ?key) (count(?s1) as
     ?inner_numbers)
      WHERE
     {
       ?s1 bb:number ?o1 .
     }
    }
FILTER (?s = ?key)
FILTER (?p = bb:team)
}
return sem:sparql-update($query)
```

8.8 REST での SPARQL Update の使用

SPARQL Update をクライアントアプリケーションで REST と併用すると、SPARQL エンドポイント経由でグラフおよびトリプルデータを管理できます。SPARQL Update を REST と併用する方法については、「REST クライアント API を使用した SPARQL Update」（222 ページ）を参照してください。

SPARQL を Node.js クライアントアプリケーションと併用する方法の詳細については、『*Node.js Application Developer's Guide*』の Working With Semantic Data を参照してください。Java を使用したセマンティッククライアントアプリケーションの場合は、GitHub（http://github.com/marklogic/java-client-api）で Java Client API を見つけるか、Maven のセントラルレポジトリから Java Client API を入手してください。

9.0 REST クライアント API でセマンティックを使用する

このセクションでは、REST クライアント API で MarkLogic のセマンティックを使用して、REST (Representational State Transfer) over HTTP でトリプルおよびグラフの表示、クエリ、および修正する方法について説明します。REST クライアント API を使用すると、クライアントアプリケーションで SPARQL クエリや更新を実行できるようになります。MarkLogic でのトリプルへのアクセスには、SPARQL クエリや SPARQL Update と使用できる MarkLogic SPARQL エンドポイント（`/v1/graphs/sparql`）が使用可能です。`/v1/graphs/sparql` サービスは、SPARQL 1.1 プロトコル（http://www.w3.org/TR/2013/REC-sparql11-protocol-20130321/#terminology）によって定義されているとおり、これに準拠した SPARQL エンドポイントです。クライアントで SPARQL Update または Query エンドポイントの設定が必要な場合は、このサービスを使用してください。

SPARQL endpoint は、SPARQL protocol を実装する web サービスであり、SPARQL クエリに応答できます。SPARQL エンドポイントを使用して web に公開した RDF データをクエリすると、すべてのデータをダウンロードする代わりに特定の質問に回答できます。SPARQL エンドポイント経由で標準的なクエリや更新を行うアプリケーションがある場合は、アプリケーションにこのエンドポイントをポイントさせることができます。

SPARQL エンドポイント URL は、次のようにアドレス指定されます。

```
http://host:port/v1/graphs/sparql
```

`graph` エンドポイントは、グラフでの CRUD 手順（グラフの作成、読み取り、更新、および削除）に使用します。URL は、次のようにアドレス指定されます。

```
http://host:port/v1/graphs
```

`things` エンドポイントは、データベース内のコンテンツを表示するために使用されます。URL は、次のようにアドレス指定されます。

```
http://host:port/v1/graphs/things
```

次の表に、セマンティックに使用でき、サポートされている操作（コンテンツの表示、クエリ、挿入または削除）を示します。

操作	メソッド	説明
`/v1/graphs/sparql`		
取得	GET	データベースに対して SPARQL クエリを実行します。

操作	メソッド	説明
作成 / 取得	POST	1 つあるいは複数のグラフに対して SPARQL クエリや SPARQL Update を実行します（これらの 2 つの操作は相互排他的です）。
/v1/graphs		
取得	GET	グラフのコンテンツまたはパーミッションメタデータ、あるいは利用できるグラフ URI のリストを取得します。
マージ	POST	N-quads をマージしてトリプルストアに入れるか、または他のタイプのトリプルや新しいパーミッションをマージして名前付きグラフまたはデフォルトグラフに入れます。
作成 / 置換	PUT	トリプルストアのクアッドを作成または置換するか、名前付きグラフまたはデフォルトグラフの、他の種類のトリプルを作成または置換するか、名前付きグラフまたはデフォルトグラフのパーミッションを置換します。
（結果を）返す	HEAD	/graphs サービスでこれに相当する GET と同じヘッダを返します。
削除	DELETE	名前付きグラフまたはデフォルトグラフのトリプルを削除するか、トリプルストアからすべてのグラフを削除します。
/v1/graphs/things		
取得	GET	データベースにあるすべてのグラフノードのリスト、または指定したノードの集まりを取得します。

サービスの使用方法およびパラメータの詳細については、『REST Client APIs for Semantics』を参照してください。

この章は、次のセクションで構成されています。

SPARQL クエリで SPARQL エンドポイントまたはグラフエンドポイントを使用するには、ご使用の環境のセキュリティ要件を満たすとともに、rest-reader 権限が必要になります。SPARQL エンドポイントまたはグラフエンドポイントで SPARQL Update を使用するには、rest-writer 権限が必要になります。パーミッションの詳細については、『*REST Application Developer's Guide*』の「Controlling Access to Documents Created with the REST API」を参照してください。

9.1 前提

セクションの以下の各例は、次のことを前提として説明されています。

- GovTrack データセットにアクセスできる。詳細については、「例を実行する準備」（125 ページ）を参照してください。GovTrack データにアクセスできない場合、または独自のデータを使用する場合は、使用するデータに合わせてクエリを修正できます。
- HTTP リクエストを発行する `curl`、またはそれに相当するコマンドラインツールがインストールされている。

  注： 各例では `curl` を利用していますが、HTTP リクエストを送信できるツールであればどのようなものでも使用できます。`curl` の使い方がわからない、またはシステムに `curl` がない場合は、『*REST Application Developer's Guide*』の「Introduction to the curl Tool」を参照してください。

9.2 パラメータの指定

REST サービスには各種のパラメータが使用できます。完全なリスト（POST:/v1/graphs/sparql など）については、『REST Client APIs』をご覧ください。このセクションでは、SPARQL クエリおよび / または SPARQL Update で使用できるパラメータについて説明します。

パラメータについては、「*」も「?」も、そのパラメータがオプションであることを示します。「*」は、そのパラメータを 0 回以上使用できるという意味です。「?」は、そのパラメータを 0 回または 1 回使用できるという意味です。

9.2.1 SPARQL クエリのパラメータ

POST:/v1/graphs/sparql または GET:/v1/graphs/sparql を使用する、SPARQL エンドポイントに対する SPARQL クエリについてサポートされているパラメータのいくつかを次に示します。

- query - 実行する SPARQL クエリ
- default-graph-uri* - クエリ操作でデフォルトグラフとして使用するグラフ（複数可）の URI です。これは、http://host:port/v1/graphs/sparql?*default-graph-uri=<default-graph-uri*>* のようにアドレス指定されます。
- named-graph-uri* - クエリ操作に含めるグラフ（複数）の URI です。http://host:port/v1/graphs/sparql?*named-graph-uri=<named-graph-uri*>* のようにアドレス指定されます。

  「*」は、1 つあるいは複数の default-graph-uri または named-graph-uri パラメータを指定できることを示します。「named-graph-uri」パラメータは、クエリで FROM NAMED および GRAPH とともに使用して、特定の種類のクエリ内で名前に置換する IRI を指定します。クエリの一部として 1 つあるいは複数の *named-graph-uri* を指定できます。

- database? - クエリを実行するデータベースです。
- base? - クエリ用の初期ベース IRI です。
- bind:{name}* - バインドの名前と値です。この形式は、バインド変数のタイプが IRI であることを前提としています。
- bind:{name}:{type}* - バインドの名前、タイプ、および値です。このパラメータには、「string」、「date」、「unsignedLong」など、XSD タイプが利用できます。
- bind:{name}@{lang}* - バインドの名前、言語タグ、および値です。言語タグ付きの文字列にバインドするには、このパターンを使用してください。

- txid? - このリクエストを処理するマルチステートメントトランザクションのトランザクション識別子です。マルチステートメントトランザクションを作成および管理するには、/transactions サービスを使用してください。
- start? - 1 番目に返す結果のインデックスです。結果は 1 から番号付けされます。デフォルトは 1 です。
- pageLength? - このリクエストで返す結果の最大数です。

オプションであるこれらの検索クエリパラメータを使用すると、SPARQL クエリで検索するドキュメントを制限できます。

- q? - 文字列のクエリです。
- structuredQuery? - 構造化された検索クエリ文字列です。search:query 要素のシリアライズされた表現です。
- options? - クエリオプションの名前です。

クエリでグラフ名を指定していない場合は、すべてのグラフの和集合がクエリされます。default-graph-uri を指定すると、指定した 1 つあるいは複数のグラフ名がクエリされます（これは、名前なしトリプルが含まれている「デフォルト」グラフではありません）。名前なしトリプルが格納されている http://marklogic.com/semantics#default-graph をクエリすることもできます。

このようなグラフ名には有効な IRI を使用できます（例えば、/my_graph/ または http://www.example.com/rdf-graph-store/）。default-graph-uri は、操作の一部としてクエリする 1 つあるいは複数のグラフを指定するのに使用され、named-graph-uri は、操作で使用する 1 つあるいは複数の追加グラフを指定できます。データセットが定義されていない場合、データセットにはすべてのトリプル（すべてのグラフの和集合）が含まれます。

リクエストパラメータとクエリの両方でデータセットを指定した場合、named-graph-uri または default-graph-uri で定義されたデータセットが優先されます。REST クライアント API 経由で、1 つのクエリに複数の default-graph-uri または named-graph-uri を指定する場合の形式は、クエリ内の名前付きグラフごとに http://host:port/v1/graphs/sparql?*named-graph-uri*=<*named-graph-uri**> になります。

例えば、次に示すのは、bills.sparql ファイルの SPARQL クエリを送信し、結果を JSON で返すシンプルな REST リクエストです。

```
curl --anyauth --user admin:admin -i -X POST \
--data-binary @./bills.sparql \
-H "Content-type: application/sparql-query" \
-H "Accept: application/sparql-results+json" \
```

```
http://localhost:8000/v1/graphs/sparql

=>
HTTP/1.1 200 OK
Content-type: application/sparql-results+json; charset=UTF-8
Server: MarkLogic
Content-Length: 1268
Connection: Keep-Alive
Keep-Alive: timeout=5

{"head":{"vars":["bill","title"]},
"results":{"bindings":[
   {"bill":{"type":"uri",
   "value":"http://www.rdfabout.com/rdf/usgov/congress/108/
            bills/h1171"},
   "title":{"type":"literal", "value":"H.R.108/1171: Iris
             Scan Security Act of 2003",
   "datatype":"http://www.w3.org/2001/XMLSchema#string"}},
   {"bill":{"type":"uri",
   "value":"http://www.rdfabout.com/rdf/usgov/congress/108/
            bills/h1314"},
   "title":{"type":"literal", "value":"H.R.108/1314:
           Screening Mammography Act of 2003",
   "datatype":"http://www.w3.org/2001/XMLSchema#string"}},
   {"bill":{"type":"uri",
   "value":"http://www.rdfabout.com/rdf/usgov/congress/108/
            bills/h1384"},
   "title":{"type":"literal", "value":"H.R.108/1384: To amend
           the Railroad Retirement Act of 1974 to
           eliminate a limitation on benefits.",
   "datatype":"http://www.w3.org/2001/XMLSchema#string"}},
   {"bill":{"type":"uri",
   "value":"http://www.rdfabout.com/rdf/usgov/congress/108/
            bills/h1418"},
   "title":{"type":"literal", "value":"H.R.108/1418:
             Veterans' Housing Equity Act",
   "datatype":"http://www.w3.org/2001/XMLSchema#string"}},
...
}]}}
```

注： 上記のコマンドライン例では、長い行は UNIX の行継続文字「\」で複数の行に分割しています。また、改行を追加しています。結果では、読みやすくするための改行を追加しています。

9.2.2 SPARQL Update のパラメータ

クエリパラメータに加え、SPARQL Update では `POST:/v1/graphs/sparql` エンドポイントに対して次のパラメータも使用できます。

- `update` - URL エンコードされた SPARQL Update 操作です。リクエストパラメータをリクエストのボディに置く場合に限ってこのパラメータを使用し、`application/x-www-form-urlencoded` をリクエストのコンテンツタイプとして使用してください。
- `using-graph-uri*` - SPARQL Update 操作の一部としてアドレス指定するグラフ（複数可）の URI です。`http://host:port/v1/graphs/sparql?using-graph-uri=<using-graph-uri*>` のようにアドレス指定されます。
- `using-named-graph-uri*` - SPARQL Update 操作の一部としてアドレス指定する名前付きグラフ（複数可）の URI です。`http://host:port/v1/graphs/sparql?using-named-graph-uri=<using-named-graph-uri*>` のようにアドレス指定されます。
- `perm:{role}*` - 挿入したグラフにパーミッションを割り当てます。パーミッションには、ロールと機能があります。新しいグラフを挿入する場合は、特定ロールの特定機能を許可するようにそのパーミッションを設定できます。パーミッションとして有効な値：`read`、`update`、`execute`。パーミッションが適用されるのは、新しく作成されたグラフのみです。パーミッションの詳細については、「グラフパーミッションの管理」（230 ページ）を参照してください。
- `txid?` - このリクエストを処理するマルチステートメントトランザクションのトランザクション識別子です。マルチステートメントトランザクションを作成および管理するには、`/transactions` サービスを使用してください。
- `database?` - クエリを実行するデータベースです。
- `base?` - クエリ用の初期ベース IRI です。
- `bind:{name}*` - バインドの名前と値です。この形式は、バインド変数のタイプが IRI であることを前提としています。
- `bind:{name}:{type}*` - バインドの名前、タイプ、および値です。このパラメータには、「string」、「date」、「unsignedLong」など、XSD タイプが利用できます。
- `bind:{name}@{lang}*` - バインドの名前、言語タグ、および値です。言語タグ付きの文字列にバインドするには、このパターンを使用してください。

グラフの詳細については、「ターゲット RDF グラフ」（82 ページ）を参照してください。RDF トリプルで REST クライアント API を使用する方法の詳細については、『*REST Application Developer's Guide*』の「Querying Triples」を参照してください。

9.3 REST クライアント API に対してサポートされている操作

/v1/graphs/sparql エンドポイントに対しては、次の操作がサポートされています。

操作	説明	メソッド	権限
取得	SPARQL クエリを評価して、名前付きグラフを取得します。	GET	rest-reader
作成 / 取得	パラメータとしての SPARQL クエリ、または POST のボディの一部として URL エンコードされた SPARQL クエリを評価します。 SPARQL Update を使用すると、パラメータとして使用する場合、または URL エンコードされた SPARQL Update として POST のボディで使用する場合は、POST によってトリプルが名前付きグラフにマージされます。 SPARQL クエリと SPARQL Update の操作は相互排他的です。	POST	rest-writer（SPARQL クエリ） rest-writer（SPARQL Update）

注： SPARQL Update の場合、POST:/v1/graphs/sparql のみがサポートされています。

また、/v1/graphs エンドポイントと、RDF データのアクセスおよび表示を行うための /v1/graphs/things エンドポイントもあります。/v1/graphs エンドポイントの場合、次の動詞がサポートされています。

操作	説明	メソッド	権限
取得	グラフ、または利用できるグラフ URI のリストを取得します。	GET	rest-reader
マージ	パラメータを指定しなかった場合、クアッドがマージされてトリプルストアに入ります。graph または default を指定した場合、トリプルがマージされて名前付きグラフまたはデフォルトグラフに入ります。	POST	rest-writer

操作	説明	メソッド	権限
作成 / 置換	パラメータを指定しなかった場合、クアッドが作成または置換されます。`default` または `graph` を指定した場合、名前付きグラフまたはデフォルトグラフのトリプルが作成または置換されます。`PUT` に空のグラフを指定して使用すると、グラフが削除されます。	PUT	rest-writer
削除	パラメータを指定しなかった場合、すべてのグラフがトリプルストアから削除されます。`graph` または `default` を指定した場合、名前付きグラフまたはデフォルトグラフのトリプルが削除されます。	DELETE	rest-writer
（結果を）返す	`/graphs` サービスでこれに相当する GET と同じヘッダを返します。	HEAD	rest-reader

また、`/v1/graphs/things` エンドポイントの場合、REST リクエストに対して次のような動詞 `GET` がサポートされています。

操作	説明	メソッド	権限
取得	データベースにあるすべてのグラフノードのリスト、または指定したノードの集まりを取得します。	GET	rest-reader

9.4 シリアライゼーション

RDF データの読み込み、クエリ、更新が行われるたびに、データのシリアライゼーションが発生します。データはさまざまな方法でシリアライズできます。サポートされているシリアライゼーションは、「サポートされている RDF トリプルの形式」（31 ページ）に示されている表で示しています。

いくつかのタイプの最適化されたシリアライゼーションが、SPARQL の結果（ソリューション - バインドの集まり）および RDF（トリプル）over REST で利用できます。インタクラクションでこれらのシリアライゼーションを使用すると、シリアライゼーションが高速化されます。これらのシリアライゼーションは、入力および出力形式に MIME タイプを指定します。形式は、REST リクエストの Accept ヘッダの一部として指定します。

/v1/graphs/sparql エンドポイントを使用する場合、SPARQL の結果に使用する最適化されたシリアライゼーションには、次のいずれかを使用することをお勧めします。

形式	SPARQL クエリタイプ /v1/graphs/sparql	MIME タイプ /Accept ヘッダ
json	SELECT または ASK	application/sparql-results+json
csv	SELECT または ASK	application/sparql-results+csv
n-triples	CONSTRUCT または DESCRIBE	application/n-triples

CONSTRUCT または DESCRIBE の場合、サポートされているトリプルの全形式がサポートされています。「サポートされている RDF トリプルの形式」（31 ページ）の表を参照してください。

注： N-Quads 形式と TriG 形式はクアッド形式です。トリプル形式ではありません。REST はクアッドをシリアライズしません。

/v1/graphs エンドポイントを使用する場合は、最適化された RDF 結果（トリプルまたはクアッド）には、次のシリアライゼーションオプションのいずれかを選択します。

形式	RDF クエリタイプ /v1/graphs	MIME タイプ /Accept ヘッダ
n-triples	CONSTRUCT または DESCRIBE	application/n-triples
n-quads	SELECT または ASK	application/quads

この情報は、「SPARQL クエリタイプと出力形式」（207 ページ）では異なる方法で記載されています。シリアライゼーションの詳細については、『*Administrator's Guide*』の「Setting Output Options for an HTTP Server」を参照してください。

9.4.1 サポートされていないシリアライゼーション

サポートされていないシリアライゼーションの応答に対して GET または POST リクエストを実行すると、「406 Not Acceptable」エラーが発生します。また、SPARQL ペイロードのパースで障害が発生すると、応答によって「400 Bad Request」エラーが発生します。

例えば以下のようになります。

```
<rapi:error xmlns:rapi="http://marklogic.com/rest-api">
  <rapi:status-code>400</rapi:status-code>
  <rapi:status>Bad Request</rapi:status>
  <rapi:message-code>RESTAPI-INVALIDCONTENT</rapi:
message-code>
  <rapi:message>RESTAPI-INVALIDCONTENT: (err:FOER0000)
Invalid content:
  Unexpected Payload: c:\space\example.ttl</rapi:message>
</rapi:error>
```

REST クライアント API のエラー処理の詳細については、『*REST Application Developer's Guide*』の Error Reporting を参照してください。

9.5 curl および REST を使用したシンプルな例

ここで示す 2 つの例では、REST を使用してセマンティッククエリを行うため、cygwin（Linux）および Windows で `curl` を使用しています。例では、次の SPARQL クエリがエンコードされて使用されています。

```
SELECT *
WHERE {
 ?s ?p ?o }
```

cygwin（Linux）の例では、読みやすくするために文字「\」を使用して改行を示しています。実際のリクエストは 1 つの連続した行で入力する必要があります。クエリは次のようになります。

```
curl --anyauth --user user:password
"http://localhost:8000/v1/graphs/sparql" \
-H "Content-type:application/x-www-form-urlencoded" \
-H "Accept:application/sparql-results+xml" \
-X POST --data-binary 'query=SELECT+*+WHERE+{+%3fs+%3fp+%3fo+}'

=>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
<head>
  <variable name="s"/>
  <variable name="p"/>
  <variable name="o"/>
</head>
<results>
  <result>
    <binding
```

```
name="s"><uri>http://example/book1/</uri></binding>
 <binding
name="p"><uri>http://purl.org/dc/elements/1.1/title</uri></b
inding>
 <binding name="o"><literal>A new book</literal></binding>
  </result>
  <result>
    <binding name="s">
      <uri>http://example/book1/</uri>
    </binding>
    <binding name="p">
      <uri>http://purl.org/dc/elements/1.1/title</uri>
    </binding>
    <binding name="o">
      <literal>Inside MarkLogic Server</literal>
    </binding>
  </result>
  <result>
    <binding name="s">
      <uri>http://www.w3.org/2000/01/rdf-schema#subClassOf</uri>
    </binding>
    <binding name="p">
      <uri>http://www.w3.org/2000/01/rdf-schema#domain</uri>
    </binding>
    <binding name="o">
      <uri>http://www.w3.org/2000/01/rdf-schema#Class</uri>
    </binding>
  </result>
  <result>
    <binding name="s">
      <uri>http://www.w3.org/2000/01/rdf-schema#subClassOf</uri>
    </binding>
    <binding name="p">
      <uri>http://www.w3.org/2000/01/rdf-schema#range</uri>
    </binding>
    <binding name="o">
      <uri>http://www.w3.org/2000/01/rdf-schema#Class</uri>
    </binding>
  </result>
</results></sparql>
```

注： わかりやすくするため、結果の書式を整えています。

Windows の例では、読みやすくするために文字「^」を使用して改行を示しています。実際のリクエストは 1 つの連続した行で入力する必要があります。Windows のクエリは、次のようになります。

```
curl --anyauth --user user:password
"http://localhost:8000/v1/graphs/sparql" ^
  -H "Content-type:application/x-www-form-urlencoded" ^
  -H "Accept:application/sparql-results+xml" ^
  -X POST --data-binary 'query=SELECT+*+WHERE+{+%3fs+%3fp+%3fo+}'
=>

<sparql xmlns="http://www.w3.org/2005/sparql-results#">
<head>
  <variable name="s"/>
  <variable name="p"/>
  <variable name="o"/>
</head>
<results>
 <result>
   <binding name="s">
    <uri>http://example.org/marklogic/people/Jack_Smith</uri>
   </binding>
   <binding name="p">
    <uri>http://example.org/marklogic/predicate/livesIn</uri>
   </binding>
   <binding name="o"><literal>Glasgow</literal>
   </binding>
 </result>
 <result>
   <binding name="s">
    <uri>http://example.org/marklogic/people/Jane_Smith</uri>
   </binding>
   <binding name="p">
    <uri>http://example.org/marklogic/predicate/livesIn</uri>
   </binding>
   <binding name="o">
     <literal>London</literal>
   </binding>
 </result>
 <result>
   <binding name="s">
    <uri>http://example.org/marklogic/people/John_Smith</uri>
   </binding>
   <binding name="p">
    <uri>http://example.org/marklogic/predicate/livesIn</uri>
   </binding>
```

```
      <binding name="o">
        <literal>London</literal>
      </binding>
    </result>
  </results></sparql>
```

注： わかりやすくするため、結果の書式を整えています。

9.6 応答の出力形式

このセクションでは、REST クライアント API で SPARQL エンドポイントを使用するときに使用できるヘッダタイプおよび応答の出力形式について説明します。さまざまな形式での結果の例を記載しています。次のトピックで構成されています。

- SPARQL クエリタイプと出力形式
- 例：XML として結果を返す
- 例：JSON として結果を返す
- 例：HTML として結果を返す
- 例：CSV として結果を返す
- 例：CSV として結果を返す
- 例：XML または JSON としてブール型の値を返す

9.6.1 SPARQL クエリタイプと出力形式

REST クライアント API（`GET:/v1/graphs/sparql` または `POST:/v1/graphs/sparql`）で SPARQL エンドポイントをクエリするときは、結果の出力形式を指定できます。応答タイプの形式は、HTTP Accept ヘッダの MIME type およびクエリタイプに応じて異なります。

SPARQL の `SELECT` クエリは、結果を XML、JSON、HTML、または CSV として返すことができます。一方、SPARQL の `CONSTRUCT` クエリは、N-Triples 形式または N-Quads 形式のトリプルとして、あるいはサポートされている任意のトリプル形式の XML または JSON トリプルとして結果を返すことができます。SPARQL の `DESCRIBE` クエリは、クエリによって見つかったトリプルについて記述するトリプルを XML 形式、N-Triples 形式、または N-Quads 形式で返します。SPARQL の `ASK` クエリを使用すると、ブール型の値（`true` または `false`）が XML または JSON で返されます。
クエリタイプの詳細については、「SPARQL クエリのタイプ」（73 ページ）を参照してください。

次の表では、さまざまなタイプの SPARQL クエリの MIME タイプ、および Accept ヘッダ / 出力形式（MIME タイプ）について説明します。

クエリタイプ	形式	Accept ヘッダ MIME タイプ
SELECT または ASK SPARQL の結果（ソリューション）を返します	xml	application/sparql-results+xml 詳細は「例：XML として結果を返す」（209 ページ）および「例：XML または JSON としてブール型の値を返す」（215 ページ）を参照してください。
	json	application/sparql-results+json 「例：JSON として結果を返す」（211 ページ）を参照してください。
	html	text/html 「例：HTML として結果を返す」（212 ページ）を参照してください。
	csv	text/csv 「例：CSV として結果を返す」（213 ページ）を参照してください。 **注**：維持するのは結果の順序のみです。タイプは維持しません。
	注：ASK クエリは、ブール型の値（true または false）を返します。	
CONSTRUCT または DESCRIBE RDF トリプルを返します	n-triples	application/n-triples シリアライゼーションの高速化については、「例：N-triples として結果を返す」（214 ページ）を参照してください。 **注**：トリプルを JSON 形式で返させる場合、適切な MIME タイプは application/rdf+json です。
	その他	CONSTRUCT または DESCRIBE クエリは、利用できる任意の形式で RDF トリプルを返します。「サポートされている RDF トリプルの形式」（31 ページ）を参照してください。

注： 任意のトリプル MIME タイプ（application/rdf+xml、text/turtle など）をリクエストできますが、最適なパフォーマンスを得るには application/n-triples を使用してください。詳細については、「シリアライゼーション」（202 ページ）を参照してください。

次の例では、この SPARQL の SELECT クエリを使用して、Robert Andrews（「A000210」）が提出者になっていた米国議会法案を調べます。

```
#filename bills.sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX bill: <http://www.rdfabout.com/rdf/schema/usbill/>
PREFIX people: <http://www.rdfabout.com/rdf/usgov/
congress/people/>PREFIX dc:
<http://purl.org/dc/elements/1.1/>

SELECT  ?bill ?title
WHERE { ?bill rdf:type bill:HouseBill ;
           dc:title ?title ;
           bill:sponsor people:A000210 .
    }
    LIMIT 5
```

SPARQL クエリは、bills.sparql として保存されます。クエリでは、応答が 5 つの結果までに制限されます。curl と REST クライアント API を使用すると、SPARQL エンドポイントをクエリし、結果をさまざまな形式で得ることができます。

注： curl を使用して PUT または POST リクエストを実行し、ファイルからリクエストのボディを読み取る場合は、-d ではなく --data-binary を使用して入力ファイルを指定してください。--data-binary を使用すると、curl によってファイルからリクエストのボディにデータがそのまま挿入されます。-d を使用すると、curl によって入力から改行が取り除かれるため、トリプルデータや SPARQL がシンタックスの面で無効になる可能性があります。

9.6.2 例：XML として結果を返す

次の例では、bills.sparql ファイルの SPARQL SELECT クエリが XML 形式で応答を返します。

```
curl --anyauth --user admin:password -i -X POST \
--data-binary @./bills.sparql \
-H "Content-type: application/sparql-query" \
-H "Accept: application/sparql-results+xml" \
http://localhost:8050/v1/graphs/sparql

=>

HTTP/1.1 200 OK
Content-type: application/sparql-results+xml
Server: MarkLogic
```

```
Content-Length: 1528
Connection: Keep-Alive
Keep-Alive: timeout=5

<sparql xmlns="http://www.w3.org/2005/sparql-results/">
 <head><variable name="bill"/>
       <variable name="title"/>
 </head>
 <results>
   <result>
     <binding name="bill">
      <uri>http://www.rdfabout.com/rdf/usgov/congress/108/
       bills/h1171
      </uri>
     </binding><binding name="title">
      <literal datatype="http://www.w3.org/2001/
      XMLSchema#string">
       H.R.108/1171: Iris Scan Security Act of 2003
       </literal>
     </binding>
   </result>
   <result>
     <binding name="bill">
      <uri>http://www.rdfabout.com/rdf/usgov/congress/108/
      bills/h1314/
      </uri>
     </binding>
     <binding name="title">
      <literal
datatype="http://www.w3.org/2001/XMLSchema#string">
       H.R.108/1314: Screening Mammography Act of 2003
     </literal>
     </binding>
   </result>
   <result>
     <binding name="bill"><uri>http://www.rdfabout.com/
    rdf/usgov
     /congress/108/bills/h1384/</uri>
     </binding>
...
   </result>
 </results>
</sparql>
```

注： 上記の例では、長い行は UNIX の行継続文字「\」で複数の行に分割しています。また、改行を追加しています。結果では、読みやすくするための改行も追加しています。

9.6.3 例：JSON として結果を返す

次の例では、bills.sparql ファイルの SPARQLSELECT クエリが JSON 形式で応答を返します。

```
curl --anyauth --user admin:password -i -X POST \
--data-binary @./bills.sparql \
-H "Content-type: application/sparql-query" \
-H "Accept: application/sparql-results+json" \
http://localhost:8050/v1/graphs/sparql

=>

HTTP/1.1 200 OK
Content-type: application/sparql-results+json
Server: MarkLogic
Content-Length: 1354
Connection: Keep-Alive
Keep-Alive: timeout=5

{"head":{"vars":["bill","title"]},
"results":{"bindings":[
   {"bill":{"type":"uri",
   "value":"http://www.rdfabout.com/rdf/usgov/congress/
           108/bills/h1171"},
   "title":{"type":"literal", "value":"H.R.108/1171:
   Iris Scan
            Security Act of 2003",
   "datatype":"http://www.w3.org/2001/XMLSchema#string"}},
   {"bill":{"type":"uri",
  "value":"http://www.rdfabout.com/rdf/usgov/congress/108/
           bills/h1314"},
   "title":{"type":"literal", "value":"H.R.108/1314:
Screening
            Mammography Act of 2003",
   "datatype":"http://www.w3.org/2001/XMLSchema#string"}},
   {"bill":{"type":"uri",
  "value":"http://www.rdfabout.com/rdf/usgov/congress/108/
           bills/h1384"},
   "title":{"type":"literal", "value":"H.R.108/1384: To
  amend the Railroad Retirement Act of 1974 to eliminate
  a limitation on benefits.",
   "datatype":"http://www.w3.org/2001/XMLSchema#string"}},
   {"bill":{"type":"uri",
   "value":"http://www.rdfabout.com/rdf/usgov/congress/108/
           bills/h1418"},
```

```
   "title":{"type":"literal", "value":"H.R.108/1418:
  Veterans'
           Housing Equity Act",
   "datatype":"http://www.w3.org/2001/XMLSchema#string"}},
...
}]}}
```

注： 上記のコマンドライン例では、長い行は UNIX の行継続文字「\」で複数の行に分割しています。また、改行を追加しています。結果では、読みやすくするための改行も追加しています。

9.6.4 例：HTML として結果を返す

次の例では、`bills.sparql` の同じ SPARQL`SELECT` クエリが HTML 形式で応答を返します。

```
curl --anyauth --user admin:password -i -X POST \
--data-binary @./bills.sparql \
-H "Content-type: application/sparql-query" \
-H "Accept: text/html" http://localhost:8050/v1/graphs/sparql"

=>

HTTP/1.1 200 OK
Content-type: text/html; charset=UTF-8
Server: MarkLogic
Content-Length: 1448
Connection: Keep-Alive
Keep-Alive: timeout=5

<html xmlns="http://www.w3.org/1999/xhtml">
 <head>
  <title>SPARQL results</title>
 </head>
 <body><table border="1">
 <tr>
  <th>bill</th>
  <th>title</th></tr>
 <tr>
  <td><a href="/v1/graphs/things?iri=http%3a//
  www.rdfabout.com/
    rdf/usgov/congress/108/bills/h1171">http://
   www.rdfabout.com/
    rdf/usgov/congress/108/bills/h1171</a>
  </td>
  <td>H.R.108/1171: Iris Scan Security Act of 2003</td>
```

```
 </tr><tr>
  <td><a href="/v1/graphs/things?iri=http%3a//
  www.rdfabout.com/
    rdf/usgov/congress/108/bills/h1314">
    http://www.rdfabout.com/
    rdf/usgov/congress/108/bills/h1314</a>
  </td>
  <td>H.R.108/1314: Screening Mammography Act of 2003</td>
 </tr><tr>
  <td><a href="/v1/graphs/things?iri=http%3a//
  www.rdfabout.com/
    rdf/usgov/congress/108/bills/h1384">http://
    www.rdfabout.com/
    rdf/usgov/congress/108/bills/h1384</a>
  </td>
  <td>H.R.108/1384: To amend the Railroad Retirement Act of
    1974 to eliminate a limitation on benefits.
  </td>
 </tr>
...
 </table>
 </body>
</html>
```

注： 上記の例では、長い行は UNIX の行継続文字「\」で複数の行に分割しています。また、改行を追加しています。結果では、読みやすくするための改行も追加しています。

9.6.5 例：CSV として結果を返す

次は、同じ SPARQL SELECT クエリ（bills.sparql）で、結果を CSV 形式で返す例です。

```
curl --anyauth --user Admin:janem-3 -i -X POST --data-
binary \
@./bills.sparql -H "Content-type: application/sparql-query" \
-H "Accept: text/csv" http://janem-3:8000/v1/graphs/sparql
=>
bill,title

http://www.rdfabout.com/rdf/usgov/congress/108/bills/h1171,
H.R.108/1171: Iris Scan Security Act of 2003

http://www.rdfabout.com/rdf/usgov/congress/108/bills/h1314,
H.R.108/1314: Screening Mammography Act of 2003
```

```
http://www.rdfabout.com/rdf/usgov/congress/108/bills/h1384,
H.R.108/1384: To amend the Railroad Retirement Act of 1974
to eliminate a limitation on benefits.

http://www.rdfabout.com/rdf/usgov/congress/108/bills/h1418,
H.R.108/1418: Veterans' Housing Equity Act

http://www.rdfabout.com/rdf/usgov/congress/108/bills/h1419,
H.R.108/1419: Seniors' Family Business Enhancement
Act[jmckean@janem-3 ~]$
```

注： 上記の例では、長い行は UNIX の行継続文字「\」で複数の行に分割しています。また、改行を追加しています。結果では、読みやすくするための改行も追加しています。

9.6.6 例：N-triples として結果を返す

この例としては、すでに導入および使用されている次の DESCRIBE クエリを使用します。

```
DESCRIBE <http://dbpedia.org/resource/Pascal_Bedrossian>
```

次のコマンドは、このクエリを使用して、Pascal について説明しているトリプルとマッチさせ、N-Triples として結果を返します。下記のコマンドにある長い行は、UNIX の行継続文字「\」で分割されています。クエリは、URL エンコードされており、「query」リクエストパラメータの値として渡されます。

```
curl -X GET --digest --user "admin:password" \
-H "Accept: application/n-triples" \
-H "Content-type: application/x-www-form-urlencoded"\
"http://localhost:8321/v1/graphs/sparql?query=DESCRIBE%20%
3Chttp\%3A%2F%2Fdbpedia.org%2Fresource%2FPascal_Bedrossian
%3E"
=>
<http://dbpedia.org/resource/Pascal_Bedrossian>
<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/France> .

<http://dbpedia.org/resource/Pascal_Bedrossian>
<http://dbpedia.org/ontology/birthPlace>
<http://dbpedia.org/resource/Marseille> .

<http://dbpedia.org/resource/Pascal_Bedrossian>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://xmlns.com/foaf/0.1/Person> .
```

```
<http://dbpedia.org/resource/Pascal_Bedrossian>
<http://xmlns.com/foaf/0.1/surname> "Bedrossian"@en .

<http://dbpedia.org/resource/Pascal_Bedrossian>
<http://xmlns.com/foaf/0.1/givenName> "Pascal"@en .

<http://dbpedia.org/resource/Pascal_Bedrossian>
<http://xmlns.com/foaf/0.1/name> "Pascal Bedrossian"@en .

<http://dbpedia.org/resource/Pascal_Bedrossian>
<http://purl.org/dc/elements/1.1/description>
"footballer"@en .

<http://dbpedia.org/resource/Pascal_Bedrossian>
<http://dbpedia.org/ontology/birthDate> "1974-11-
28"^^<http://www.w3.org/2001/XMLSchema#date> .
```

9.6.7 例：XML または JSON としてブール型の値を返す

次の例では、前述の例から SPARQL の `ASK` クエリを使用して、Carolyn Kennedy が Eunice Kennedy よりも後に出生したかどうかを判断します。

以下に、次のクエリで使用されている `ask-sparql.sparql` ファイルのコンテンツを示します。

```
#file: ask-sparql.sparql
PREFIX db: <http://dbpedia.org/resource/>
PREFIX onto: <http://dbpedia.org/ontology/>
ASK
{
    db:Carolyn_Bessette-Kennedy onto:birthDate ?by .
    db:Eunice_Kennedy_Shriver onto:birthDate ?bd .
    FILTER (?by>?bd).
}
```

注： `curl` を使用して `PUT` または `POST` リクエストを実行し、ファイルからリクエストのボディを読み取る場合は、`-d` ではなく `--data-binary` を使用して入力ファイルを指定してください。`--data-binary` を使用すると、`curl` によってファイルからリクエストのボディにデータがそのまま挿入されます。`-d` を使用すると、`curl` によって入力から改行が取り除かれるため、トリプルデータや SPARQL がシンタックスの面で無効になる可能性があります。

SPARQL の ASK クエリを含んでいるこのリクエストは、ブール型の結果を XML 形式で返します。

```
curl --anyauth --user user:password -i -X POST \
--data-binary @./ask-sparql.sparql \
-H "Content-type: application/sparql-query" \
-H "Accept: application/sparql-results+xml" \
http://localhost:8050/v1/graphs/sparql

=>

HTTP/1.1 200 OK
Content-type: application/sparql-results+xml
Server: MarkLogic
Content-Length: 89
Connection: Keep-Alive
Keep-Alive: timeout=5

<sparql
<xmlns="http://www.w3.org/2005/sparql-results/">
<boolean>true</boolean>
</sparql>
```

次は、同じリクエスト（SPARQL の ASK クエリを含んでいます）で、ブール型の結果が JSON 形式で返されます。

```
curl --anyauth --user user:password -i -X POST \
--data-binary @./ask-sparql.sparql \
-H "Content-type: application/sparql-query" \
-H "Accept: application/sparql-results+json" \
http://localhost:8050/v1/graphs/sparql

=>

HTTP/1.1 200 OK
Content-type: application/sparql-results+json
Server: MarkLogic
Content-Length: 17
Connection: Keep-Alive
Keep-Alive: timeout=5

{"boolean":true}
```

9.7 REST クライアント API を使用した SPARQL クエリ

SPARQL クエリ（SELECT、DESCRIBE、CONSTRUCT、および ASK）は、POST か GET のいずれかおよび REST クライアント API との併用が可能です。クエリタイプおよび出力の詳細については、「SPARQL クエリタイプと出力形式」（207 ページ）の表を参照してください。

このセクションでは、次の内容を取り上げます。

- POST リクエストでの SPARQL クエリ
- GET リクエストでの SPARQL クエリ

9.7.1 POST リクエストでの SPARQL クエリ

このセクションでは、SPARQL クエリを使用して、/v1/graphs/sparql エンドポイントを通じてグラフおよびトリプルデータを管理する方法について説明します。

```
http://hostname:port/v1/graphs/sparql
```

「hostname」および「port」は、MarkLogic を実行しているホストとポートです。

POST:/v1/graphs/sparql への入力 SPARQL クエリの指定は次の方法で行うことができます。

- POST のボディで SPARQL クエリをファイルとして含める。
- URL にエンコードされたデータとして SPARQL クエリを含める。

次の例は、米国議会法案 44 について調べるシンプルな SPARQL の DESCRIBE クエリを示しています。このクエリは、bill44.sparql という名前のファイルとして保存されています。

```
#file name bill44.sparql
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX bill: <http://www.rdfabout.com/rdf/schema/usbill/>
DESCRIBE ?x WHERE { ?x rdf:type bill:HouseBill ;
                       bill:number "44" .}
```

注： curl を使用して PUT または POST リクエストを実行し、ファイルからリクエストのボディを読み取る場合は、-d ではなく --data-binary を使用して入力ファイルを指定してください。--data-binary を使用すると、curl によってファイルからリクエストのボディにデータがそのまま挿入されます。-d を使用すると、curl によって入力から改行が取り除かれるため、トリプルデータや SPARQL がシンタックスの面で無効になる可能性があります。

エンドポイントでは、SPARQL クエリをパラメータにするか、POST のボディに入れることを必要とします。次の例では、DESCRIBE クエリが記述されている bill44.sparql ファイルが POST リクエストのボディに渡されます。

```
# Windows users, see Modifying the Example Commands for Windows
curl --anyauth --user admin:password \
-i -X POST --data-binary @./bill44.sparql \
-H "Content-type: application/sparql-query" \
-H "Accept: application/rdf+xml" \
http://localhost:8000/v1/graphs/sparql
```

リクエストのボディの MIME タイプは application/sparql-query と指定され、リクエストされる応答 MIME タイプ（Accept:）は application/rdf+xml と指定されています。出力は XML 形式のトリプルとして返されます。詳細については、「応答の出力形式」（207 ページ）を参照してください。

このクエリは、法案 44 について説明する次のトリプルを返します。

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
ns">
 <OfficialTitle rdf:ID="bnodebnode3097714188l9f878"
    xmlns="http://www.rdfabout.com/rdf/schema/usbill/">
  <rdf:typerdf:resource="http://www.rdfabout.com/rdf/schema/
  usbill
    /OfficialTitle"/>
  <rdf:value rdf:datatype="http://www.w3.org/2001/
  XMLSchema#string">
    To amend the Internal Revenue Code of 1986 to provide
    reduced capital gain rates for qualified economic
    stimulus gain and to index the basis of assets of
    individuals for purposes of determining gains and
    losses.</rdf:value></OfficialTitle>
 <ShortTitle rdf:ID="bnodebnode30b47143b819db78"
   xmlns="http://www.rdfabout.com/rdf/schema/usbill/">
  <rdf:type rdf:resource="http://www.rdfabout.com/rdf/
  schema/usbill/ShortTitle"/>
  <rdf:value rdf:datatype="http://www.w3.org/2001/
  XMLSchema#string">Investment Tax Incentive Act of 2003
  </rdf:value></ShortTitle>
 <ShortTitle rdf:ID="bnodebnodee1860b72fb58b315"
   xmlns="http://www.rdfabout.com/rdf/schema/usbill/">
  <rdf:type rdf:resource="http://www.rdfabout.com/rdf/
  schema/usbill
   /ShortTitle"/>
```

```
  <rdf:value rdf:datatype="http://www.w3.org/2001/
  XMLSchema#string">
   Investment Tax Incentive Act of 2003</rdf:value>
   </ShortTitle>
 <HouseBill rdf:about="http://www.rdfabout.com/rdf/usgov/
 congress/108/bills/h44"
xmlns="http://www.rdfabout.com/rdf/schema/usbill/">
 <inCommittee rdf:resource="http://www.rdfabout.com
  /rdf/usgov/congress/committees/HouseWaysandMeans"/>
 <cosponsor rdf:resource="http://www.rdfabout.com/rdf/
  usgov/congress/people/A000358"/>
 <cosponsor rdf:resource="http://www.rdfabout.com/rdf/
  usgov/congress/people/B000208"/>
 <cosponsor rdf:resource="http://www.rdfabout.com/rdf/
  usgov/congress
   /people/B000575"/>
 <cosponsor
.....
```

POST リクエストの使用方法は、「query」パラメータの値として、URL エンコードされたクエリを指定し、リクエストのボディの MIME タイプとして application/x-www-form-urlencoded を使用するものもあります（『*REST Client API*』のセマンティックに関するドキュメントで説明しています）。

次の SPARQL の SELECT クエリは、BioGuide ID が「A000358」である人物が共同提出者であった議院法案を調べます。

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX bill: <http://www.rdfabout.com/rdf/schema/usbill/>
PREFIX people: <http://www.rdfabout.com/rdf/usgov/
congress/people/>
SELECT ?x WHERE { ?x rdf:type bill:HouseBill ;
bill:cosponsor  people:A000358.}
```

この例では、SELECT クエリが URL エンコードされています。これが、フォームエンコードされたデータとして送信されます。

```
curl -X POST --anyauth --user admin:password \
-H "Accept:application/sparql-results+xml" --data-binary \
"query=PREFIX%20rdf%3A%20%3Chttp%3A%2F%2Fwww.w3.org%2F1999%2
F02%2F22-rdf-syntax-ns\
%23%3E%20%0APREFIX%20bill%3A%20%3Chttp%3A%2F%2Fwww.rdfabout.
com%2Frdf\
%2Fschema%2Fusbill%2F%3E%0APREFIX%20people%3A%20%3Chttp%3A%2
F%2Fwww.rdfabout.com\
```

```
%2Frdf%2Fusgov%2Fcongress%2Fpeople%2F%3E%0ASELECT%20%3Fx%20W
HERE%20%7B%20%3Fx\
%20rdf%3Atype%20bill%3AHouseBill%20%3B%20bill%3Acosponsor%20
%20people%3AA000358.%20%7D%0A"\
-H "Content-type:application/x-www-form-urlencoded" \
http://localhost:8000/v1/graphs/sparql
=>

<sparql xmlns="http://www.w3.org/2005/sparql-results/">
   <head><variable name="x"/></head>
  <results>
    <result><binding name="x">
     <uri>http://www.rdfabout.com/rdf/usgov/congress/108/
     bills/h1036
     </uri></binding></result>
    <result><binding name="x">
     <uri>http://www.rdfabout.com/rdf/usgov/congress/108/
     bills/h1057
     </uri></binding></result>
    <result><binding name="x">
     <uri>http://www.rdfabout.com/rdf/usgov/congress/108/
     bills/h1078
     </uri></binding></result>
    <result><binding name="x">
     <uri>http://www.rdfabout.com/rdf/usgov/congress/108/
     bills/h110
     </uri></binding></result>
    <result><binding name="x">
     <uri>http://www.rdfabout.com/rdf/usgov/congress/108/
     bills/h1117
     </uri></binding></result>
    <result><binding name="x">
     <uri>http://www.rdfabout.com/rdf/usgov/congress/108/
     bills/h1153
     </uri></binding></result>
.......
    <result><binding name="x">
     <uri>http://www.rdfabout.com/rdf/usgov/congress/108/
     bills/h975
     </uri></binding></result>
  </results>
</sparql>
```

注： 読みやすくするため、この長いコマンドラインを UNIX の行継続文字「\」で複数の行に分割しています。また、URL エンコードされたクエリを読みやすくするため、追加の改行が挿入されています。

9.7.2 GET リクエストでの SPARQL クエリ

GET リクエストの場合、クエリリクエストパラメータ内の SPARQL クエリは URL エンコードされている必要があります。次に、エンコードする前の、米国議会法案（44）を検索する SPARQL の DESCRIBE クエリを示します。

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX bill: <http://www.rdfabout.com/rdf/schema/usbill/>
DESCRIBE ?x
WHERE {
  ?x rdf:type bill:HouseBill ;
    bill:number "44" .}
```

次の例では、curl によって HTTP GET リクエストが送信されて、SPARQL の DESCRIBE クエリが実行されます。

```
curl -X GET --digest --user "user:password" \
-H "accept: application/sparql-results+xml" \
"http://localhost:8000/v1/graphs/sparql?query=PREFIX%20rdf%3
A%20%3C\
http%3A%2F%2Fwww.w3.org%2F1999%2F02%2F22-rdf-syntax-
ns%23%3E%20%0A\
PREFIX%20bill%3A%20%3Chttp%3A%2F%2Fwww.rdfabout.com%2Frdf%2F
schema\
%2Fusbill%2F%3E%0ADESCRIBE%20%3Fx%20WHERE%20%7B%20%3Fx%20rdf
%3Atype\
%20bill%3AHouseBill%20%3B%20%20bill%3Anumber%20%2244%22%20.%
20%7D"
```

結果は次のようなトリプルになります。

```
<http://www.rdfabout.com/rdf/usgov/congress/108/bills/h44>
<http://purl.org/dc/elements/1.1/title> "H.R.108/44:
Investment Tax
Incentive Act of 2003" .

<http://www.rdfabout.com/rdf/usgov/congress/108/bills/h44>
<http://purl.org/dc/terms/created> "2003-01-07" .

<http://www.rdfabout.com/rdf/usgov/congress/108/bills/h44>
<http://purl.org/ontology/bibo/shortTitle> "H.R.44：
Investment Tax
Incentive Act of 2003" .

<http://www.rdfabout.com/rdf/usgov/congress/108/bills/h44>
<http://www.rdfabout.com/rdf/schema/usbill/congress> "108" .
```

```
<http://www.rdfabout.com/rdf/usgov/congress/108/bills/h44>
<http://www.rdfabout.com/rdf/schema/usbill/cosponsor>
<http://www.rdfabout.com/rdf/usgov/congress/people/A000358> .

<http://www.rdfabout.com/rdf/usgov/congress/108/bills/h44>
<http://www.rdfabout.com/rdf/schema/usbill/cosponsor>
<http://www.rdfabout.com/rdf/usgov/congress/people/B000208> .

<http://www.rdfabout.com/rdf/usgov/congress/108/bills/h44>
<http://www.rdfabout.com/rdf/schema/usbill/cosponsor>
<http://www.rdfabout.com/rdf/usgov/congress/people/B000575> .
```

トリプルには、タイトル、作成日、共同提出者など米国議会の法案 44 に関する情報が記述されています。

9.8 REST クライアント API を使用した SPARQL Update

このセクションでは、SPARQL Update を使用して、`/v1/graphs/sparql` エンドポイントを通じてグラフおよびトリプルデータを管理する方法について説明します。

```
http://hostname:port/v1/graphs/sparql
```

「hostname」および「port」は、MarkLogic を実行しているホストとポートです。

`POST:/v1/graphs/sparql` への SPARQL Update（`DELETE/INSERT`) の指定は次の方法で行うことができます。

- `POST` のボディで SPARQL Update をファイルとして含める。

  ```
  http://host:port/v1/graphs/sparql
  content-type:application/sparql-update
  ```

  「POST リクエストでの SPARQL Update」（223 ページ）を参照してください。

- 次の形式で、URL エンコードされたデータとして SPARQL Update を含める。

  ```
  http://host:port/v1/graphs/sparql
  content-type:application/x-www-form-urlencoded
  ```

  「POST と URL エンコードされたパラメータを使用した SPARQL Update」（225 ページ）を参照してください。

このセクションの例では、URL エンコーディングなしで `POST` リクエストを使用し、`content-type:application/sparql-update` を指定しています。

using-graph-uri と using-named-graph-uri リクエストパラメータを使用して、更新を実行する対象となる RDF データセットを指定するか、または更新内に、RDF データセットを指定できます。データセットをリクエストパラメータと更新のいずれにも指定した場合、リクエストパラメータによって定義されたデータセットが使用されます。いずれにも指定しなかった場合、操作にすべてのグラフ（すべてのグラフの UNION（和集合））が含まれます。

> **注：** USING 節、USING NAMED 節、WITH 節を使用する操作を含んでいる SPARQL 1.1 Update リクエストとともに using-graph-uri パラメータまたは using-named-graph-uri パラメータを含めると、エラーが発生します。

REST クライアント API で使用するパラメータを指定する方法の詳細については、「パラメータの指定」（197 ページ）を参照してください。POST の詳細については、「REST クライアント API に対してサポートされている操作」（201 ページ）および SPARQL Update のグラフストアエンドポイントでサポートされている動詞のリストを参照してください。SPARQL エンドポイントの詳細については、POST:/v1/graphs/sparql を参照してください。

このセクションでは、次の内容を取り上げます。

- POST リクエストでの SPARQL Update
- POST と URL エンコードされたパラメータを使用した SPARQL Update

9.8.1 POST リクエストでの SPARQL Update

リクエストのボディに SPARQL Update を含めることで、POST メソッドを使用してリクエストを送信できます。Content-type HTTP ヘッダは application/sparql-update に設定してください。

次に、リクエストのボディに追加する前の SPARQL Update を示します。

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
WITH <BOOKS>
DELETE {?b dc:title "A new book"}
INSERT
{?b dc:title "Inside MarkLogic Server" }
WHERE {?b dc:title "A new book".}
```

<BOOKS> という名前のグラフでは、SPARQL Update は、述語の位置に dc:title があり、目的語の位置に A new book があるトリプルとマッチし、それを削除します。その後、新しいトリプルが挿入されます（?b dc:title "MarkLogic Server"）。

次の例では、リクエストのボディでクエリに application/sparql-update と -d オプションを使用して、SPARQL Update が送信されます。

```
# Windows users, see Modifying the Example Commands for Windows

curl --anyauth --user admin:admin -i -X POST \
-H "Content-type:application/sparql-update" \
-H "Accept:application/sparql-results+xml" \
-d 'PREFIX dc: <http://purl.org/dc/elements/1.1/> \
WITH <BOOKS> \
DELETE {?b dc:title "A new book"} \
INSERT {?b dc:title "Inside MarkLogic Server" } \
WHERE {?b dc:title "A new book".}'\
http://localhost:8000/v1/graphs/sparql
```

注： わかりやすくするため、長いコマンドラインは行継続文字「\」を使用して複数の行に分割しています。curl コマンドを使用するときは、行継続文字を削除してください（Windows の場合、行継続文字は「^」になります）。

代わりに、curl を使用して、ファイルから POST リクエストの一部として SPARQL Update を実行することもできます。SPARQL Update は、booktitle.sparql という名前のファイルに保存されます。次に、ファイルの内容を示します。

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
INSERT DATA
{
<http://example/book1> dc:title "book title" ;
dc:creator "author name" .
}
```

ファイルに SPARQL Update を含む POST リクエストは、次のようになります。

```
curl --anyauth --user admin:admin -i -X POST \
--data-binary @./booktitle.sparql \
-H "Content-type:application/sparql-update" \
-H "Accept:application/sparql-results+xml" \
http://localhost:8000/v1/graphs/sparql
```

リクエストは、-d ではなく --data-binary オプションを使用して SPARQL Update を含んでいるファイルを呼び出していることに注意してください。using-graph-uri、using-named-graph-uri および role-capability は、HTTP リクエストのパラメータとして含めることができます。このシンタックスでは、perm パラメータがロールと機能とともに要求されます。

```
perm:admin=update&perm:admin=execute
```

パーミッションの詳細については、「デフォルトパーミッションと必須の権限」（230 ページ）を参照してください。

9.8.2 POST と URL エンコードされたパラメータを使用した SPARQL Update

パラメータを URL エンコードして、HTTP の `POST` メソッドを使用した更新プロトコルリクエストを送信することもできます。これを実行する場合は、URL のすべてのパラメータをパーセント記号でエンコードし、`application/x-www-form-urlencoded` メディアタイプを使用してリクエストボディ内にパラメータとして含めます。HTTP リクエストのコンテンツタイプヘッダは `application/x-www-form-urlencoded` に設定します。

次の例では、URL エンコードされたパラメータを含む SPARQL Update および `POST` を使用して、データ（およびパーミッションの集まり）をグラフ `C1` に挿入しています。

```
curl --anyauth --user admin:admin -i -X POST \
--data-urlencode update='PREFIX dc:
<http://purl.org/dc/elements/1.1/> \
INSERT DATA \
{<http://example/book1> dc:title "book title" ; \
dc:creator "author name" .}'\
-H "Content-type:application/x-www-form-urlencoded" \
-H "Accept:application/sparql-results+xml" \
'http://localhost:8000/v1/graphs/sparql?using-named-graph-
uri=C1 \
&perm:admin=update&perm:admin+execute'
```

このプロトコルを使用して `USING NAMED` または `WITH` 節を使用する SPARQL 1.1 Update リクエストを伝達する場合、`using-graph-uri` または `using-named-graph-uri` パラメータを指定すると、この操作はエラーになります。

次の `curl` の例では、SPARQL Update とパーミッションの URL エンコードを含む `POST` を使用しています。

```
curl --anyauth --user admin:admin -i -X POST \
-H "Content-type:application/x-www-form-urlencoded" \
-H "Accept:application/sparql-results+xml" \
--data-urlencode update='PREFIX dc:
<http://purl.org/dc/elements/1.1/>
INSERT DATA{ GRAPH <C1> {http://example/book1/> dc:title "C
book"} }' \
```

```
--data-urlencode perm:rest-writer=read \
--data-urlencode perm:rest-writer=update \
http://localhost:8321/v1/graphs/sparql
```

新しい RDF グラフが作成されると、サーバーは `201 Created` メッセージで応答します。更新リクエストに対する応答では、HTTP 応答ステータスコード（`200` または `400`）でリクエストの成功または失敗を示します。リクエストのボディが空の場合、サーバーは `204 No Content` で応答します。

9.9 REST クライアント API でグラフ名をリストする

`things` エンドポイントを使用する REST クライアント API で、データベース内のグラフをリストできます。

```
http://hostname:port/v1/things
```

「hostname」および「port」は、MarkLogic を実行しているホストとポートです。

例えば、次のようにパラメータを指定しないでこのエンドポイントを呼び出すと、データベース内のグラフのリストが返されます。

```
http://localhost:8000/v1/things
```

このリクエストにより、次のようなグラフが返されることがあります。

```
graphs/MyDemoGraph
http://marklogic.com/semantics#default-graph
http://marklogic.com/semantics#graphs
```

9.10 REST クライアント API でトリプルを探索する

次のエンドポイントは、データベース内で参照される知識（物）への RESTful なアクセスを提供します。このエンドポイントは、データベース内の主語ノードすべてのリストを取得します。

```
http://hostname:port/v1/graphs/things
```

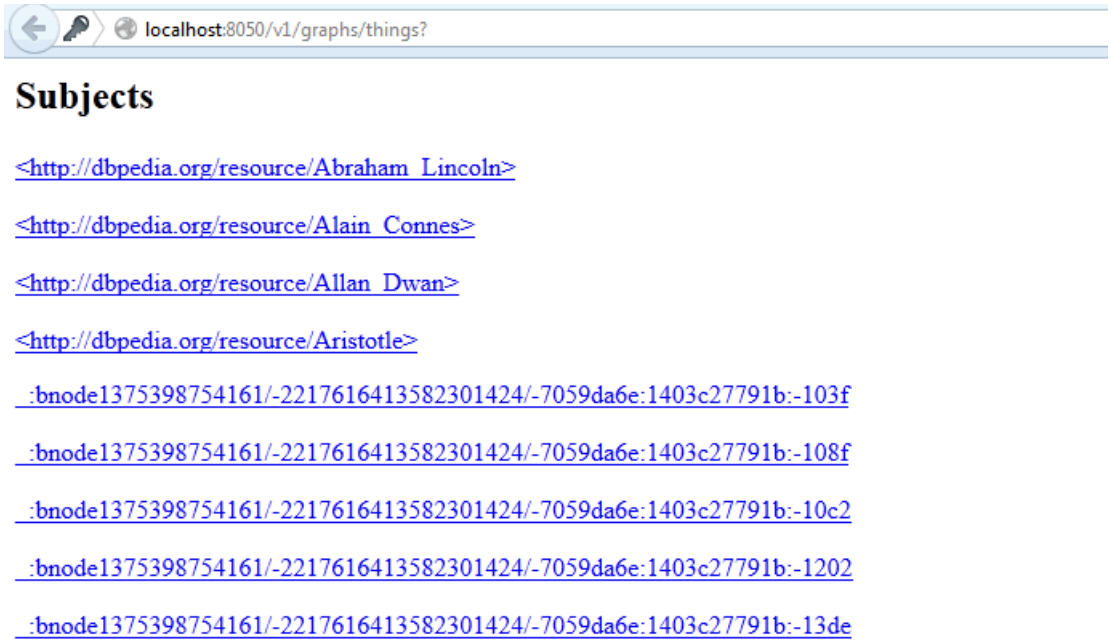
「hostname」および「port」は、MarkLogic を実行しているホストとポートです。

例えば以下のようになります。

```
http://localhost:8050/v1/graphs/things
```

また、主語ノードの集まりを返すように指定することもできます。このエンドポイントがパラメータなしで呼び出されると、データベース内のすべてのトリプルに関してデータベース内の主語ノードのリストが返されます。

次の例は、web ブラウザでの応答、つまり IRI としてのノードのリストを示したものです。



注： このエンドポイントは、表示する項目数を 10,000 件に制約するようにハードコーディングされています。ページネーションはサポートされません。

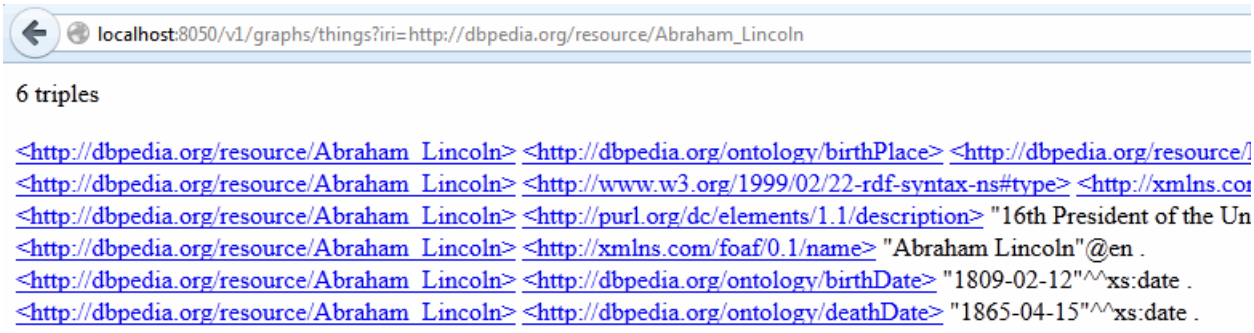
リンクをクリックしてノードをドリルダウンすることで、データベース内のトリプルをトラバースおよびナビゲートできます。IRI をクリックすると、1 つあるいは複数の関連トリプルが返されることがあります。

オプションの `iri` パラメータを使用して、返す情報に関する特定の IRI を Turtle トリプルのシリアライゼーションで指定できます。

例えば、次のリクエストをブラウザに貼り付けることができます。

```
http://localhost:8050/v1/graphs/things?iri=http://dbpedia.
org/resource/Abraham_Lincoln
```

IRI `http://dbpedia.org/resources/Abraham_Lincoln` で選択されたノードが、Turtle シリアライゼーションで返されます。



HTTP リクエストの発行に、curl、またはそれに相当するコマンドラインツールを使用している場合は、次の MIME タイプをリクエストの Accept ヘッダに指定できます。

- パラメータが指定されない場合は、リクエストの Accept ヘッダで text/html を使用します。
- iri パラメータを使用する場合は、「SPARQL クエリタイプと出力形式」（207 ページ）にリストされている MIME タイプのいずれかを使用する必要があります。RDF トリプルの形式の詳細については、「サポートされている RDF トリプルの形式」（31 ページ）を参照してください。

次の例では、GET リクエストが、指定されている iri パラメータで選択されたノードを Turtle トリプルのシリアライゼーションで返します。

```
curl --anyauth --user admin:password -i -X GET \
-H "Accept: text/turtle" \
http://localhost:8051/v1/graphs/things?iri=http://dbpedia.
org/resource/Aristotle

=>

HTTP/1.1 200 OK
Content-type: text/turtle; charset=UTF-8
Server: MarkLogic
Content-Length: 628
Connection: Keep-Alive
Keep-Alive: timeout=5

@prefix xs: <http://www.w3.org/2001/XMLSchema#> .
<http://dbpedia.org/resource/Aristotle>
<http://dbpedia.org/ontology/deathPlace>
 <http://dbpedia.org/resource/Chalcis> .
<http://dbpedia.org/resource/Aristotle>
<http://dbpedia.org/ontology/birthPlace>
 <http://dbpedia.org/resource/Stagira_(ancient_city)> .
<http://dbpedia.org/resource/Aristotle>
<http://www.w3.org/1999/02/22-rdf-syntax
-ns#type/> <http://xmlns.com/foaf/0.1/Person> .
<http://dbpedia.org/resource/Aristotle>
<http://xmlns.com/foaf/0.1/name>
"Aristotle" .
<http://dbpedia.org/resource/Aristotle>
<http://purl.org/dc/elements/1.1/description>
"Greek philosopher" .
```

注： 指定されている IRI が存在しない場合の応答は、「404 Not Found」です。サポートされていないシリアライゼーションの応答に対して `GET` リクエストを実行すると、「406 Not Acceptable」が発生します。

9.11 グラフパーミッションの管理

このセクションでは、グラフパーミッションを設定、修正、および取得するための REST クライアント API サポートについて取り上げます。MarkLogic セキュリティモデルについてまだよくわからない場合は、『*Security Guide*』を確認してください。

ここでは、次の内容を取り上げます。

- デフォルトパーミッションと必須の権限
- 他の操作の一部としてパーミッションを設定する
- パーミッションの単独設定
- グラフパーミッションの取得

9.11.1 デフォルトパーミッションと必須の権限

REST クライアント API を使用して作成および管理されるグラフはすべて、「読み取り」機能を `rest-reader` ロールに、「更新」機能を `rest-writer` ロールにそれぞれ付与します。これらのデフォルトパーミッションは、明示的に指定しなくても、常にグラフに割り当てられます。

例えば、`PUT /v1/graphs` を使用して新しいグラフを作成し、何もパーミッションを指定しなかった場合、グラフには次のようなパーミッションが設定されます。

XML	JSON
`<metadata xmlns="http://marklogic.com/rest-api">` `  <permissions>` `  <permission>` `  <role-name>rest-writer` `  </role-name>` `  <capability>update</capability>` `  </permission>` `  <permission>` `  <role-name>rest-reader` `  </role-name>` `  <capability>read</capability>` `  </permission>` `  </permissions>` `</metadata>`	`{"permissions":[` `  {"role-name":` `  "rest-writer",` `  "capabilities":` `  ["update"]` `  },` `  {"role-name":` `  "rest-reader",` `  "capabilities":` `  ["read"]` `  }` `]}`

グラフの作成時に、他のパーミッションを明示的に指定した場合も、上記のデフォルトパーミッションが設定され、さらに指定したパーミッションも設定されます。

カスタムロールを使用すると、選択したユーザーにアクセスをグラフ単位で制限できます。カスタムロールには、同等の `rest-reader` 権限と `rest-writer` 権限を含める必要があります。これを行わないと、このロールを持つユーザーは REST クライアント API を使用してセマンティックデータを管理およびクエリできません。

詳細については、『*REST Application Developer's Guide*』の「Security Requirements」を参照してください。

9.11.2 他の操作の一部としてパーミッションを設定する

他のグラフ操作の一部としてパーミッションの設定、上書き、または追加を行うには、`perm` リクエストパラメータを使用します。グラフのコンテンツを修正しない場合にパーミッションを管理するには、代わりに、`category` パラメータを使用してください。詳細については、「パーミッションの単独設定」（232 ページ）を参照してください。

`perm` パラメータは次の形式を取ります。

```
perm:role=capability
```

ここで、*role* には、MarkLogic で定義されているロール名を入れます。*capability* には、「read」、「insert」、「update」、「execute」のいずれかを入れます。

`perm` パラメータを複数回使用すると、同じロールに複数の機能を付与したり、複数のロールに対してパーミッションを設定したりできます。例えば、次のパラメータの集まりは、「readers」ロールに「read」機能、「writers」ロールに「update」機能を付与しています。

```
perm:readers=read&perm:writers=update
```

注： グラフのパーミッションを設定または変更しても、そのグラフ内に組み込まれたトリプルを含んでいるドキュメントのパーミッションには影響しません。

perm パラメータは次の操作で使用できます。

操作	REST クライアント API のメソッド
グラフのコンテンツの作成または上書き中に、名前付きグラフまたはデフォルトグラフのパーミッションを設定または上書きします。	`PUT /v1/graphs?graph=uri&perm:` *`role=capability`* `PUT /v1/graphs?default&perm:` *`role=capability`* リクエストのボディには、グラフの新しいコンテンツ（トリプル）が格納されます。
SPARQL Update の操作中に、更新の一部として、作成されたグラフすべてのパーミッションを設定します。	`POST /v1/graphs/sparql?perm:` *`role=capability`* リクエストのボディには、SPARQL Update が格納されます。
グラフへのコンテンツの追加中に、名前付きグラフにパーミッションを追加します。	`POST /v1/graphs?graph=uri&perm:` *`role=capability`* `POST /v1/graphs?default&perm:` *`role=capability`* リクエストのボディには、グラフの更新（トリプル）が格納されます。

次の制限が適用されます。

- perm パラメータを /v1/graphs で使用する場合は、graph または default リクエストパラメータのいずれかも含める必要があります。
- perm パラメータは、category=permissions や category=metadata とは併用できません。
- SPARQL Update 操作の一部として perm パラメータを使用してパーミッションを指定すると、そのパーミッションは更新の一部として作成されたグラフにのみ影響します。すでに存在するグラフのパーミッションは変更されません。

9.11.3 パーミッションの単独設定

グラフのコンテンツに影響を与えることなくパーミッションを管理するには、category リクエストパラメータを permissions に設定します。例えば、リクエストのボディにパーミッションのメタデータを含む次の形式のリクエストは、デフォルトグラフのパーミッションを設定しますが、グラフのコンテンツは変更しません。

```
PUT /v1/graphs?default&category=permissions
```

パーミッションをグラフのコンテンツとともに設定および追加するには、perm リクエストパラメータを使用します。詳細については、「他の操作の一部としてパーミッションを設定する」（231 ページ）を参照してください。

category パラメータは、permissions または metadata のいずれかに設定できます。これらは、グラフ管理のコンテキストでは等価です。

リクエストのボディには、パーミッションのメタデータを格納する必要があります。XML では、metadata 要素または permissions 要素をルートにすることができます。また、XML では、メタデータを名前空間 http://marklogic.com/rest-api に入れる必要があります。

例えば、次のすべてが利用できます。

XML	JSON
<metadata xmlns="http://marklogic.com/rest-api"> <permissions> <permission> <role-name>roleA</role-name> <capability>read</capability> <capability>update</capability> </permission> <permission> <role-name>roleB</role-name> <capability>read</capability> </permission> </permissions> </metadata>	{"permissions":[{"role-name":"roleA", "capabilities":["read", "update"] }, {"role-name":"roleB", "capabilities": "["read"] }]}
<permissions xmlns="http://marklogic.com/ rest-api"> <permission> <role-name>roleA</role-name> <capability>read</capability> <capability>update</capability> </permission> <permission> <role-name>roleB</role-name> <capability>read</capability> </permission> </permissions>	

注： グラフのパーミッションを設定または変更しても、そのグラフ内に組み込まれたトリプルを含んでいるドキュメントのパーミッションには影響しません。

次の手法でグラフのパーミッションを管理するには、`category=permissions` パターンを使用できます。いずれの場合も、グラフのコンテンツには影響しません。

操作	REST クライアント API のパターン
名前付きグラフまたはデフォルトグラフのパーミッションを設定または上書きする	`PUT /v1/graphs?graph=uri&category=permissions` `PUT /v1/graphs?default&category=permissions` リクエストのボディには、パーミッションのメタデータが格納されます。また、`category=metadata` も使用できます。
名前付きグラフまたはデフォルトグラフにパーミッションを追加する	`POST /v1/graphs?graph=uri&category=permissions` `POST /v1/graphs?default&category=permissions` リクエストのボディには、パーミッションのメタデータが格納されます。また、`category=metadata` も使用できます。
名前付きグラフまたはデフォルトグラフのパーミッションをデフォルトのパーミッションにリセットする	`DELETE /v1/graphs?graph=uri&category=permissions` `DELETE /v1/graphs?default&category=permissions`

次の制限が適用されます。

- `category=permissions` または `category=metadata` を `/v1/graphs` で使用する場合は、`graph` または `default` リクエストパラメータのいずれかも含める必要があります。
- `category=permissions` や `category=metadata` は、`perm` パラメータとは併用できません。

9.11.4 グラフパーミッションの取得

名前付きグラフに関する、パーミッションのメタデータを取得するには、次の形式の GET リクエストを行います。

```
GET /v1/graphs?graph=graphURI&category=permissions
```

デフォルトグラフに関する、パーミッションのメタデータを取得するには、次の形式の GET リクエストを行います。

```
GET /v1/graphs?default&category=permissions
```

メタデータは XML か JSON のいずれかでリクエストできます。デフォルトの形式は、XML です。

例えば、次のコマンドは、URI が `/my/graph` のグラフのパーミッションを XML として取得します。この場合、グラフには、デフォルトの `rest-writer` パーミッションと `read-reader` パーミッション、および「GroupA」という名前のカスタムロールのパーミッションが含まれます。

```
curl --anyauth --user user:password -X GET -i \
  -H "Accept: application/xml" \
'http://localhost:8000/v1/graphs?graph=/my/graph&category=
permissions'

HTTP/1.1 200 OK
Content-type: application/xml; charset=utf-8
Server: MarkLogic
Content-Length: 868
Connection: Keep-Alive
Keep-Alive: timeout=5

<rapi:metadata uri="/my/graph" ...
     xmlns:rapi="http://marklogic.com/rest-api" ...>
  <rapi:permissions>
    <rapi:permission>
      <rapi:role-name>rest-writer</rapi:role-name>
      <rapi:capability>update</rapi:capability>
    </rapi:permission>
    <rapi:permission>
      <rapi:role-name>rest-reader</rapi:role-name>
      <rapi:capability>read</rapi:capability>
    </rapi:permission>
    <rapi:permission>
      <rapi:role-name>GroupA</rapi:role-name>
      <rapi:capability>read</rapi:capability>
      <rapi:capability>update</rapi:capability>
    </rapi:permission>
  </rapi:permissions>
</rapi:metadata>
```

次のデータは、これに相当するパーミッションのメタデータを JSON で表したものです。

```
{"permissions":[
  {"role-name":"rest-writer","capabilities":["update"]},
  {"role-name":"rest-reader","capabilities":["read"]},
  {"role-name":"GroupA","capabilities":["read","update"]}
]}
```

10.0 XQuery および JavaScript のセマンティック API

この章では、XQuery および JavaScript のセマンティック API について説明します。セマンティック API には、XQuery ライブラリモジュールおよびビルトインのセマンティック関数が含まれており、SPARQL、SPARQL Update、および RDF をサポートしています。セマンティック API は、大規模な実稼動トリプルストアおよびアプリケーション向けに設計されています。セマンティック関数の詳細なリストについては、http://docs.marklogic.com/sem/semantics を参照してください。

この章では、セマンティック API の使用例について説明します。セマンティック API は、MarkLogic でトリプルおよびグラフを作成、クエリ、更新、および削除できるように設計されています。

また、XQuery API、REST API、Node.js クライアント API、および Java クライアント API では、各種のクエリスタイルを使用した、MarkLogic のセマンティック機能をサポートしています（このガイドのセマンティックトリプルの読み込み、セマンティッククエリ、および XQuery およびサーバーサイド JavaScript でのトリプルの挿入、削除、および修正を参照）。

この章は、次のセクションで構成されています。

- セマンティック用の XQuery ライブラリモジュール
- トリプルの生成
- コンテンツからのトリプルの抽出
- トリプルのパース
- データを調べる

注： セマンティックは独立した製品で、別途ライセンスが必要です。セマンティックを使用するには、有効なセマンティックライセンスキーが必要です。

10.1 セマンティック用の XQuery ライブラリモジュール

セマンティックの XQuery 関数には、import ステートメントを必要としないビルトイン関数もありますが、XQuery ライブラリモジュールに実装されていて、import ステートメントを必要とするものもあります。処理を単純化するため、セマンティック API を使用する XQuery モジュールや JavaScript モジュールごとに、セマンティック API ライブラリをインポートすることをお勧めします。

10.1.1 XQuery でのセマンティックライブラリモジュールのインポート

セマンティック API ライブラリモジュールは、次のプロローグステートメントを使用して XQuery にインポートすることで、XQuery で使用できるようになります。

```
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";
```

XQuery におけるすべてのセマンティック関数のプレフィックスは、`http://marklogic.com/semantics` です。セマンティック API では、サーバーで定義されているプレフィックス `sem:` または `rdf:` を使用します。関数のシグネチャと説明の詳細については、『*XQuery and XSLT Reference Guide*』の「XQuery Library Modules」に掲載されているセマンティックに関するドキュメント、および『*MarkLogic XQuery and XSLT Function Reference*』を参照してください。

10.1.2 JavaScript でのセマンティックライブラリモジュールのインポート

JavaScript では、次のステートメントを使用してセマンティック API ライブラリモジュールを JavaScript にインポートすることで、このモジュールを使用できるようになります。

```
var sem = require("/marklogic/semantics.xqy");
```

JavaScript におけるすべてのセマンティック XQuery 関数のプレフィックスは、`/marklogic.com/semantics.xqy` です。JavaScript では、セマンティック API はプレフィックス `sem` を使用します。これはサーバーで定義されています。関数のシグネチャと説明の詳細については、『*MarkLogic XQuery and XSLT Function Reference*』および『*JavaScript Reference Guide*』の「JavaScript Library Modules」に掲載されているセマンティックに関するドキュメントを参照してください。

10.2 トリプルの生成

XQuery の `sem:rdf-builder` 関数は、セマンティック API でトリプルを動的に生成するための強力なツールです（JavaScript での関数は、`sem.rdfBuilder` です）。

この関数は、CURIE および空白ノードのシンタックスからトリプルを構築します。空白ノードは先頭のアンダースコア（`_`）で示され、空白ノード識別子が割り当てられます。空白ノードでは、複数の呼び出し間で状態が維持されます。例えば、「`_:person1`」は、やはり「`_:person1`」に言及する後の呼び出しと同じノードを参照します。例えば以下のようになります。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";
```

```
 let $builder := sem:rdf-builder((), sem:iri("my-named-
   graph"))
 let $t1 := $builder("_:person1", "a", "foaf:Person")
 let $t2 := $builder("_:person2", "a", "foaf:Person")
 let $t3 := $builder("_:person1", "foaf:knows",
 "_:person2")
 return ($t1,$t2,$t3)

=>

@prefix xs: <http://www.w3.org/2001/XMLSchema#> .

<http://marklogic.com/semantics/blank/4892021155019117627>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type/>
<http://xmlns.com/foaf/0.1/Person> .

<http://marklogic.com/semantics/blank/6695700652332466909>
        <http://xmlns.com/foaf/0.1/knows>
        _:bnode4892021155019117627 ;
        <http://www.w3.org/1999/02/22-rdf-syntax-ns#type/>
        <http://xmlns.com/foaf/0.1/Person> .
```

この例には、`sem:rdf-builder` を使用して Turtle シリアライゼーションで生成された 3 つのトリプルがあります。これらのトリプルは、person1 および person2 が人物であること、およびその関係性は person1 が person2 を知っているというものであること、というファクトを表しています。

次の点に注意してください。

- 1 番目のパラメータでは、オプションのプレフィックスマッピングの集まりが利用できます。この例では、空の引数です。「空」とはデフォルトのことであるため、`$common-prefixes` が 1 番目の引数に使用されます。2 番目の引数は、`sem:rdf-builder` 出力の名前付きグラフです。
- 述語の位置では、「a」という特殊な値が `rdf:type` の完全な IRI に展開されています。
- よく使用されるプレフィックスについては、人間が認識可能な CURIE が使用されています（例えば長い IRI ではなく `foaf:knows`）。「CURIE を扱う」（137 ページ）を参照してください。
- 3 番目のトリプルで生成される空白ノードは、1 番目および 2 番目で定義された空白ノードのアイデンティティにマッチします。

10.3 コンテンツからのトリプルの抽出

`sem:rdf-builder` 関数を使用すると、既存のコンテンツや SPARQL クエリの結果からトリプルを簡単に抽出し、データベースに対するクエリや挿入に使用する RDF グラフをすばやく構築できます。

例えば、都市や国を COL（生活コスト）の順に並べてランク付けした Web ページがあるとします。この COL は、CPI（消費者物価指数）および CPI ベースのインフレ率に基づいています。インフレ率は、前年の消費者物価と比較した年間消費者物価変動率として定義されています。メキシコのモンテレーに値 100 を割り当てて基準点とすると、データベース内のその他の都市のインフレ（`Inflation`）値はモンテレーの COL と各都市の COL を比較して計算されます。例えば、`Inflation` 値が 150 の場合は、モンテレーで生活するよりも COL が 50% 高いということです。

Ranking	City	Inflation *	Ranking	City	Inflation *
1	London (United Kingdom)	270	71	Minneapolis (United States)	134
2	Stockholm (Sweden)	266	72	Turin (Italy)	134
3	Zurich (Switzerland)	251	73	Shanghai (China)	133
4	Geneva (Switzerland)	247	74	Ljubljana (Slovenia)	132
5	New York City (United States)	225	75	Orlando (United States)	129
6	Singapore (Singapore)	219	76	Saint Petersburg (Russia)	125
7	San Francisco (United States)	209	77	Austin (United States)	125
8	Paris (France)	208	78	Phoenix (United States)	123
9	Sydney (Australia)	205	79	Almaty (Kazakhstan)	121
10	Brisbane (Australia)	203	80	Salt Lake City (United States)	120
11	Copenhagen (Denmark)	202	81	Beijing (China)	120
12	Oslo (Norway)	201	82	Lisbon (Portugal)	120
13	Tokyo (Japan)	200	83	Santiago (Chile)	120
14	Hong Kong (Hong Kong)	200	84	Montevideo (Uruguay)	119
15	Perth (Australia)	198	85	Belo Horizonte (Brazil)	116
16	Melbourne (Australia)	190	86	Brasília (Brazil)	116
17	Amsterdam (Netherlands)	189	87	Valencia (Spain)	116
18	Wellington (New Zealand)	186	88	Kansas City (United States)	116
19	Washington D.C. (United States)	185	89	Raleigh, North Carolina (United States)	115
20	Helsinki (Finland)	182	90	Istanbul (Turkey)	114
21	Dublin (Ireland)	182	91	Bangkok (Thailand)	114
22	Boston (United States)	174	92	Bogotá (Colombia)	111
23	Frankfurt am Main (Germany)	172	93	Zagreb (Croatia)	109
24	Toronto (Canada)	170	94	Oporto (Portugal)	108
25	Munich (Germany)	168	95	Taipei (Taiwan)	108
26	Manchester (United Kingdom)	166	96	Amman (Jordan)	107
27	Vancouver (Canada)	165	97	Curitiba (Brazil)	107
28	Tel Aviv (Israel)	165	98	Porto Alegre (Brazil)	107
29	Malmo (Sweden)	165	99	Kuala Lumpur (Malaysia)	105
30	Calgary (Canada)	164	100	Johannesburg (South Africa)	103

注： ここに示す値は架空のものであり、公的なソースに基づくものではありません。

COL 表の基になる HTML コードは、次のようなものです。

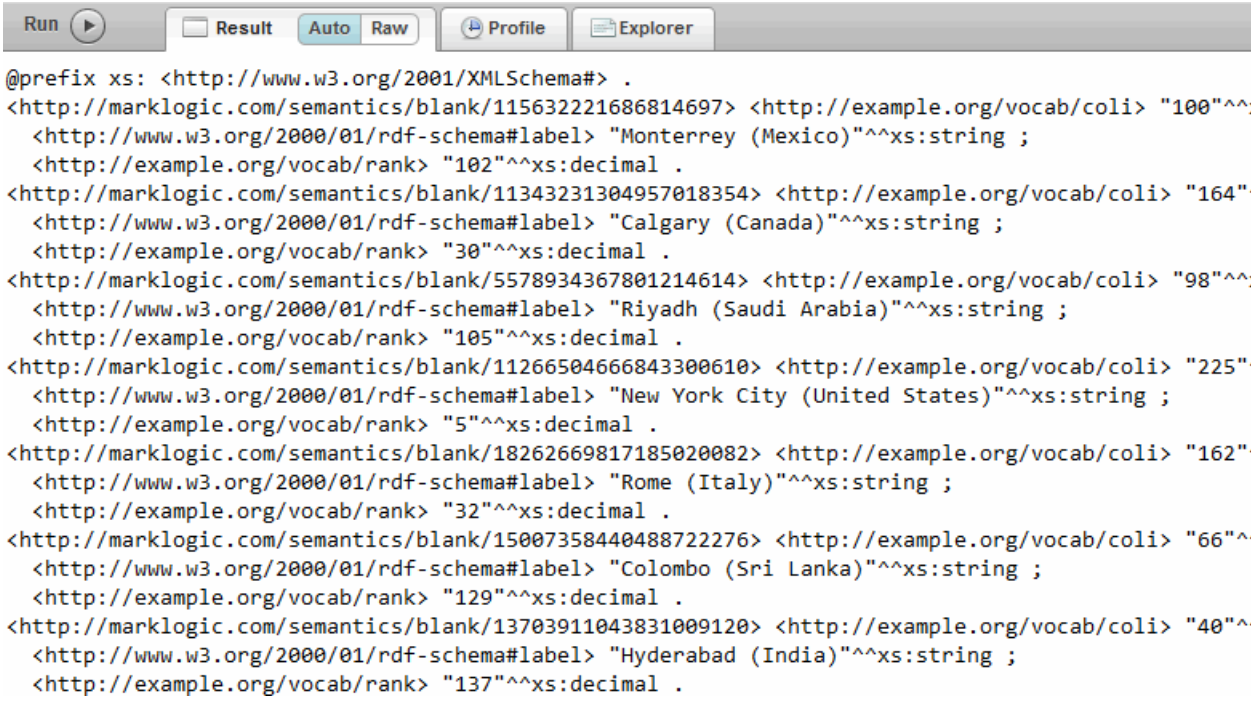
```
<table class="city-index"
style="max-width:58%;float:left;margin-right:2em;">
    <thead>
        <tr>
            <th>Ranking</th>
            <th class="city-name">City</th>
            <th class="inflation">Inflation
            <a href="#inflation-explanation">*</a></th>
        </tr>
    </thead>

           <tbody><tr>
              <td class="ranking">1</td>
              <td class="city-name">
              <a href="http://www.example.org/
              IncreasedCoL/london">
              London (United Kingdom)</a></td>
              <td class="inflation">270</td>
             </tr>

             <tr>
              <td class="ranking">2</td>
              <td class="city-name">
              <a href="http://www.example.org/
              IncreasedCoL/stockholm">
              Stockholm (Sweden)</a></td>
              <td class="inflation">266</td>
             </tr>

             <tr>
              <td class="ranking">3</td>
              <td class="city-name">
              <a href="http://www.example.org/
              IncreasedCoL/zurich">
              Zurich (Switzerland)</a></td>
              <td class="inflation">251</td>
             </tr>

             <tr>
              <td class="ranking">4</td>
              <td class="city-name">
              <ahref="http://www.example.org/
               IncreasedCoL/geneva">
              Geneva (Switzerland)</a></td>
```

```
    <td class="inflation">247</td>
  </tr>

  <tr>
    <td class="ranking">5</td>
    <td class="city-name">
    <ahref="http://www.example.org/IncreasedCoL/
    new-york">
    New York City (United States)</a></td>
    <td class="inflation">225</td>
  </tr>
```

この例では、sem:rdf-builder 関数を使用して HTML コンテンツからトリプルを抽出します。この関数は、HTML コードがすでに整形されていて、有用な分類ノード（@class）があるということを利用します。

```
import module namespace sem = "http://marklogic.com/
  semantics"
  at "/MarkLogic/semantics.xqy";
declare namespace html="http://www.w3.org/1999/xhtml";

let $doc := xdmp:tidy(xdmp:document-get
  ("C:\Temp\CoLIndex.html",
      <options xmlns="xdmp:document-get">
        <repair>none</repair>
        <format>text</format>
      </options>))[2]

let $rows := ($doc//html:tr)[html:td/@class eq 'ranking']
let $builder := sem:rdf-builder
          (sem:prefixes("my: http://example.org/vocab/"))
for $row in $rows
let $bnode-name := "_:"|| $row/html:td[@class eq 'ranking']
return (
        $builder($bnode-name, "my:rank", xs:decimal(
                $row/html:td[@class eq 'ranking'] )),
        $builder($bnode-name, "rdfs:label", xs:string(
                $row/html:td[@class eq 'city-name'] )),
        $builder($bnode-name, "my:coli", xs:decimal(
                $row/html:td[@class eq 'inflation'] ))
        )
```

結果は、次のようなインメモリのトリプルとして返されます。



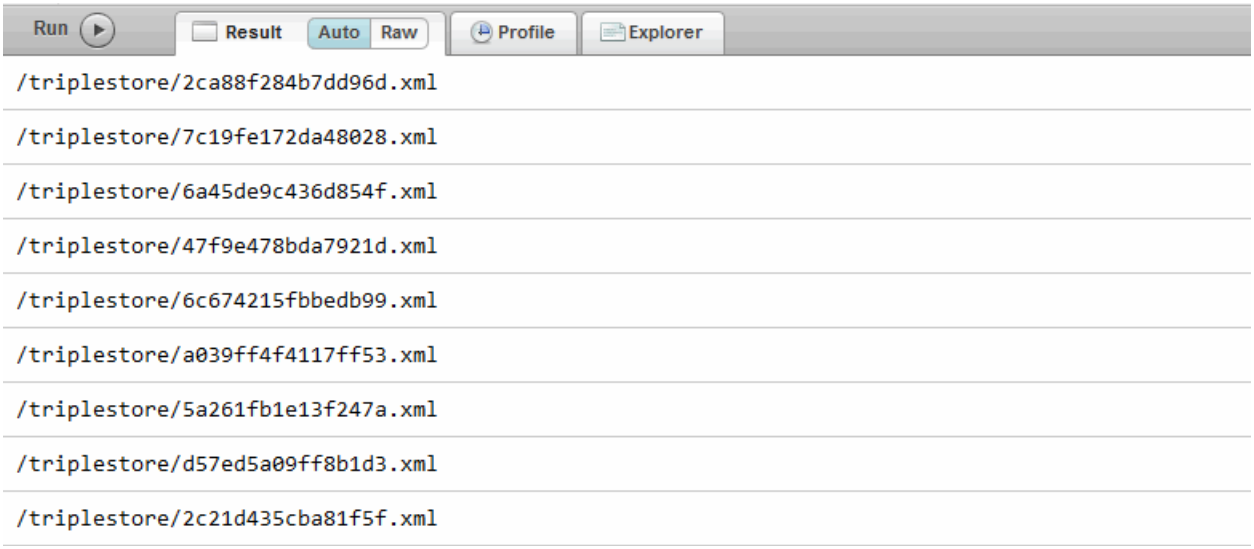
10.4 トリプルのパース

この例では前の例を拡張し、`sem:rdf-insert` 関数を使用してパースしたトリプルをデータベースに挿入します。

```
import module namespace sem = "http://marklogic.com/
  semantics"
   at "/MarkLogic/semantics.xqy";
declare namespace html="http://www.w3.org/1999/xhtml";

let $doc := xdmp:tidy(xdmp:document-get
  ("C:\Temp\CoLIndex.html",
      <options xmlns="xdmp:document-get">
        <repair>none</repair>
        <format>text</format>
      </options>))[2]

let $rows := ($doc//html:tr)[html:td/@class eq 'ranking']
let $builder := sem:rdf-builder(
          sem:prefixes("my: http://example.org/vocab/"))
for $row in $rows
let $bnode-name := "_:"|| $row/html:td[@class eq 'ranking']
let $triples := $row
return (sem:rdf-insert((
```

```
            $builder($bnode-name, "my:rank", xs:decimal
                  ( $row/html:td[@class eq 'ranking'] )),
            $builder($bnode-name, "rdfs:label", xs:string
                  ( $row/html:td[@class eq 'city-name'] )),
            $builder($bnode-name, "my:coli", xs:decimal
                  ( $row/html:td[@class eq 'inflation'] )))
      ))
```

ドキュメント IRI は文字列として返されます。

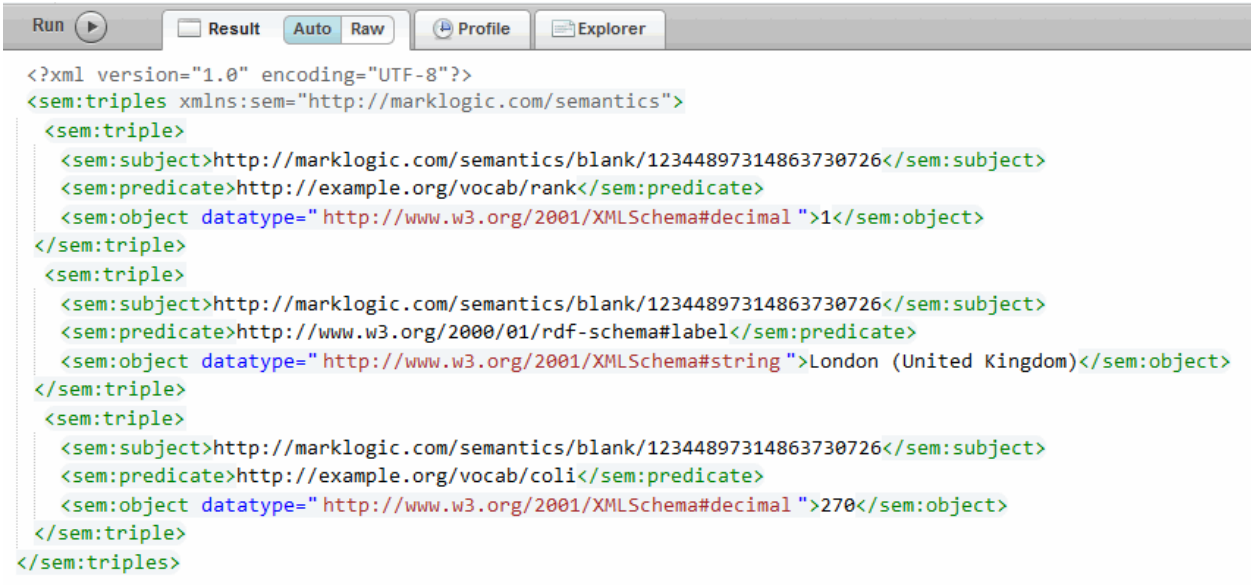


注： XQuery でトリプルを挿入およびパースする詳細については、「XQuery を使用したトリプルのロード」（44 ページ）を参照してください。

パーサーがオンになっている場合は、トリプルとして整形されたマークアップがスキーマの有効なトリプルとして挿入され、トリプルインデックスでインデックス付けされるようになります。「トリプルインデックスの有効化」（58 ページ）を参照してください。

ドキュメントのコンテンツを表示したりトリプルを確認したりするには、`fn:doc` を使用します。

```
fn:doc("/triplestore/2ca88f284b7dd96d.xml")
```



空白ノード識別子（`$bnode-name`）ごとにドキュメントが 1 つ作成されます。

> **注：** 生成プロセス時は `$builder` が状態を維持するため、各空白ノードラベルを追跡する必要はなく、必ず同じ `sem:blank` 値にマッピングされます。

セマンティック API には、N-Quad および Turtle パーサーのリペアオプションが用意されています。通常操作時は、RDF パーサーが次のタスクを実行します。

- Turtle のパースでは、ベース IRI を使用して相対 IRI を解決します。結果が相対の場合は、エラーが発生します。
- N-Quad のパースでは、ベース IRI を使用した解決を行いません。そのため、ドキュメントの IRI が相対の場合は、エラーが発生します。

リペア操作時は、RDF パーサーが次のタスクを実行します。

- Turtle のパースでは、ベース IRI を使用して相対 IRI を解決します。結果として生じる相対 IRI ではエラーは発生しません。
- N-Quad のパースでも、ベース IRI を使用して相対 IRI を解決します。結果として生じる相対 IRI ではエラーは発生しません。

10.5 データを調べる

セマンティック API には、データベース内の RDF データにアクセスするための関数が多数用意されています。このセクションでは、次の内容を取り上げます。

- sem:triple 関数
- sem:transitive-closure

10.5.1 sem:triple 関数

次の表は、トリプルデータの定義または検索に使用する `sem:triple` 関数について説明したものです。

関数	説明
`sem:triple`	RDF トリプルを表現するトリプルオブジェクトを作成します。この RDF トリプルには、主語、述語、目的語、およびオプションのグラフ識別子（グラフ IRI）を表現するアトミック値が含まれます。
`sem:triple-subject`	`sem:triple` 値から主語を返します。
`sem:triple-predicate`	`sem:triple` 値から述語を返します。
`sem:triple-object`	`sem:triple` 値から目的語を返します。
`sem:triple-graph`	`sem:triple` 値からグラフ識別子（グラフ IRI）を返します。

次の例では、`sem:triple` 関数を使用してトリプルを作成しています。このトリプルには、述語として CURIE が、目的語として `rdf:langString` 値が含まれています。また、言語タグとして英語（`en`）が指定されています。

```
sem:triple(sem:iri("http://id.loc.gov/authorities/subjects/
sh85040989"),
sem:curie-expand("skos:prefLabel"),
rdf:langString("Education", "en"))

=>
@prefix xs: <http://www.w3.org/2001/XMLSchema#> .

<http://id.loc.gov/authorities/subjects/sh85040989>
<http://www.w3.org/2004/02/skos/core#prefLabel/>
"Education"@en .
```

10.5.2 sem:transitive-closure

RDF グラフをトラバースして、到達可能性に関する問題を調べたり、RDF データに関する詳細な情報を発見したりするためには、`sem:transitive-closure` 関数を使用します（JavaScript では、`sem.transitiveClosure` 関数を使用します）。例えば、あるノードから到達可能なノードを調べることができます。

`sem:transitive-closure` 関数では、SKOS（Simple Knowledge Organization System）などの語彙を使用できます。SKOS とは、「シソーラス、分類スキーム、件名標目システム、タクソノミーなどの知識組織化体系のための共通データモデル」であり、『W3C SKOS Simple Knowledge Organization System Reference』に記述されています。

http://www.w3.org/TR/skos-reference/

例えば、米国議会法案に関連する件名標目のトリプルで構成されたファイルがあるとします。また、これらのトリプルが SKOS 語彙でマークアップされているとします。トリプルを抽出すると、次のようになります。

```
<http://id.loc.gov/authorities/subjects/sh85002310/>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type/>
<http://www.w3.org/2004/02/skos/core#Concept/> .

<http://id.loc.gov/authorities/subjects/sh85002310/>
<http://www.w3.org/2004/02/skos/core#prefLabel/>
"Agricultural education"@en .

<http://id.loc.gov/authorities/subjects/sh85002310/>
<http://www.w3.org/2008/05/skos-xl#altLabel/>
_:bnode7authoritiessubjectssh85002310 .

_:bnode7authoritiessubjectssh85002310
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type/>
<http://www.w3.org/2008/05/skos-xl#Label/> .

_:bnode7authoritiessubjectssh85002310
<http://www.w3.org/2008/05/skos-xl#literalForm/>
"Education, Agricultural"@en .

<http://id.loc.gov/authorities/subjects/sh85002310/>
<http://www.w3.org/2004/02/skos/core#broader/>
<http://id.loc.gov/authorities/subjects/sh85133121/> .

<http://id.loc.gov/authorities/subjects/sh85002310/>
<http://www.w3.org/2004/02/skos/core#narrower/>
<http://id.loc.gov/authorities/subjects/sh85118332/> .
```

```
<http://id.loc.gov/authorities/subjects/sh85133121/>
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type/>
<http://www.w3.org/2004/02/skos/core#Concept/> .

<http://id.loc.gov/authorities/subjects/sh85133121/>
<http://www.w3.org/2004/02/skos/core#prefLabel/>
"Technical education"@en .
```

このデータセットでは「Technical education」（技術教育）は、`skos:broader` という述語で定義されているとおり「Agricultural education」（農業教育）のより広義の（broader）件名標目です。

```
<http://id.loc.gov/authorities/subjects/sh85002310/>
<http://www.w3.org/2004/02/skos/core#broader/>
<http://id.loc.gov/authorities/subjects/sh85133121/> .
```

注： `skos:broader` は厳密には他動詞ではありませんが、他動詞である `skos:broaderTransitive` のサブプロパティです。

```
<sem:triple>
  <sem:subject>http://id.loc.gov/authorities/subjects/sh85002310</sem:subject>
  <sem:predicate>http://www.w3.org/1999/02/22-rdf-syntax-ns#type</sem:predicate>
  <sem:object>http://www.w3.org/2004/02/skos/core#Concept</sem:object>
</sem:triple>
<sem:triple>
  <sem:subject>http://id.loc.gov/authorities/subjects/sh85002310</sem:subject>
  <sem:predicate>http://www.w3.org/2004/02/skos/core#prefLabel</sem:predicate>
  <sem:object xml:lang="en">Agricultural education</sem:object>
</sem:triple>
<sem:triple>
  <sem:subject>http://id.loc.gov/authorities/subjects/sh85002310</sem:subject>
  <sem:predicate>http://www.w3.org/2008/05/skos-xl#altLabel</sem:predicate>
  <sem:object>http://marklogic.com/semantics/blank/17142585114552908287</sem:object>
</sem:triple>
<sem:triple>
  <sem:subject>http://marklogic.com/semantics/blank/17142585114552908287</sem:subject>
  <sem:predicate>http://www.w3.org/1999/02/22-rdf-syntax-ns#type</sem:predicate>
  <sem:object>http://www.w3.org/2008/05/skos-xl#Label</sem:object>
</sem:triple>
<sem:triple>
  <sem:subject>http://marklogic.com/semantics/blank/17142585114552908287</sem:subject>
  <sem:predicate>http://www.w3.org/2008/05/skos-xl#literalForm</sem:predicate>
  <sem:object xml:lang="en">Education, Agricultural</sem:object>
</sem:triple>
<sem:triple>
  <sem:subject>http://id.loc.gov/authorities/subjects/sh85002310</sem:subject>
  <sem:predicate>http://www.w3.org/2004/02/skos/core#broader</sem:predicate>
  <sem:object>http://id.loc.gov/authorities/subjects/sh85133121</sem:object>
</sem:triple>
```

注： 空白ノード識別子を使用してトリプルを MarkLogic にロードすると、元の `_:bnode` 識別子は MarkLogic の空白ノード識別子に置換されます。例えば元のデータセットの `_:bnode7authoritiessubjectssh85002310` は読み込み時に `<http://marklogic.com/semantics/blank/17142585114552908287>` になります。このアイデンティティは、元の空白ノードに関連するすべてのトリプルで維持されます。

この例では `cts:triples` を使用して、述語が `skos:prefLabel` の CURIE であり、目的語が `Agricultural education` であるトリプルの主語 IRI を調べます。`cts:triples` クエリで見つかった主語 IRI は、その後、より広義の主語タームを深さ `3` まで特定するために、`skos:broader` で使用されます。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/
  semantics" at "/MarkLogic/semantics.xqy";

let $triple-subject := sem:triple-subject(cts:triples((),
sem:curie-
expand("skos:prefLabel"),
rdf:langString("Agricultural education", "en")))
return
sem:transitive-closure($triple-subject, sem:curie-
expand("skos:broader"), 3)

=>
<http://id.loc.gov/authorities/subjects/sh85133121/>
<http://id.loc.gov/authorities/subjects/sh85002310/>
<http://id.loc.gov/authorities/subjects/sh85026423/>
<http://id.loc.gov/authorities/subjects/sh85040989/>
```

想定される IRI の他に、次の主語の IRI は、

- `<http://id.loc.gov/authorities/subjects/sh85002310/>`
- `<http://id.loc.gov/authorities/subjects/sh85133121/>`

次の主語に対する結果でも返されます。

- `<http://id.loc.gov/authorities/subjects/sh85040989/>`
- `<http://id.loc.gov/authorities/subjects/sh85026423/>`

データセットを詳しく調べると、「Agricultural education」および「Technical Education」よりもさらに広義の主語である「Education」（教育）および「Civilization」（文明）の IRI も返されます。

```
<http://id.loc.gov/authorities/subjects/sh85040989/>
<http://www.w3.org/2004/02/skos/core#prefLabel/>
"Education"@en .
<http://id.loc.gov/authorities/subjects/sh85040989/>
<http://www.w3.org/2004/02/skos/core#broader/>
<http://id.loc.gov/authorities/subjects/sh85026423/> .
...
<http://id.loc.gov/authorities/subjects/sh85026423/>
<http://www.w3.org/2004/02/skos/core#prefLabel/>
"Civilization"@en .
...
```

11.0 セマンティック向けのクライアントサイド API

MarkLogic のセマンティックには、クライアントサイド API を通じてアクセスできます。クライアントサイド API は、トリプルやグラフの管理、SPARQL と SPARQL Update、および MarkLogic サーバーの検索機能へのアクセスへのサポートを提供しています。Java クライアントおよび Node.js クライアントのソースは GitHub で入手できます。また、GitHub では、MarkLogic Sesame や MarkLogic Jena も入手できます。

この章は、次のセクションで構成されています。

- Java クライアント API
- MarkLogic Sesame API
- MarkLogic Jena API
- Node.js クライアント API
- オプティック API を使用したクエリ

11.1 Java クライアント API

Java クライアント API では、Java を使用してグラフおよびトリプルを管理したり、MarkLogic（Java クライアント API 3.0.4 以降）の機能である SPARQL クエリおよび SPARQL Update にアクセスしたりできます。

Java クライアント API は、RDF のさまざまな形式をサポートしています。例えば、グラフをデータとともに Turtle シンタックスで記述し、それを N-Triples シンタックスとして読み取り、`JacksonHandle` および SPARQL を使用してクエリできます。

Java クライアント API は GitHub（http://github.com/marklogic/java-client-api）で入手できます。また、Maven のセントラルレポジトリからも入手可能です。セマンティックの操作には、次に示す `com.marklogic.client.semantics` パッケージの Java クライアント API インターフェイスを使用します。

- GraphManager
- SPARQLQueryManager
- SPARQLQueryDefinitions
- GraphPermissions
- SPARQLBindings
- RDFTypes
- RDFMimeTypes
- SPARQLRuleset

次のトピックでは、Java クライアント API のセマンティック機能について詳しく説明します。

- Java クライアント API の核となる概念
- グラフ管理
- SPARQL クエリ

11.1.1 Java クライアント API の核となる概念

このセクションでは、Java クライアント API の概念について簡単に説明します。これらの概念は、この API を使用するセマンティックの例を理解するのに役立ちます。詳細については、『*Java Application Developer's Guide*』を参照してください。

Java クライアント API を通じた MarkLogic サーバーとのすべてのインタラクションには、`DatabaseClient` オブジェクトが必要になります。`DatabaseClient` オブジェクトは、MarkLogic サーバーへの接続詳細をカプセル化します。`DatabaseClientFactory.newClient` を使用して `DatabaseClient` オブジェクトを作成します。詳細については、『*Java Application Developer's Guide*』の「Creating, Working With, And Releasing a Database Client」を参照してください。

例えば、次のコードは、`DatabaseClient` を作成してから、それを使用して `GraphManager` オブジェクトを作成します。例である `DatabaseClient` は、ローカルホストにある MarkLogic インストールへの接続詳細を表しています。このインストールは、ポート 8000 で listen し、「myDatabase」という名前のデータベースで動作しています。クライアントはユーザー「myuser」と digest 認証を使用します。

```
import com.marklogic.client.DatabaseClientFactory;
import com.marklogic.client.DatabaseClient;
import com.marklogic.client.DatabaseClient.DigestAuthContext;
...
DatabaseClient client = DatabaseClientFactory.newClient(
    "localhost", 8000, "myDatabase",
    new DigestAuthContext("username", "password"));
GraphManager gmgr = client.newGraphManager();
```

一般に、Java クライアント API は、`Handle` オブジェクトを使用してアプリケーションと MarkLogic サーバーの間で交換されるデータを参照します。例えば、`FileHandle` を使用してファイルからトリプルを読み取ったり、インメモリのシリアライズされたトリプルを参照する `StringHandle` を作成したりできます。詳細については、『*Java Application Developer's Guide*』の「Using Handles for Input and Output」を参照してください。

次の例は、ファイルから Turtle 形式のトリプルを FileHandle に読み取る方法を示します。結果となるハンドルは、GraphManager.write のような操作への入力として使用できます。

```
import com.marklogic.client.io.FileHandle;
...
FileHandle fileHandle =
  new FileHandle(new File("example.ttl"))
    .withMimetype(RDFMimeTypes.TURTLE);
```

11.1.2 グラフ管理

グラフの作成、読み取り、更新、削除など、グラフの管理操作を実行するには、GraphManager インターフェイスを使用します。次の表に、主な GraphManager のメソッドを要約します。詳細については、『*Java Client API Documentation*』を参照してください。

GraphManager のメソッド	説明
read readAs	特定のグラフからトリプルを取得します。
write writeAs	グラフを作成または上書きします。グラフがすでにある場合、そのグラフを削除してから入力データでグラフを再作成した場合と同じ結果になります。
replaceGraphs replaceGraphsAs	すべてのグラフからトリプルを削除し、クアッドを入力データセットに挿入します。非管理対象トリプルは影響を受けません。最初に GraphManager.deleteGraphs を呼び出してから、クアッドを挿入する操作と同じ結果になります。
merge mergeAs	トリプルを名前付きグラフまたはデフォルトグラフに追加します。グラフは存在しない場合、作成されます。
mergeGraphs mergeGraphsAs	クアッドを入力クアッドデータで指定されたグラフに追加します。まだ存在しないグラフがあれば、作成されます。
delete	特定のグラフを削除します。
deleteGraphs	すべてのグラフを削除します。非管理対象トリプルは影響を受けません。

入力にグラフ URI が利用できるメソッドでデフォルトグラフを参照するには、GraphManager.DEFAULT_GRAPH を使用します。

以前の例で示したように、DatabaseClient.newGraphManager を使用して GraphManager オブジェクトを作成します。次のコードは、DatabaseClient を作成してから、それを使って GraphManager を作成します。

```
DatabaseClient client = DatabaseClientFactory.newClient(...);
GraphManager gmgr = client.newGraphManager();
```

グラフを作成（または置換）するには、GraphManager.write を使用します。次の例は、ファイルから Turtle 形式のトリプルを FileHandle に読み込んでから、そのトリプルを URI「myExample/graphURI」の名前付きグラフに書き込みます。

```
FileHandle fileHandle =
  new FileHandle(new File("example.ttl"))
    .withMimetype(RDFMimeTypes.TURTLE);
gmgr.write("myExample/graphURI", fileHandle);
```

GraphManager.write を使用してクアッドを読み込んだ場合、write に渡されたグラフ URI パラメータが優先され、クアッド内のグラフ URI は無視されることに注意してください。

トリプルをマージして既存のグラフに入れるには、GraphManager.merge を使用します。次の例は、Turtle 形式の単一のトリプルをデフォルトグラフに追加しています。トリプルは StringHandle を介し、トリプルの形式を示す StringHandle.withMimetype を使用して操作に渡されています。

```
StringHandle stringHandle = new StringHandle()
    .with("<http://example.org/subject2> " +
          "<http://example.org/predicate2> " +
          "<http://example.org/object2> .")
    .withMimetype(RDFMimeTypes.TURTLE);
gmgr.merge("myExample/graphUri", stringHandle);
```

各 Handle ではなく GraphManager オブジェクトで、デフォルトの MIME タイプを設定することもできます。詳細については、『*Java Client API Documentation*』の「GraphManager.setMimetype」を参照してください。

グラフのコンテンツを読み取るには、GraphManager.read を使用します。出力 Handle で setMimetype を呼び出すか、GraphManager オブジェクトでデフォルトの MIME タイプを設定して、出力トリプルの形式を指定します。次の例は、デフォルトグラフのコンテンツを Turtle 形式のトリプルとして取得しています。結果は、StringHandle を介して、文字列として入手できます。

```
StringHandle triples = gmgr.read(
    GraphManager.DEFAULT_GRAPH,
    new StringHandle().withMimetype(RDFMimeTypes.TURTLE);
//...work with the triples as one big string
```

グラフを削除するには、GraphManager.delete を使用します。すべてのグラフを削除するには、GraphManager.deleteGraphs を使用します。次の例は、URI が「myExample/graphUri」のグラフを削除しています。

```
gmrg.delete("myExample/graphUri");
```

また、GraphManager インターフェイスを使用してグラフパーミッションを管理することもできます。これを行うには、パーミッションを個別のグラフ操作（書き込み、マージなど）に渡すか、GraphManager.writePermissions など、明示的なパーミッション管理メソッドを呼び出します。GraphManager.permission を使用して、グラフに適用するパーミッションの集まりを作成します。

例えば、次のコードは、グラフ「myExample/graphUri」のパーミッションを新しいパーミッションの集まりに置換しています。

```
import com.marklogic.client.semantics.Capability;
...
gmgr.writePermissions(
  "http://myExample/graphUri",
  gmgr.permission("role1", Capability.READ)
      .permission("role2", Capability.READ,
       Capability.UPDATE));
```

次の例は、同じパーミッションをグラフマージの一部として追加しています。

```
gmgr.merge(
  "myExample/graphUri", someTriplesHandle,
  gmgr.permission("role1", Capability.READ)
      .permission("role2", Capability.READ,
       Capability.UPDATE));
```

11.1.3 SPARQL クエリ

RDF データセットのクエリには、SPARQLQueryManager インターフェイスを使用します。このとき、SELECT、CONSTRUCT、DESCRIBE、および ASK クエリを使用します。SPARQLQueryManager インターフェイスは、更新と読み取りの両方のクエリをサポートしています。SPARQLQueryDefinition インターフェイスは、クエリ、バインドの形式をカプセル化します。SPARQL クエリで使用される変数をバインドするには、SPARQLBindings インターフェイスを使用します。

SPARQL の読み取りまたは更新クエリの評価は、次のような基本的な手順で構成されます。

1. DatabaseClient.newSPARQLQueryManager を使用してクエリマネージャを作成します。例えば以下のようになります。

```
DatabaseClient client = ...;
SPARQLQueryManager sqmgr = client.newSPARQLManager();
```

2. SPARQLQueryManager.newSPARQLQueryDefinition を使用してクエリを作成し、必要に応じてクエリを設定します。例えば以下のようになります。

```
SPARQLQueryDefinition query = sqmgr.newSPARQLQueryDefinition(
  "SELECT * WHERE { ?s ?p ?o } LIMIT 10")
  .withBinding("o", "http://example.org/object1");
```

3. クエリを評価し、結果を受け取ります。このとき、SPARQLQueryManager の execute* メソッドのいずれかを呼び出します。例えば以下のようになります。

```
JacksonHandle results = new JacksonHandle();
results.setMimetype(SPARQLMimeTypes.SPARQL_JSON));
results = sqmgr.executeSelect(query, results);
```

SPARQLQueryManager インターフェイスには、サポートされている SPARQL クエリタイプを評価するための次のメソッドが含まれます。

- executeSelect
- executeAsk
- executeConstruct
- executeDescribe
- executeUpdate

次の例は、SELECT クエリの評価を実現しています。クエリには、変数「o」のバインドが含まれます。クエリの結果は、JSON として返されます。

```
import com.marklogic.client.semantics.SPARQLQueryManager;
import com.marklogic.client.semantics.SPARQLQueryDefinition;
import com.marklogic.client.io.JacksonHandle;
```

```
// create a query manager
SPARQLQueryManager sparqlMgr =
databaseClient.newSPARQLQueryManager();

// create a SPARQL query
String sparql = "SELECT * WHERE { ?s ?p ?o } LIMIT 10";
SPARQLQueryDefinition query =
sparqlMgr.newQueryDefinition(sparql)
    .withBinding("o", "http://example.org/object1");

// evaluate the query
JacksonHandle handle = new JacksonHandle();
handle.setMimetype(SPARQLMimeTypes.SPARQL_JSON);
JacksonHandle results = sparqlMgr.executeSelect(query,
handle);

// work with the results
JsonNode tuples =
results.get().path("results").path("bindings");
for ( JsonNode row : tuples ) {
    String s = row.path("s").path("value").asText();
    String p = row.path("p").path("value").asText();
    ...
}
```

変数バインドの定義、クエリの適用先であるデフォルトグラフまたは名前付きグラフの設定、最適化レベルの設定など、SPARQL クエリのさまざまな側面を設定できます。詳細については、『*Java Client API Documentation*』の「`SPARQLQueryDefinition`」を参照してください。

`SPARQLQueryDefinition.withBinding` 定義を使用して変数バインドを一度に 1 つずつ定義できます。または、`SPARQLBindings` を使用してバインドの集まりを構築し、`SPARQLQueryDefinition.setBindings` を使用してクエリに接続することもできます。

例えば以下のようになります。

```
// incrementally attach bindings to a query
SPARQLQueryDefinition query = ...;
query.withBinding("o", "http://example.org/object1")
     .withBinding(...);

// build up a set of bindings and attach them to a query
SPARQLBindings bindings = new SPARQLBindings();
bindings.bind("o", "http://example.org/object1");
```

```
bindings.bind(...);
query.setBindings(bindings);
```

バインドを使用したその他の例については、「SPARQLBindings」を参照してください。

SPARQL の SELECT クエリを評価すると、デフォルトでは、すべての結果が返されます。SPARQLQueryManager.setPageLength または SPARQL の LIMIT 節を使用すると、1 つの「ページ」で返される結果の数を制限できます。異なるページ開始位置を指定して executeSelect を繰り返し呼び出すと、後続の結果ページを取得できます。例えば以下のようになります。

```
// Change the max page length
sqmgr.setPageLength(NRESULTS);

// Fetch at most the first N results
long start = 1;
JacksonHandle results = sparqlMgr.executeSelect(query,
handle, start);

// Fetch the next N results
start += N;
JacksonHandle results = sparqlMgr.executeSelect(query,
handle, start);
```

Java クライアント API には、ビルトインルールセットの集まりと、カスタムルールセットを使用できるようにするファクトリメソッドを持つ SPARQLRuleset クラスが含まれます。ルールセットをクエリに関連付けるには、SPARQLQueryDefinition.withRuleset または SPARQLQueryDefinition.setRulesets を使用します。

デフォルトの推論は、SPARQLQueryDefinition.withIncludeDefaultRulesets(Boolean) メソッドを使用してオン / オフを切り替えることができます。デフォルトでは、オンになっています。ルールセットの保守は、管理 API の機能です。「ルールセット」（149 ページ）を参照してください。

MarkLogic Java クライアントのセマンティックメソッドの全リストについては、『*Java Client API Documentation*』の「GraphManager」および「SPARQLQueryManager」を参照してください。

11.2 MarkLogic Sesame API

Sesame とは、RDF データの加工および処理用の Java API です。これには、このデータの作成、パース、格納、推論、およびクエリが含まれます。Sesame API を使い慣れた Java 開発者の方は、同じ API を使用して MarkLogic の RDF データにアクセスできるよ

うになりました。MarkLogic Sesame API は、フル機能で使いやすいインターフェイスです。このインターフェイスは、MarkLogic のセマンティック機能へのシンプルなアクセスを提供します。MarkLogic Sesame API には、永続性レイヤーの一部である `MarkLogicRepository` インターフェイスが含まれます。

MarkLogic Sesame API をインクルードすると、SPARQL クエリおよび SPARQL Update に標準の Sesame を使用して、MarkLogic をトリプルストアとして活用できます。MarkLogic Sesame API は、標準の Sesame API を拡張しているため、複合クエリ、変数バインド、トランザクションを行うこともできます。`MarkLogicRepository` クラスは、トランザクションと変数バインドの両方へのサポートを提供しています。

次の例は、MarkLogic Sesame API を使用して、MarkLogic データベースに格納されているセマンティックデータに対し、SPARQL クエリと SPARQL Update を実行しています。まず、`MarkLogicRepository` をインスタンス化し、デフォルトルールセットとデフォルトのパーミッションを定義しています。SPARQL クエリが、結果を複合クエリの結果に制限する追加クエリによって制約されていることに注意してください。

```
package com.marklogic.semantics.sesame.examples;

import com.marklogic.client.DatabaseClient;
import com.marklogic.client.DatabaseClientFactory;
import com.marklogic.client.Database
ClientFactory.DigestAuthContext;
import com.marklogic.client.io.Format;
import com.marklogic.client.io.StringHandle;
import com.marklogic.client.query.QueryManager;
import com.marklogic.client.query.RawCombinedQueryDefinition;
import com.marklogic.client.query.StringQueryDefinition;
import com.marklogic.client.semantics.Capability;
import com.marklogic.client.semantics.GraphManager;
import com.marklogic.client.semantics.SPARQLRuleset;
import com.marklogic.semantics.sesame.MarkLogicRepository;
import com.marklogic.semantics.sesame.MarkLogic
RepositoryConnection;
import com.marklogic.semantics.sesame.query.MarkLogic
TupleQuery;
import com.marklogic.semantics.sesame.query.MarkLogic
UpdateQuery;
import org.openrdf.model.Resource;
import org.openrdf.model.URI;
import org.openrdf.model.ValueFactory;
import org.openrdf.model.vocabulary.FOAF;
import org.openrdf.model.vocabulary.RDF; import
org.openrdf.model.vocabulary.RDFS;
```

```
import org.openrdf.model.vocabulary.XMLSchema;
import org.openrdf.query.*;
import org.openrdf.repository.RepositoryException;
import org.openrdf.rio.RDFParseException;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.IOException;

public class Example2_Advanced {

  protected static Logger logger =
    LoggerFactory.getLogger(Example2_Advanced.class);

  public static void main(String... args) throws
    RepositoryException, IOException, RDFParseException,
    MalformedQueryException, QueryEvaluationException {

    // instantiate MarkLogicRepository with Java api
    // client DatabaseClient
    DatabaseClient adminClient =
      DatabaseClientFactory.newClient("localhost", 8200,
        new DigestAuthContext("username", "password"));
    GraphManager gmgr = adminClient.newGraphManager();
    QueryManager qmgr = adminClient.newQueryManager();

    // create repo and init
    MarkLogicRepository repo = new MarkLogicRepository
(adminClient);
    repo.initialize();

    // get repository connection
    MarkLogicRepositoryConnection conn = repo.getConnection();

    // set default rulesets
    conn.setDefaultRulesets(SPARQLRuleset.ALL_VALUES_FROM);

     // set default perms
     conn.setDefaultGraphPerms(
       gmgr.permission("admin", Capability.READ)
           .permission("admin", Capability.EXECUTE));

     // set a default Constraining Query
     StringQueryDefinition stringDef =
       qmgr.newStringDefinition().withCriteria("First");
     conn.setDefaultConstrainingQueryDefinition(stringDef);
```

```
// return number of triples contained in repository
logger.info("1. number of triples: {}", conn.size());

// add a few constructed triples
Resource context1 = conn.getValueFactory()
  .createURI("http://marklogic.com/examples/context1");
Resource context2 = conn.getValueFactory()
  .createURI("http://marklogic.com/examples/context2");
ValueFactory f= conn.getValueFactory();
String namespace = "http://example.org/";
URI john = f.createURI(namespace, "john");

//use transactions to add triple statements
conn.begin();
conn.add(john, RDF.TYPE, FOAF.PERSON, context1);
conn.add(john, RDFS.LABEL,
  f.createLiteral("John", XMLSchema.STRING), context2);
conn.commit();

logger.info("2. number of triples: {}", conn.size());

// perform SPARQL query
String queryString = "select * { ?s ?p ?o }";
MarkLogicTupleQuery tupleQuery =
  conn.prepareTupleQuery(QueryLanguage.SPARQL,
   queryString);

// enable rulesets set on MarkLogic database
tupleQuery.setIncludeInferred(true);

// set base uri for resolving relative uris
tupleQuery.setBaseURI("http://www.example.org/base/");

// set rulesets for infererencing
tupleQuery.setRulesets(SPARQLRuleset.ALL_VALUES_FROM,
  SPARQLRuleset.HAS_VALUE);

// set a combined query
String combinedQuery =
  "{\"search\":"+ "{\"qtext\":\"*\"}}";
RawCombinedQueryDefinition rawCombined =
  qmgr.newRawCombinedQueryDefinition(
    new StringHandle()
      .with(combinedQuery)
      .withFormat(Format.JSON));
tupleQuery.setConstrainingQueryDefinition(rawCombined);
```

```
        // evaluate query with pagination
        TupleQueryResult results = tupleQuery.evaluate(1,10);

        //iterate through query results
        while(results.hasNext()){
          BindingSet bindings = results.next();
          logger.info("subject:{}",bindings.getValue("s"));
          logger.info("predicate:{}", bindings.getValue("p"));
          logger.info("object:{}", bindings.getValue("o"));
        }
        logger.info("3. number of triples: {}", conn.size());

        //update query
        String updatequery = "INSERT DATA { " +
          "GRAPH <http://marklogic.com/test/context10> {" +
          "<http://marklogic.com/test/subject> <pp1> <oo1> } }";
        MarkLogicUpdateQuery updateQuery =
          conn.prepareUpdate(QueryLanguage.SPARQL, updatequery,
            "http://marklogic.com/test/baseuri");

        // set perms to be applied to data
        updateQuery.setGraphPerms(
          gmgr.permission("admin", Capability.READ)
              .permission("admin", Capability.EXECUTE));

        try {
          updateQuery.execute();
        } catch (UpdateExecutionException e) {
          e.printStackTrace();
        }

        logger.info("4. number of triples: {}", conn.size());

        // clear all triples
        conn.clear();
        logger.info("5. number of triples: {}", conn.size());

        // close connection and shutdown repository
        conn.close();
        repo.shutDown();
      }
    }
```

MarkLogic Sesame API は、GitHub（http://github.com/marklogic/marklogic-sesame）で javadocs や例とともに入手できます。

次に、MarkLogic Sesame API の主なインターフェイスをリストします。

- MarkLogicRepository
- MarkLogicRepositoryConnection
- MarkLogicQuery
    - MarkLogicTupleQuery
    - MarkLogicGraphQuery
    - MarkLogicBooleanQuery
    - MarkLogicUpdateQuery

11.3 MarkLogic Jena API

Jena とは、RDF データの加工および処理用の Java API です。これには、このデータの作成、パース、格納、推論、およびクエリが含まれます。Jena API を使い慣れた Java 開発者の方は、同じ API を使用して MarkLogic の RDF データにアクセスできるようになりました。MarkLogic Jena API は、フル機能で使いやすいインターフェイスです。このインターフェイスは、MarkLogic のセマンティック機能へのシンプルなアクセスを提供します。

MarkLogic Jena API をインクルードすると、SPARQL クエリおよび SPARQL Update に標準の Jena を使用して、MarkLogic をトリプルストアとして活用できます。MarkLogic Jena API は、Jena API を拡張しているため、複合クエリ、変数バインド、トランザクションを行うこともできます。`MarkLogicDataSet` クラスは、トランザクションと変数バインドの両方へのサポートを提供しています。

次の例では、MarkLogic Jena を使用してクエリを実行する方法を示します。

```
package com.marklogic.jena.examples;

import org.apache.jena.riot.RDFDataMgr;
import org.apache.jena.riot.RDFFormat;

import com.hp.hpl.jena.graph.NodeFactory;
import com.hp.hpl.jena.query.QueryExecutionFactory;
import com.hp.hpl.jena.query.QuerySolution;
import com.hp.hpl.jena.query.ResultSet;
import com.hp.hpl.jena.update.UpdateExecutionFactory;
import com.hp.hpl.jena.update.UpdateFactory;
import com.hp.hpl.jena.update.UpdateProcessor;
import com.hp.hpl.jena.update.UpdateRequest;
import com.marklogic.semantics.jena.MarkLogicDatasetGraph;
```

```
/**
 * How to run queries.
 */
public class SPARQLUpdateExample {

    private MarkLogicDatasetGraph dsg;

    public SPARQLUpdateExample() {
        dsg = ExampleUtils.loadPropsAndInit();
    }

    private void run() {
        dsg.clear();

        String insertData = "PREFIX foaf:
        <http://xmlns.com/foaf/0.1/> "
                + "PREFIX : <http://example.org/> "
                +"INSERT DATA {GRAPH :g1 {"
                + ":charles a foaf:Person ; "
                + "        foaf:name \"Charles\" ;"
                + "        foaf:knows :jim ."
                + ":jim    a foaf:Person ;"
                + "        foaf:name \"Jim\" ;"
                + "        foaf:knows :charles ."
                + "} }";

        System.out.println("Running SPARQL update");

        UpdateRequest update = UpdateFactory.create
          (insertData);
        UpdateProcessor processor =
          UpdateExecutionFactory.create(update, dsg);
        processor.execute();

        System.out.println("Examine the data as JSON-LD");
        RDFDataMgr.write(System.out,
         dsg.getGraph(NodeFactory.createURI
         ("http://example.org/g1")),
          RDFFormat.JSONLD_PRETTY);

        System.out.println("Remove it.");

        update = UpdateFactory.create("PREFIX :
        <http://example.org/> DROP GRAPH :g1");
```

```
            processor = UpdateExecutionFactory.create(update,
            dsg);
            processor.execute();
            dsg.close();
        }

        public static void main(String... args) {
            SPARQLUpdateExample example = new
            SPARQLUpdateExample();
            example.run();
         }

    }
```

MarkLogic Jena のソースは、GitHub（http://github.com/marklogic/marklogic-jena）で javadocs や例とともに入手できます。

次に、MarkLogic Jena API の主なインターフェイスをリストします。

- MarkLogicDatasetGraph
- MarkLogicDatasetGraphFactory
- MarkLogicQuery
- MarkLogicQueryEngine

11.4 Node.js クライアント API

Node.js クライアント API は、グラフに対する CRUD 操作（トリプルとグラフの作成、読み取り、更新、および削除）に使用できます。`DatabaseClient.graphs.write` 関数を使用すると、トリプルを含んでいるグラフを作成できます。`DatabaseClient.graphs.read` 関数はグラフから読み取ります。`DatabaseClient.graphs.remove` 関数はグラフを削除します。セマンティックデータをクエリするには、`DatabaseClient.graphs.sparql` 関数を使用します。

詳細については、『*Node.js Application Developer's Guide*』の「Working With Semantic Data」を参照してください。Node.js クライアントのソースは、GitHub（http://github.com/marklogic/node-client-api）にあります。その他の操作については、『*Node.js Client API Reference*』を参照してください。

> **注：** これらの操作を使用できるのは、グラフに含まれている管理対象トリプルのみです。組み込みトリプルは、Node.js クライアント API を使用して操作することはできません。

11.5 オプティック API を使用したクエリ

オプティック API を使用すると、クライアントサイドクエリとサーバーサイドクエリの両方でセマンティックトリプルを検索および操作できます。オプティックは、Java クライアント API や REST クライアント API でトリプルデータのクライアントサイドクエリに使用できますが、Node.js では使用できません。詳細については、『*Java Application Developer's Guide*』の Optic Java API for Relational Operations、および『*REST Application Developer's Guide*』の行マネージャと行エンドポイントに関する記述を参照してください。

オプティック API を使用したサーバーサイドクエリの詳細については、「オプティック API を使用したトリプルのクエリ」（144 ページ）を参照してください。また、オプティック API の「`op:from-triples`」関数や「`op.fromTriples` 関数についての記述、および『*Application Developer's Guide*』の「Data Access Functions」セクションも参照してください。

12.0 XQuery およびサーバーサイド JavaScript でのトリプルの挿入、削除、および修正

XQuery またはサーバーサイド JavaScript で MarkLogic の xdmp ビルトインを使用すると、トリプルを修正できます。MarkLogic によって管理されているトリプル（ドキュメントルートが sem:triples）は、SPARQL Update を使用して修正できます。管理対象トリプルの修正の詳細については、「SPARQL Update の使用」（173 ページ）を参照してください。

「非管理対象」トリプル（別のドキュメントに組み込まれ、要素ノード sem:triple を持つトリプル）は、XQuery またはサーバーサイド JavaScript と、xdmp ビルトインを使用した場合にのみ修正できます。データストア内のトリプル（管理対象トリプルまたは非管理対象トリプル）に対して更新を実行するには、新しいトリプルを挿入し、既存のトリプルを削除します。既存のトリプルの更新は行われません。更新操作は、実際には INSERT/DELETE という手順です。

この章は、次のセクションで構成されています。

12.1 トリプルの更新

XQuery またはサーバーサイド JavaScript の関数で、INSERT/DELETE を使用すると、データベース内の既存のトリプルを更新してノードを置換できます。管理対象トリプルの不正確なデータを修正する場合は、sem:database-nodes（サーバーサイド JavaScript の場合は sem.databaseNode）および xdmp:node-replace（サーバーサイド JavaScript の場合は xdmp.nodeReplace）関数を使用します。

例えば、データにリソース「André-Marie Ampère」（「Andr%C3%A9-Marie_Amp%C3%A8re」と入力）の主語ノードが含まれているとします。

```
<sem:triples xmlns:sem="http://marklogic.com/semantics">
 <sem:triple>
  <sem:subject>http://dbpedia.org/resource/Andr%C3%A9-Marie_
    Amp%C3%A8re
  </sem:subject>
  <sem:predicate>http://www.w3.org/1999/02/22-rdf-syntax-
ns#type/
  </sem:predicate>
  <sem:object>http://xmlns.com/foaf/0.1/Person/
  </sem:object>
 </sem:triple>
</sem:triples>
```

次の例では、sem:rdf-builder 関数により、1 つのトリプルが構築され、sem:database-nodes 関数に渡されています。また、新たに $replace トリプルが作成され、xdmp:node-replace 関数に渡されます。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

let $builder := sem:rdf-builder(sem:prefixes("dbpedia:
http://dbpedia.org/resource/"))

let $triple := $builder("dbpedia:Andr%C3%A9-
Marie_Amp%C3%A8re", "a", "foaf:Person")

let $node := sem:database-nodes($triple)

let $replace :=
<sem:triple>
<sem:subject>http://dbpedia.org/resource/André-Marie_Ampère/
</sem:subject>
{$node[1]/sem:predicate, $node[1]/sem:object}
</sem:triple>

return $node ! xdmp:node-replace($node, $replace);
```

この例では、変数 $builder にバインドされたトリプルだけが式にマッチし、更新（置換）されます。

この例をサーバーサイド JavaScript で記述すると、次のようになります。

```
declareUpdate();
var sem = require("/MarkLogic/semantics.xqy");
var builder = sem.rdfBuilder();
var triple = xdmp.apply(builder, "dbpedia:André-Marie_Ampère",
  "a","foaf:Person");
var node = sem.databaseNodes(triple);

var replace = new NodeBuilder();
replace.startElement("sem:triple","http://marklogic.com/
  semantics/");
replace.addElement("sem:subject","http://dbpedia.org/resource
  /André-Marie_Ampère/","http://marklogic.com/semantics");
replace.addNode(node.getElementsByTagNameNS("http://marklogi
  c.com/semantics","predicate")[0]);
replace.addNode(node.getElementsByTagNameNS("http://marklogi
  c.com/semantics","object")[0]);
```

```
replace.endElement();
for (var node of nodes) {
xdmp.nodeReplace(node, replace);
};
```

更新するトリプルが複数ある場合は、XQuery またはサーバーサイド JavaScript を使用し（または、管理対象トリプルの場合は、SPARQL Update を使用）、マッチするトリプルを探し、ノードにわたって反復して置換できます。

この例では、`cts:triples` の呼び出しによって、主語の位置に「Andr%C3%A9-Marie_Amp%C3%A8re」を持つすべてのトリプルを調べ、見つかったそれぞれの項目を置換します。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/Marklogic/semantics.xqy";

let $triples :=
cts:triples(sem:iri("http://dbpedia.org/resource/Andr%C3%A9-
    Marie_Amp% C3%A8re"),()())
for  $triple in  $triples
let $node := sem:database-nodes($triple)
let $replace :=
    <sem:triple>
       <sem:subject>http://dbpedia.org/resource/André-
Marie_Ampère
</sem:subject>
       {$node/sem:predicate, $node/sem:object}
    </sem:triple>
return $node ! xdmp:node-replace(., $replace)
```

どちらの例でも、置換が行われているため、空のシーケンスが返されます。更新が行われたかどうかは、次のようなシンプルな `cts:triples` の呼び出しを使用して確認できます。

```
cts:triples(sem:iri("http://dbpedia.org/resource/
André-Marie_Ampère"),
         ()())
```

注： `xdmp:node-replace` 関数を使用すると新しいフラグメントが作成され、古いフラグメントは削除されます。マージが実行されると、削除済みフラグメントは恒久的に削除されます。管理者がオフにしていない限り、自動マージが実行されます。

12.2 トリプルの削除

このセクションでは、MarkLogic で RDF データを削除する手法について説明します。次のトピックから構成されています。

- XQuery またはサーバーサイド JavaScript でのトリプルの削除
- REST API を使用したトリプルの削除

12.2.1 XQuery またはサーバーサイド JavaScript でのトリプルの削除

データベースからトリプルを削除する場合は、複数の関数を使用できます。このセクションでは、次の関数を取り上げます。

- sem:graph-delete
- xdmp:node-delete
- xdmp:document-delete

12.2.1.1 sem:graph-delete

注： この関数は、管理対象トリプルに対してのみ機能します。

`sem:graph-delete` 関数を使用すると、名前付きグラフ内のすべての管理対象トリプルドキュメントを削除できます。パラメータとして、グラフ IRI を指定します。

例えば以下のようになります。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

sem:graph-delete(sem:iri("mynamedgraph"))
```

サーバーサイド JavaScript の場合、コマンドは次のようになります。

```
var sem = require("/marklogic/semantics.xqy");

sem.graphDelete(sem.iri("mynamedgraph"))
```

次の例では、デフォルトグラフのすべての管理対象トリプルを削除します。その他に名前付きグラフが存在しない場合は、これによってデータベースからすべてのトリプルが削除されることがあります。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";
```

```
sem:graph-delete(sem:iri("http://marklogic.com/semantics#default-
graph
"))
```

注： sem:graph-delete 関数は、グラフストア API によって挿入されたトリプル（ドキュメントのルート要素が sem:triple）のみを削除します。そのため、特定の名前付きグラフを削除しても、埋め込みトリプルが含まれるドキュメント（sem:triples 要素ノードを持つドキュメント）には影響がないため、グラフが存在し続ける可能性があります。

12.2.1.2 xdmp:node-delete

データベースからトリプルの集まりを削除するには、sem:database-nodes 関数と xdmp:node-delete を使用します。この関数は、管理対象トリプルまたは非管理対象トリプルで機能します。

例えば以下のようになります。

```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

let $triples :=
cts:triples(sem:iri("http://www.rdfabout.com/rdf/usgov/congress
/people/D000596"),()())

for $triple in $triples
return (sem:database-nodes($triple) ! xdmp:node-delete(.))
```

注： 単一のドキュメントからすべてのトリプルが削除されている場合、このクエリを実行しても空の sem:triple ドキュメント要素は削除されません。

この例をサーバーサイド JavaScript で記述すると、次のようになります。

```
var sem = require("/MarkLogic/semantics.xqy");

let $triples :=
cts.triples(sem.iri("http://www.rdfabout.com/rdf/usgov/
congress/people/D000596"),()())

for $triple in $triples
return (sem.databaseNodes($triple) ! xdmp.nodeDelete(.))
```

12.2.1.3 xdmp:document-delete

`xdmp:document-delete` 関数を使用すると、データベースからトリプルが含まれるドキュメントを削除できます。この関数は、ドキュメントとそのすべてのプロパティを削除し、管理対象トリプルと非管理対象トリプルの両方で機能します。パラメータとして、削除するドキュメントの IRI を指定します。

例えば以下のようになります。

```
xquery version "1.0-ml";

xdmp:document-delete("example.xml")
```

注： `xdmp:document-delete` 関数は、ドキュメントとプロパティを削除します。そのドキュメントに埋め込まれたすべてのトリプルも削除されます。あるディレクトリ内のドキュメントをすべて削除するには、`xdmp:directory-delete` 関数を使用します。

この例をサーバーサイド JavaScript で記述すると、次のようになります。

```
var sem = require("/MarkLogic/semantics.xqy");

xdmp.documentDelete("example.xml")
```

12.2.2 REST API を使用したトリプルの削除

REST API を使用すると、`DELETE` リクエストを `DELETE:/v1/graphs` サービスに送信することで、デフォルトグラフまたは名前付きグラフのトリプルを削除できます。名前付きグラフからトリプルを削除するには、`curl` を使用して、次の形式の `DELETE` リクエストを送信します。

```
http://<host:port>/<version>/graphs?graph=graph-iri
```

graph-iri は名前付きグラフの IRI です。

リクエストにおける名前付きグラフの IRI は、`http://<host:port>/<version>/graphs?default` です。例えば、次の `DELETE` リクエストは、ポート 8321 でデフォルトグラフからすべてのトリプルを削除します。

```
#Windows users, see Modifying the Example Commands for Windows

$ curl --anyauth --user user:password -X DELETE \
http://localhost:8321/v1/graphs?default
```

注： データセットを削除する前は確認チェックが行われないため、グラフを指定するときは注意が必要です。

次の curl コマンドは、mygraph という名前のグラフでトリプルを削除します。

```
#Windows users, see Modifying the Example Commands for Windows

$ curl --anyauth --user user:password -X DELETE \
http://localhost:8321/v1/graphs?graph=http://marklogic.com
/semantics#mygraph/
```

sem:graph-delete 関数と同様に、DELETE リクエストは、格納先ドキュメントのルート要素が sem:triples であるトリプル（管理対象トリプル（managed triples））をグラフから削除します。埋め込みトリプル（embedded triples）を含む XML ドキュメントは影響を受けません。グラフに両方のタイプのドキュメントが含まれている場合は、DELETE 操作の実行後もグラフが存在し続けることがあります。

PUT リクエストを送信すると、トリプルは名前付きグラフ内で置換されます。また、そのグラフが存在しない場合は、空のグラフに追加されます。これは、DELETE の後に POST を実行することと同等です。

例えば以下のようになります。

```
# Windows users, see Modifying the Example Commands for Windows

$ curl --digest --user admin:password -s -X PUT
-H "Content-type:text/turtle" --data-binary '@./example.ttl'
    "http://localhost:8033/v1/graphs?graph=mynamed-graph"
```

REST API を使用して DELETE 操作と同等の処理を実行するには、curl を使用し、空のグラフを指定して PUT リクエストを送信します。

13.0 ドキュメント内のトリプルを識別するテンプレートを使用する

インデックス付けするデータを既存ドキュメント内のトリプルとして識別するためのテンプレートを定義できます。トリプルとして表現するあらゆるタイプのデータを含むドキュメントは、テンプレートを使用してインデックス付けできます。テンプレートによって識別されるトリプルは、unmanaged triples と同様、組み込みトリプルと呼ばれることもあります。

このようなトリプルをインデックス付けしたら、非管理対象トリプルとまったく同じ方法でクエリできます。つまり、SPARQL、`xdmp.sparql()`、複合クエリ、新しいオプティック API、および `cts:triple-range-query` を使用できます。トリプルの操作の詳細については、「非管理対象トリプル」（64 ページ）を参照してください。テンプレートの作成および使用の詳細については、『*Application Developer's Guide*』の「Template Driven Extraction (TDE)」を参照してください。

この章では、次の内容を取り上げます。

- テンプレートの作成
- テンプレートの要素
- 例
- TDE と SQL で生成されるトリプル

13.1 テンプレートの作成

次に、トリプルを識別するシンプルなテンプレートの例を示します。これには、テンプレートの名前空間とコンテキストの定義が含まれます。トリプルの主語、目的語、述語の記述、および値のデータマッピングも含まれています。

```
<template xmlns="http://marklogic.com/xdmp/tde">
  <context>/article/topic</context>
    <vars>
      <var>
        <name>EX</name>
        <val>"http://example.org/ex#"</val>
      </var>
    </vars>
    <triples>
     <triple>
      <subject>
        <val>sem:iri( $EX || "who" )</val>
      </subject>
      <predicate>
        <val>sem:iri( $EX || "what" )</val>
      </predicate>
      <object>
```

```
        <val>xs:string( $EX || "where" )</val>
      </object>
    </triple>
  </triples>
</template>
```

トリプルについては、主語と述語の記述に値 `sem:iri` を設定する必要があります。ここでは、テンプレートは、IRI を指定するときの入力量を少なくするために省略形として `vars` を採用しています。トリプルを識別するテンプレートを作成するときは、XQuery 言語の式のサブセットを使用して抽出する値のタイプを指定できます。詳細については、『*Application Developer's Guide*』の「Template Dialect and Data Transformation Functions」を参照してください。

注： テンプレートを使用して識別されたトリプルは、直接修正したり、（SPARQL Update を使用するなどして）トリプルとして修正したりすることはできません。できるのは、トリプルが存在しなくなるようにテンプレートを無効にして削除することや、基になるドキュメントデータを修正してトリプルを修正することです。

テンプレートのセキュリティは、protected コレクションを設定することで制御できます。『*Application Developer's Guide*』の「Security on TDE Documents」を参照してください。

13.2 テンプレートの要素

テンプレートには、次の要素とその子要素が含まれています。

要素	説明
`context`	テンプレートのアクティブ化およびデータの抽出に使用される参照ノードです。詳細については、『*Application Developer's Guide*』の「Context」を参照してください。
`description`	テンプレートの説明です（オプション）。
`collections` `collection` `collections-and` `collection`	コレクションスコープです（オプション）。複数のコレクションスコープを OR または AND 演算できます。 `<collections>` セクションは、以下のシーケンスの、トップレベルの OR です。 テンプレートを特定のコレクションにスコープ指定する `<collection>`。 一体として AND 演算される `<collection>` シーケンスを含んでいる `<collections-and>`。 詳細については、『*Application Developer's Guide*』の「Collections」を参照してください。

要素	説明
directories directory	ディレクトリスコープです（オプション）。複数のディレクトリスコープは、OR 演算されます。
vars var	現在のコンテキストレベルで抽出される中間変数です（オプション）。 この要素は、トリプル内で IRI（プレフィックス）の省略形として使用できます。詳細については、『*Application Developer's Guide*』の「Variables」を参照してください。
triples triple subject val invalid-values predicate val invalid-values object val invalid-values	これらの要素は、トリプル抽出テンプレートに使用します。 triples は、トリプルの抽出記述のシーケンスを格納します。各 triple 記述は、subject、predicate、object のデータマッピングを定義します。 テンプレートを通じて、抽出されたトリプルのグラフを指定することはできません。グラフは、組み込みトリプルと同様、ドキュメントのコレクションによって暗黙で定義されます。
templates template	サブテンプレートのシーケンスです（オプション）。詳細については、『*SQL Data Modeling Guide*』の「Creating Views from Multiple Templates」および「Creating Views from Nested Templates」を参照してください。
path-namespaces path-namespace	名前空間バインドのシーケンスです（オプション）。詳細については、『*Application Developer's Guide*』の「path-namespaces」を参照してください。
enabled	ブール型です。テンプレートをオン（true）にするかオフにするか（false）を指定します。デフォルト値は true です。

context、vars、triples は、パス式を使用して XQuery 要素や JSON プロパティを識別します。var 要素を使用すると、トリプル内の要素のプレフィックスを指定できます。

例えば以下のようになります。

```
<vars>
 <var>
   <name>ex</name>
```

```
    <val>"http://example.org/ex#"</val>
  </var>
</vars>
```

パス式は XPath に基づいています。XPath については、『*XQuery and XSLT Reference Guide*』の「XPath Quick Reference」および『*Application Developer's Guide*』の「Traversing JSON Documents Using XPath」で説明しています。

13.2.1 テンプレートによってトリガーされる再インデックス付け

トリプルテンプレートを追加または修正すると、再インデックス付けがトリガーされます。テンプレートによって抽出されたトリプルは、トリプルインデックスに出現し始めたらすぐに使用できるようになります。再インデックス付けが行われるのは、コンテキスト要素、ディレクトリスコープおよびコレクションスコープがマッチしているドキュメントのみです。したがって、不要な（再）インデックス付け動作を避けるには、これらを慎重に選択してください。

- 新しいテンプレートの場合、ドキュメントがインデックス付けされたときにトリプルがインデックスに出現します。
- 修正されたテンプレートの場合（および再インデックス付けが完了するまでは）、テンプレートの前バージョンで抽出された既存トリプル（まだ再インデックス付けされていないドキュメントのもの）と、テンプレートの新しいバージョンによって抽出された新しいトリプル（再インデックス付けされたドキュメントのもの）が混在する場合があります。

13.3 例

ドキュメント内のトリプルを識別するテンプレートを検証して使用する方法は、いくつかあります。このセクションでは、それらの例を記載します。

- テンプレートの検証と挿入
- 検証と挿入をワンステップで行う
- JSON テンプレートの使用
- 可能性のあるトリプルの識別

13.3.1 テンプレートの検証と挿入

この例では、Query Console を使用して次のドキュメントを Documents データベースに挿入してください。このドキュメントは、トリプルのソースとして使用されます。

```
xdmp:document-insert("APNews.xml",
<article>
 <info>APNewswire - Nixon went to China</info>
 <triples-context>
  <confidence>80</confidence>
```

```
  <published>2011-10-14</published>
  <source>AP News</source>
 </triples-context>
 <topic>
  <who>Nixon</who>
  <what>wentTo</what>
  <where>China</where>
  </topic>
 <body>
  In 1974, Richard Nixon went to China.
 </body>
</article>
)
```

Query Console を使用して、次のテンプレート（APtemplate.xml）を検証してから、Schemas データベース内の http://marklogic.com/xdmp/tde というコレクションに挿入します。まず、次のテンプレートを検証します。

```
let $t1 :=
<template xmlns="http://marklogic.com/xdmp/tde">
  <context>/article/topic</context>
    <vars>
      <var>
        <name>EX</name>
        <val>"http://example.org/ex#"</val>
      </var>
    </vars>
  <triples>
    <triple>
      <subject>
        <val>sem:iri( $EX || "who")</val>
      </subject>
      <predicate>
        <val>sem:iri( $EX || "what")</val>
      </predicate>
      <object>
        <val>xs:string( $EX || "where")</val>
      </object>
    </triple>
  </triples>
  </template>

return tde:validate($t1)
=>
<map:map xmlns:map="http://marklogic.com/xdmp/map"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
  <map:entry key="valid">
    <map:value xsi:type="xs:boolean">true
    </map:value>
  </map:entry>
</map:map>
```

次に、有効なテンプレートを挿入します。tde:template-insert を使用してください。これにより、テンプレートの Schemas データベースへの挿入と適切なコレクションへの挿入が処理されます。

```
xquery version "1.0-ml";
import module namespace tde = "http://marklogic.com/xdmp/tde"
  at "/MarkLogic/tde.xqy";

let $t1 :=
<template xmlns="http://marklogic.com/xdmp/tde">
  <context>/article/topic</context>
    <vars>
      <var>
        <name>EX</name>
        <val>"http://example.org/ex#"</val>
      </var>
    </vars>
  <triples>
    <triple>
      <subject>
        <val>sem:iri( $EX || "who")</val>
      </subject>
      <predicate>
        <val>sem:iri( $EX || "what")</val>
      </predicate>
      <object>
        <val>xs:string( $EX || "where")</val>
      </object>
    </triple>
  </triples>
  </template>

return tde:template-insert(
"APtemplate.xml",$t1, (), "http://marklogic.com/xdmp/tde")
```

このテンプレートを使用すると、ドキュメント内のコンテンツはトリプルとしてインデックス付けされます。このトリプルは、オリジナルのドキュメントには追加されません。トリプルを確認するには、Query Console で次のクエリを実行します。

```
tde:node-data-extract(fn:doc("APNews.xml"));
```

このクエリは、ドキュメント名と、トリプルとしてインデックス付けされたコンテンツを返します。

```
=>
{"APNews.xml": [
 {
  "triple": {
   "subject": "http://example.org/ex#Nixon",
   "predicate": "http://example.org/ex#wentTo",
   "object": {
    "datatype": "http://www.w3.org/2001/XMLSchema#string",
    "value": "http://example.org/ex#China"
   }
  }
 }
]}
```

次の SPARQL クエリを使用して、トリプルがトリプルインデックス内にあることを確認します。

```
SELECT ?country
WHERE {
  <http://example.org/news/Nixon>
<http://example.org/wentTo>
  ?country
}

=> China
```

13.3.2 検証と挿入をワンステップで行う

次の例は、`tde:template-insert` を使用して、テンプレートの検証と、このコンテンツデータベースに関連付けられている Schemas データベースへのテンプレートの挿入の両方をワンステップで行っています。この例では、「非管理対象トリプル」（64 ページ）で説明しているドキュメントを挿入します。

ドキュメントを Documents データベースに挿入します（「SAR」コレクション内）。

```
xquery version "1.0-ml";
xdmp:document-insert("SAR_report.xml",
<SAR>
 <title>Suspicious vehicle...Suspicious vehicle near
airport</title>
  <date>2015-11-12Z</date>
  <type>observation/surveillance</type>
  <threat>
```

```
   <type>suspicious activity</type>
    <category>suspicious vehicle</category>
  </threat>
  <location>
    <lat>37.497075</lat>
    <long>-122.363319</long>
  </location>
  <description>A blue van with license plate ABC 123 was
observed parked behind the airport sign...
    <sem:triple>
      <sem:subject>IRIID</sem:subject>
      <sem:predicate>isa</sem:predicate>
       <sem:object
datatype="http://www.w3.org/2001/XMLSchema#string">license
-plate</sem:object>
    </sem:triple>
    <sem:triple>
      <sem:subject>IRIID</sem:subject>
      <sem:predicate>value</sem:predicate>
      <sem:object
datatype="http://www.w3.org/2001/XMLSchema#string">ABC
123</sem:object>
    </sem:triple>
  </description>
</SAR>,(),
"SAR")
```

このドキュメントには、すでに 2 つの組み込みトリプルがあります。では、レポートに記述される脅威の日付とタイプを記述している他のトリプルを識別してみましょう。トリプルを識別するテンプレートを作成し、`tde:template-insert` を使用してそのテンプレートを挿入します。`tde:template-insert` は、テンプレートを検証してから、Schemas データベースに挿入します。

```
xquery version "1.0-ml";
import module namespace tde = "http://marklogic.com/xdmp/tde"
  at "/MarkLogic/tde.xqy";

let $template :=
<template xmlns="http://marklogic.com/xdmp/tde">
  <context>/SAR</context>
    <triples>
    <triple>
      <subject>
        <val>sem:iri(threat/type)</val>
      </subject>
```

```
      <predicate>
        <val>sem:iri("http://example.org/on-date")</val>
      </predicate>
      <object>
        <val>xs:date(date)</val>
      </object>
    </triple>
    </triples>
</template>
return tde:template-insert("SARtemplate.xml", $template)
```

新しいトリプルを確認するには、Query Console で `tde:node-data-extract` を使用して次のクエリを実行します。

```
tde:node-data-extract(fn:doc("SAR_report.xml"));

=>
{
 "SAR_report.xml": [
  {
   "triple": {
    "subject": "suspicious activity",
    "predicate": "http://example.org/on-date",
    "object": {
      "datatype": "http://www.w3.org/2001/XMLSchema#date",
      "value": "2014-11-12Z"
    }
   }
  }
 ]
}
```

このドキュメント内のすべてのトリプルを確認するには、Query Console で、「SAR」コレクションに限定されている次の SPARQL クエリを実行します。

```
SELECT *
FROM <SAR>
WHERE {
  ?s ?p ?o
}
```

これにより、次のように SAR_report.xml ドキュメント内のトリプルがすべて返されます。

```
s                          p                                  o
<suspicious activity>   <http://example.org/on-date>  2014-11-
12Z"^^xs:date
<IRIID>                  <isa>                  <license-plate>
<IRIID>                   <value>               <ABC 123>
```

13.3.3　JSON テンプレートの使用

JSON テンプレートを使用して、JSON ドキュメント内のトリプルを識別できます。

注： いずれのテンプレート（XML または JSON）も、任意のドキュメント（XML または JSON）からトリプルを抽出します。

次のようにして、このドキュメントを Documents データベースに挿入します。

```
declareUpdate();
xdmp.documentInsert("/medlineCitation.json", ({
   "MedlineCitation": {
    "Status": "Completed",
    "MedlineID": 69152893,
    "PMID": 5717905,
    "Article": {
      "Journal": {
        "ISSN": "0043-5341"
      },
      "ArticleTitle": "[On the influence of calcium ... on
cholesterol in human serum]",
      "AuthorList": {
        "Author": [
          {
            "LastName": "Doe",
            "ForeName": "John"
          },
          {
            "LastName": "Smith",
            "ForeName": "Jane"
          }
        ]
      }
    }, "collections" : "http://marklogic.com/xdmp/tde"}
}));
```

次に、JSON テンプレートを検証し、挿入します。tde.templateInsert コマンドは、テンプレートを検証し、Schemas データベースに挿入します。

```
declareUpdate();
var tde = require ("/MarkLogic/tde.xqy");

var template = xdmp.toJSON({
  "template":{
    "context":"/MedlineCitation/Article",
    "vars":[
      {
        "name":"prefix1",
        "val":"\"http://marklogic.com/example/\""
      }
    ],
    "triples":[{
        "subject":{
        "val":"sem:iri($prefix1||'person/'||
        AuthorList/Author[1] \
          /ForeName||'_'||AuthorList/Author[1]/LastName)"},
        "predicate":{
          "val":"sem:iri(($prefix1||'authored'))"},
        "object":{
          "val":"xs:string(Journal/ISSN)"}
         }
        ] }});

tde.templateInsert("medlineTemplate.json", template);

// After validating the template, this inserts template
into the Schemas
database as medlineTemplate.json
```

Query Console で、次のクエリを Documents データベースに対して実行します。このクエリは、トリプルの形式になっている、ドキュメント内の 1 番目の著者を識別します。

```
tde.nodeDataExtract([fn.doc("/medlineCitation.json")]);
=>
{
 "/medlineCitation.json": [
  {
  "triple": {
    "subject": "http://marklogic.com/example/person/John_Doe",
     "predicate": "http://marklogic.com/example/authored",
```

```
      "object": {
       "datatype": "http://www.w3.org/2001/XMLSchema#string",
        "value": "0043-5341"
      }
     }
    }
   ]
  }
```

注： `nodeDataExtract` コマンドは、テンプレートビューの外観を示すヘルパーユーティリティです。通常は、生成されたビューに対して SQL または SPARQL クエリを実行します。

このテンプレートでは、1 番目の著者の名前と ISSN 番号のみを抽出しています。テンプレートにある `[1]` を `[2]` に変更すると、2 番目の著者の名前を抽出できます。

13.3.4 可能性のあるトリプルの識別

次の例には、ドキュメントとテンプレートの両方が含まれます。テンプレートは、Query Console に貼り付けることができる 1 つのクエリの一部として 2 つのトリプルを識別するために使用されています。`tde:node-data-extract` は、このドキュメントとテンプレートを挿入した場合にインデックス付けされるであろうトリプルを示すヘルプ関数です。

```
let $doc1 :=
<MedlineCitation Status="Completed">
  <MedlineID>69152893</MedlineID>
  <PMID>5717905</PMID>
  <Article>
    <Journal>
      <ISSN>0043-5341</ISSN>
      <JournalIssue>
        <Volume>118</Volume>
        <Issue>49</Issue>
        <PubDate>
          <Year>1968</Year>
          <Month>Dec</Month>
          <Day>7</Day>
        </PubDate>
      </JournalIssue>
    </Journal>
    <ArticleTitle>[On the influence of calcium ... on
cholesterol in human serum]</ArticleTitle>
    <AuthorList>
      <Author>
```

```
        <LastName>Doe</LastName>
        <ForeName>John</ForeName>
      </Author>
      <Author>
        <LastName>Smith</LastName>
        <ForeName>Jane</ForeName>
      </Author>
    </AuthorList>
  </Article>
</MedlineCitation>

let $template1 :=
<template xmlns="http://marklogic.com/xdmp/tde">
<context>/MedlineCitation/Article/AuthorList/Author</context>
  <triples>
    <triple>
      <subject>
        <val>sem:iri(concat(ForeName,' ',LastName))</val>
      </subject>
      <predicate>
        <val>sem:iri('authored')</val>
      </predicate>
      <object>
        <val>xs:string(../../ArticleTitle)</val>
      </object>
    </triple>
  </triples>
</template>

return tde:node-data-extract (($doc1), ($template1))
```

このクエリは、次のように、トリプルインデックスに追加されるであろう 2 つのトリプルを JSON 形式で返します。

```
{
 "document1": [
   {
   "triple": {
   "subject": "John Doe",
   "predicate": "authored",
   "object": {
     "value": "[On the influence of calcium ... on
cholesterol in human serum]"
    }
   }
  },
```

```
    {
     "triple": {
     "subject": "Jane Smith",
     "predicate": "authored",
     "object": {
       "datatype": "http://www.w3.org/2001/XMLSchema#string",
        "value": "[On the influence of calcium ... on
cholesterol in human serum]"
      }
     }
    }
  ]
}
```

この例の、これらのトリプルはトリプルインデックスに追加されていませんが、テンプレートがどのように動作するのか、およびドキュメントとテンプレートを挿入した場合にどのトリプルがインデックス付けされるのかを確認できます。

注： テンプレートを通じて、これらのトリプルのグラフを指定することはできません。グラフは、組み込みトリプルと同様、ドキュメントのコレクションによって暗黙で定義されます。

13.4 TDE と SQL で生成されるトリプル

SQL 用に作成された TDE ビューの中には、インデックスエントリを生成するものがあります。このインデックスエントリは、トリプルインデックスを使用した SQL の実装を基にしているため、トリプルとして存在し、確認したり、使用したりできます。このトリプルは、SPARQL クエリの結果に出現することがあります。

このトリプルは、非常に独特な主語と述語の URI を持っています。したがって、SPARQL クエリに何らかの主語または述語フィルタが含まれる限り、行テンプレートによって生成されたトリプルは結果に出現しません。

行テンプレートから生成されたトリプルの例を示します。

```
<http://marklogic.com/row/09CA32CBA69361E5/8FD41B78E884B48E>
  <http://marklogic.com/column/id/81C579F95CEA957B>
  "George Washington"
```

このような行トリプルが出現する可能性がある SPARQL 操作としては、次のような操作があります。

1. 「すべてのトリプルを表示してください」ということを表す SPARQL クエリ。初めて SPARQL を試すときは、トリプルを 10 個読み込み、次の SPARQL クエリを実行できます。

```
SELECT *
WHERE {
   ?s ?p ?o }
```

注： このクエリはデータベース内のすべてのトリプルを返すため、パフォーマンス上の理由から、大量のトリプルがあるデータベースではこのクエリを実行しないでください。

2. 「すべてのトリプルをカウント」する SPARQL クエリ。上記のクエリと同様、データベース内のすべてのトリプルにアクセスします。

3. 「他とは異なる述語をすべて表示」する SPARQL クエリ。これも、よくあるトリプルデータの探索方法です。

このようなクエリの一部として返される行トリプルを表示しないようにするには、名前付きグラフからトリプルを挿入およびクエリするか、主語または述語フィルタを含めて行トリプルを除外します。

注： トリプルを名前付きグラフに挿入して、そのグラフからクエリするのが最適な方法です。

トリプルでサーバーサイドクエリにオプティック API を使用する方法の詳細については、「オプティック API を使用したトリプルのクエリ」（144 ページ）、`op:from-triples` 関数または `op.fromTriples` 関数、『*Application Developer's Guide*』の「Data Access Functions」および「Optic API for Relational Operations」を参照してください。クライアントサイドクエリにオプティック API を使用する方法については、「オプティック API を使用したクエリ」（266 ページ）および『*Java Application Developer's Guide*』の「Optic Java API for Relational Operations」を参照してください。また、Client API Reference の「/REST/client/row-management」、『*REST Application Developer's Guide*』の行マネージャおよび行エンドポイントに関する記述も参照してください。

SQL コンテンツでのテンプレートの使用については、『*SQL Data Modeling Guide*』の「Creating Template Views」を参照してください。

14.0 実行プラン

このセクションでは、sem:sparql-plan 関数からのクエリ実行プランの出力を解釈する方法について説明します。生成されたクエリ実行プランは、指定されたクエリが SPARQL パーサーによってどのように処理されるのかを示しています。SPARQL のクエリ実行プランは、SELECT、CONSTRUCT、ASK、DESCRIBE など、すべての SPARQL クエリと連携して動作するように設計されています。

14.1 実行プランの生成

sem:sparql-plan を使用して SPARQL クエリのクエリ実行プランを生成すると、内部でクエリが処理される方法を確認できます。

例えば、次の SPARQL クエリにより、実行プランが生成されます。

```
sem:sparql-plan("select * { ?s ?p ?o }",(),"optimize=1")
```

このクエリは、次の実行プランを出力します。

```
<plan:plan xmlns:plan="http://marklogic.com/plan">
 <plan:select>
  <plan:project order="1,0,2">
   <plan:variable name="s" column-index="0"
   static-type="NONE">
   </plan:variable>
   <plan:variable name="p" column-index="1"
   static-type="NONE">
   </plan:variable>
   <plan:variable name="o" column-index="2"
   static-type="NONE">
   </plan:variable>
   <plan:triple-index order="1,0,2" permutation="PSO"
   dedup="true">
    <plan:subject>
      <plan:variable name="s" column-index="0"
      static-type="NONE">
     </plan:variable>
    </plan:subject>
    <plan:predicate>
     <plan:variable name="p" column-index="1"
     static-type="NONE">
     </plan:variable>
    </plan:predicate>
    <plan:object>
     <plan:variable name="o" column-index="2"
     static-type="NONE">
     </plan:variable>
```

```
     </plan:object>
    </plan:triple-index>
   </plan:project>
  </plan:select>
</plan:plan>
```

14.2 実行プランのパース

このセクションでは、実行プランを各部に分けて説明します。

導入部分は、プランのタイプを指定します。この場合、SELECT ステートメントに対するものです。

```
<plan:plan xmlns:plan="http://marklogic.com/plan">
 <plan:select>
```

次のセクションは、トリプルの要素（主語、述語、目的語）の射影順序を変数名（s、p、o）とカラムインデックスで識別します。

```
<plan:project order="1,0,2">
   <plan:variable name="s" column-index="0" static-
type="NONE">
   </plan:variable>
   <plan:variable name="p" column-index="1" static-
type="NONE">
   </plan:variable>
   <plan:variable name="o" column-index="2" static-
type="NONE">
   </plan:variable>
```

最後の部分は、トリプルインデックス内のトリプル変数の順序（述語、主語、目的語 - p、s、o）になります。

```
   <plan:triple-index order="1,0,2" permutation="PSO"
dedup="true">
    <plan:subject>
     <plan:variable name="s" column-index="0" static-type="NONE">
     </plan:variable>
    </plan:subject>
    <plan:predicate>
     <plan:variable name="p" column-index="1" static-type="NONE">
     </plan:variable>
    </plan:predicate>
    <plan:object>
     <plan:variable name="o" column-index="2" static-type="NONE">
     </plan:variable>
    </plan:object>
```

```
   </plan:triple-index>
  </plan:project>
 </plan:select>
</plan:plan>
```

このシンプルなクエリを実行すると、変数と値が 3 つのカラム（s、p、o）に射影されます。

```
[{"s":"<http://example.com/ns/directory#jp>","p":"<http://
example.com/ns/person#firstName>","o":"\"John-Paul\""}]
```

次に、もう少し複雑な SPARQL の SELECT クエリで、プレフィックスを含むものの例を示します。

```
sem:sparql-plan("
PREFIX rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:   <http://www.w3.org/2000/01/rdf-schema#>
PREFIX prod:   <http://example.com/products/>
PREFIX ex:     <http://example.com/>

SELECT ?product
FROM <http://marklogic.com/semantics/products/>
WHERE
  {
    ?product  rdf:type  ex:Shirt ;
              ex:color  'blue' }")
```

次に、このクエリ実行プランの出力を示します。導入部分は、名前空間と、プランのタイプを指定しています。

```
<plan:plan xmlns:plan="http://marklogic.com/plan">
 <plan:select>
```

1 番目のセクションは、射影される値の順序を識別します。この場合、「product」が 1 つだけです。

```
  <plan:project order="order(0 ASC)">
   <plan:variable name="product" column-index="0" static-
type="NONE">
    </plan:variable>
```

このセクションは、ハッシュの結合順序の種類を記述しています。

```
   <plan:hash-join order="order(0 ASC)">
    <plan:hash left="0" right="0" operator="=">
     </plan:hash>
```

次は、トリプル要素（目的語、述語、主語 - o、p、s）の射影順序です。1 番目は、タイプ「Shirt」の製品のトリプルに対するものです。

```
     <plan:triple-index order="order(0 ASC)"
permutation="OPS" dedup="true">
      <plan:subject>
       <plan:variable name="product" column-index="0"
static-type="NONE">
        </plan:variable>
       </plan:subject>
      <plan:predicate>
       <plan:iri name="http://www.w3.org/1999/02/22-rdf-
syntax-ns#type" static-type="NONE">
        </plan:iri>
       </plan:predicate>
      <plan:object>
        <plan:iri name="http://example.com/Shirt" static-
type="NONE">
         </plan:iri>
       </plan:object>
      </plan:triple-index>
```

次は、色の値が「blue」の製品のトリプルに対するものです。

```
     <plan:triple-index order="order(0 ASC)"
permutation="OPS" dedup="true">
      <plan:subject>
       <plan:variable name="product" column-index="0"
static-type="NONE">
        </plan:variable>
       </plan:subject>
      <plan:predicate>
        <plan:iri name="http://example.com/color" static-
type="NONE">
         </plan:iri>
       </plan:predicate>
      <plan:object>
       <plan:value
datatype="http://www.w3.org/2001/XMLSchema#string"
value="blue">
        </plan:value>
       </plan:object>
     </plan:triple-index>
```

そして、プランの結句です。

```
          </plan:hash-join>
        </plan:project>
      </plan:select>
    </plan:plan>
```

クエリ実行プランの詳細については、『*SQL Data Modeling Guide*』の「Execution Plan」を参照してください。