MarkLogic サーバー

Semantics Developer's Guide

MarkLogic 8
2015 年 2 月

最終更新：8.0-1、2015 年 2 月

Copyright © 2015 MarkLogic Corporation.All rights reserved.

目次

Semantics Developer's Guide

1.0 MarkLogic におけるセマンティックの概要 7
1.1 用語 8
1.2 リンクトオープンデータ 10
1.3 MarkLogic における RDF 実装 10
1.3.1 MarkLogic での RDF の使用 11
1.3.1.1 MarkLogic での RDF トリプルの格納 12
1.3.1.2 トリプルのクエリ 14
1.3.2 RDF データモデル 16
1.3.3 空白ノードの識別子 17
1.3.4 RDF データ型 18
1.3.5 IRI とプレフィックス 18
1.3.5.1 IRI 18
1.3.5.2 プレフィックス 19
1.3.6 RDF 語彙 21
1.4 データセットの例 22

2.0 MarkLogic におけるセマンティックのクイックスタート 23
2.1 MarkLogic サーバーの設定 23
2.1.1 トリプルを使用するためのデータベースの設定 23
2.1.2 追加サーバーの設定 24
2.2 トリプルのロード 25
2.2.1 データセットのダウンロード 25
2.2.2 mlcp を使用したトリプルのインポート 25
2.2.3 インポートの検証 27
2.3 トリプルのクエリ 29
2.3.1 ネイティブ SPARQL によるクエリ 29
2.3.2 sem:sparql 関数によるクエリ 32

3.0 トリプルのロード 34
3.1 埋め込み RDF トリプルのロード 34
3.2 トリプルのロード 34
3.2.1 サポートされている RDF トリプルのシリアライゼーション 36
3.2.2 RDF シリアライゼーションのシンタックスの例 37
3.2.2.1 RDF/XML 37
3.2.2.2 Turtle 38
3.2.2.3 RDF/JSON 38
3.2.2.4 N3 39
3.2.2.5 N-Triples 40

3.2.2.6 N-Quads 40
3.2.2.7 TriG 41
3.2.3 mlcp を使用したトリプルのロード 42
3.2.3.1 準備 42
3.2.3.2 import コマンドのシンタックス 43
3.2.3.3 トリプルおよびクアッドのロード 44
3.2.3.4 インポートオプション 45
3.2.3.5 コレクションおよびディレクトリの指定 46
3.2.4 XQuery を使用したトリプルのロード 49
3.2.4.1 sem:rdf-insert 49
3.2.4.2 sem:rdf-load 50
3.2.4.3 sem:rdf-get 51
3.2.5 REST API を使用したトリプルのロード 52
3.2.5.1 準備 52
3.2.5.2 グラフストアの処理 53
3.2.5.3 パラメータの指定 53
3.2.5.4 サポートされている動詞 54
3.2.5.5 サポートされているメディア形式 54
3.2.5.6 トリプルのロード 54
3.2.5.7 レスポンスエラー 56

4.0 トリプルインデックスの概要 58
4.1 トリプルインデックス：概要と使い方 58
4.1.1 トリプルデータと値のキャッシュ 58
4.1.2 トリプル値とタイプ情報 59
4.1.3 トリプルポジション 59
4.1.4 インデックスファイル 60
4.1.5 順列 61
4.2 トリプルインデックスの有効化 62
4.2.1 データベース設定ページの使用 62
4.2.2 Admin API の使用 63
4.3 その他の考慮事項 64
4.3.1 キャッシュのサイズ設定 64
4.3.2 未使用の値とタイプ 66
4.3.3 スケーリングとモニタリング 67

5.0 埋め込みトリプル 68
5.1 XML ドキュメントでのトリプルの使用方法 70
5.1.1 ドキュメントからのコンテキスト 70
5.1.2 複合クエリ 70
5.1.3 セキュリティ 71
5.2 バイテンポラル埋め込みトリプル 72

6.0 セマンティッククエリ 73
6.1 SPARQL を使用したトリプルのクエリ 74

6.1.1 SPARQL クエリのタイプ 74
6.1.2 Query Console での SPARQL クエリの実行 75
6.1.3 クエリ結果オプションの指定 76
6.1.3.1 Auto 形式と Raw 形式 76
6.1.4 結果のレンダリングの選択 78
6.1.5 SPARQL クエリの構築 79
6.1.6 プレフィックス宣言 80
6.1.7 クエリパターン 80
6.1.8 ターゲット RDF グラフ 84
6.1.8.1 FROM キーワード 85
6.1.8.2 FROM NAMED キーワード 86
6.1.8.3 GRAPH キーワード 87
6.1.9 結果節 88
6.1.9.1 SELECT クエリ 88
6.1.9.2 CONSTRUCT クエリ 89
6.1.9.3 DESCRIBE クエリ 90
6.1.9.4 ASK クエリ 91
6.1.10 クエリ節 92
6.1.10.1 OPTIONAL キーワード 92
6.1.10.2 UNION キーワード 93
6.1.10.3 FILTER キーワード 95
6.1.10.4 比較演算子 98
6.1.10.5 BIND キーワード 98
6.1.10.6 VALUES セクション 99
6.1.11 ソリューション修飾子 99
6.1.11.1 DISTINCT キーワード 99
6.1.11.2 LIMIT キーワード 100
6.1.11.3 ORDER BY キーワード 101
6.1.11.4 OFFSET キーワード 102
6.1.11.5 サブクエリ 103
6.1.11.6 射影式 104
6.1.12 プロパティパス式 104
6.1.12.1 列挙プロパティパス 105
6.1.12.2 非列挙プロパティパス 108
6.1.12.3 推論 110
6.1.13 SPARQL 集約 110
6.1.14 SPARQL のリソース 113
6.2 XQuery または JavaScript によるトリプルのクエリ 115
6.2.1 例を実行する準備 115
6.2.2 セマンティック関数を使用した検索 117
6.2.2.1 sem:sparql 117
6.2.2.2 sem:sparql-values 119
6.2.2.3 sem:store 120
6.2.2.4 インメモリでトリプルの検索 120
6.2.3 変数に対するバインディングの使用 121
6.2.4 結果の XML および RDF 表示 123

6.2.5 CURIE を扱う 125
6.2.6 search API を使用したクエリ 129
6.2.6.1 cts:triples 129
6.2.6.2 cts:triple-range-query 130
6.2.6.3 cts:search 130
6.2.6.4 cts:contains 131
6.3 REST API を使用したトリプルのクエリ 132
6.3.1 前提 133
6.3.2 パラメータの指定 133
6.3.3 POST リクエストでの SPARQL クエリの実行 134
6.3.4 GET リクエストでの SPARQL DESCRIBE クエリ 138
6.3.5 サポートされている動詞 139
6.3.6 応答の出力形式 139
6.3.6.1 例：XML として結果を返す 141
6.3.6.2 例：JSON として結果を返す 142
6.3.6.3 例：HTML として結果を返す 143
6.3.6.4 例：XML または JSON としてブール型の値を返す 144
6.3.6.5 サポートされていないシリアライゼーション 145
6.3.7 REST API を使用したトリプルの操作 145
6.4 セキュリティ 149

7.0 推論 150
7.1 自動推論 150
7.1.1 オントロジー 151
7.1.2 ルールセット 152
7.1.2.1 ルールセットの使用 153
7.1.2.2 クエリのルールセットの選択 154
7.1.2.3 データベースのデフォルトルールセットの指定 154
7.1.2.4 デフォルトルールセットの上書き 157
7.1.2.5 ルールセットの作成 159
7.2 推論を実現するその他の方法 161
7.2.1 パスを使用する推論 161
7.2.2 マテリアライズ 162
7.3 REST API による推論の使用 163
7.4 推論で使用する API のまとめ 164
7.4.1 セマンティック API 165
7.4.2 データベースルールセット API 166
7.4.3 管理 API 167

8.0 SPARQL Update 168
8.1 SPARQL Update の使用 169
8.2 SPARQL Update によるグラフの操作 170
8.2.1 CREATE 171
8.2.2 DROP 171
8.2.3 COPY 172

8.2.4 MOVE 173
8.2.5 ADD 174
8.3 グラフレベルのセキュリティ 175
8.4 SPARQL Update によるデータの操作 177
8.4.1 INSERT DATA 178
8.4.2 DELETE DATA 180
8.4.3 DELETE/INSERT WHERE 180
8.4.4 CLEAR 182
8.5 変数のバインディング 182
8.6 Query Console での SPARQL Update の使用 184
8.7 XQuery または JavaScript での SPARQL Update の使用 185
8.8 REST での SPARQL Update の使用 186
8.8.1 REST で SPARQL Update を使用する方法 186
8.8.2 POST を使用した SPARQL Update 187
8.8.3 POST と URL エンコードされたパラメータを使用した SPARQL Update 189

9.0 XQuery および JavaScript のセマンティック API 190
9.1 セマンティック用の XQuery ライブラリモジュール 190
9.1.1 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 190
9.1.2 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 191
9.2 トリプルの生成 191
9.3 コンテンツからのトリプルの抽出 192
9.4 トリプルのパース 195
9.5 データを調べる 198
9.5.1 sem:triple 関数 198
9.5.2 sem:transitive-closure 199

10.0 XQuery によるトリプルの挿入、削除、および修正 202
10.1 トリプルの更新 202
10.2 トリプルの削除 204
10.2.1 XQuery または JavaScript によるトリプルの削除 204
10.2.1.1 sem:graph-delete 205
10.2.1.2 xdmp:node-delete 205
10.2.1.3 xdmp:document-delete 206
10.2.2 REST API を使用したトリプルの削除 207

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) を使用しており、SPARQL クエリ、SPARQL Update、JavaScript、XQuery、もしくは REST を使用して、RDF トリプル（RDF triple）をネイティブに格納、検索、および管理できます。

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。

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。

- 用語
- リンクトオープンデータ
- MarkLogic における RDF 実装
- データセットの例

1.1 用語

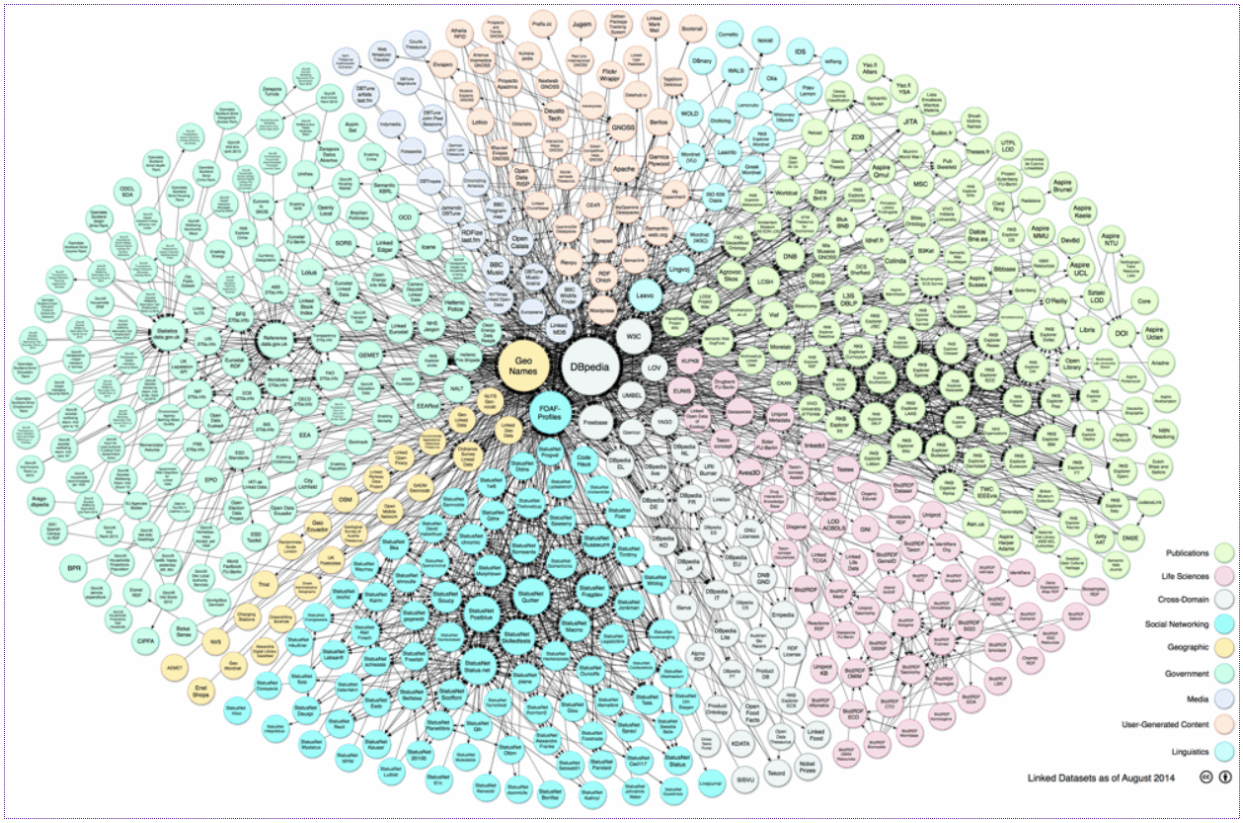
このガイドで使用する用語は、次のとおりです。

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	主語のプロパティまたは特性、あるいは主語と目的語の関係性の表現。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」（18 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

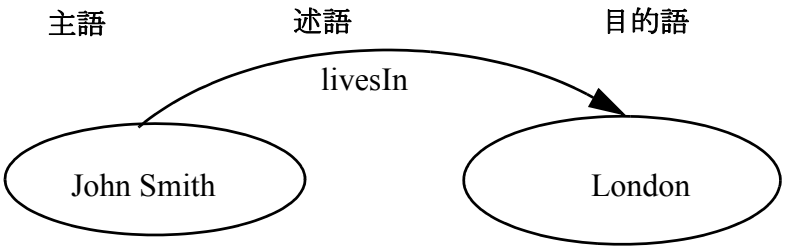
このセクションでは、MarkLogic サーバーに実装されている RDF を使用したセマンティック技術について説明します。このセクションは、次の概念で構成されています。

- 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。

例えば次のようになります。



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。

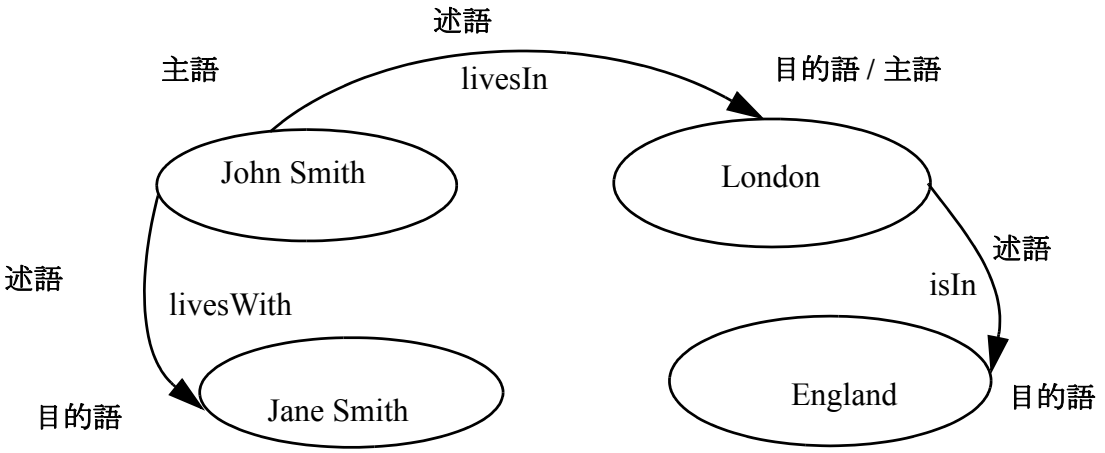
このトリプルを XML で表現すると次のようになります。

```
<sem:triples xmlns:sem="http://marklogic.com/semantics">
  <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>
```

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」（16 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	livesWith	Jane Smith
John Smith	livesIn	London
London	isIn	England

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。例えば次のようになります。

```
<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 シリアライゼーションのシンタックスの例」（37 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。これらは、ドキュメントのルート要素が `sem:triples` であるため、管理対象トリプル（managed triples）です。

```
<?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。その場合は非管理対象トリプル（unmanaged triples）であり、要素ノードは `sem:triples` 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:subject>http://example.org/article</sem:subject>
        <sem:predicate>http://example.org/mentions</sem:predicate>
        <sem:object>http://example.org/London</sem:object>
      </sem:triples>
...
  </info>
</article>
```

ロードしたトリプルは、トリプルインデックス（triple index）と呼ばれる特別なトリプル用インデックスで自動的にインデックス付けされます。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 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」（73 ページ）を参照してください。

XQuery を使用すると、cts:triples または cts:triple-range-query によってトリプル、ドキュメント、および値をクエリできます。

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";

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」（73 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";
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」（115 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。例えば次のようになります。

  ```
  <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。例えば次のようになります。

  ```
  "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。
  例えば次のようになります。

```
"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。例えば次のようになります。

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

IRI の詳細については、「IRI とプレフィックス」（18 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。例えば次のようになります。

```
_: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。例えば次のようになります。

```
<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。例えば次のようになります。

```
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

プレフィックス	プレフィックス IRI	
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。

例えば次のようなものが挙げられます。

- 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 におけるセマンティックのクイックスタート

この章は、次のセクションで構成されています。

- MarkLogic サーバーの設定
- トリプルのロード
- トリプルのクエリ

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、その後、次のタスクを実行します。

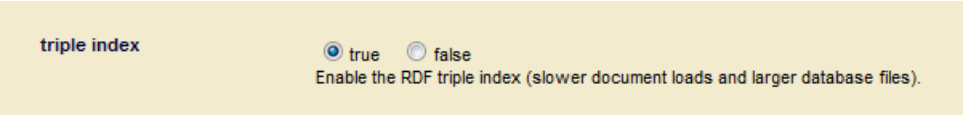
- トリプルを使用するためのデータベースの設定
- 追加サーバーの設定

2.1.1 トリプルを使用するためのデータベースの設定

トリプル用にデフォルトのデータベースを使用するには、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オンにする必要があります。

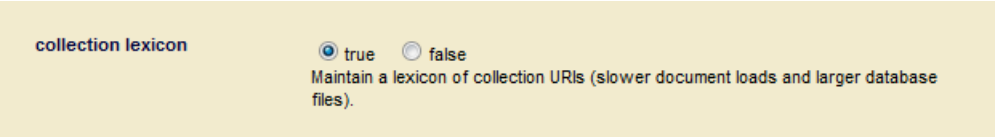
1. トリプル用にデータベースを設定するには、Admin 画面（`localhost:8001`）に移動します。Documents データベースをクリックし、［Configure］タブをクリックします。

2. ［Admin Configure］ページを下にスクロールし、［triple index］を［true］に設定します。



Admin 画面に表示されているトリプルインデックスは、セマンティックインデックスです。

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にロードする前に必要な作業は完了です。

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 MarkLogic REST 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. Persondata のサンプルのデータセットを http://wiki.dbpedia.org/Downloads からダウンロードします。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。ページを下にダウンロードして Persondata を探し、「en」であり「ttl」であるバージョン（`persondata_en.ttl.bz2`）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. 『*Loading Content Into MarkLogic Server Guide*』の「Downloading and Installing mlcp」の説明に従って、MarkLogic Content 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。次に例を示します。

```
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。

   例えば次のようになります。

   ```
   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」（25 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。

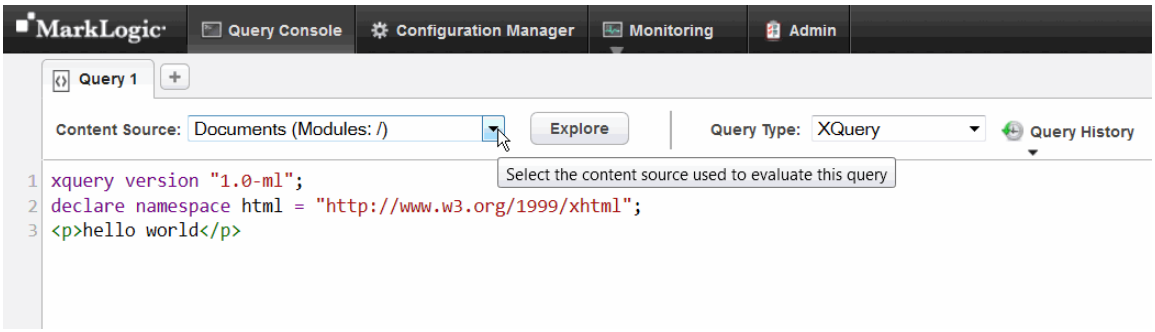
クエリを実行するには、次の手順に従います。

1. Web ブラウザで、Query Console に移動します。

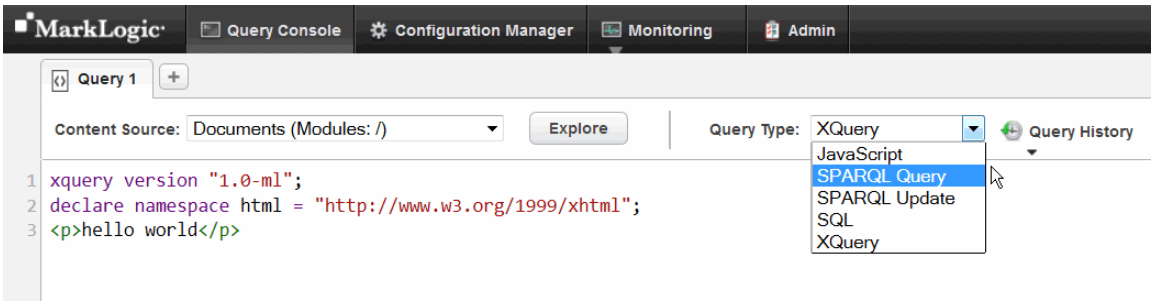
   ```
   http://hostname:8000/qconsole
   ```

   hostname は MarkLogic サーバーの名前です。

2. ［Content Source］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。

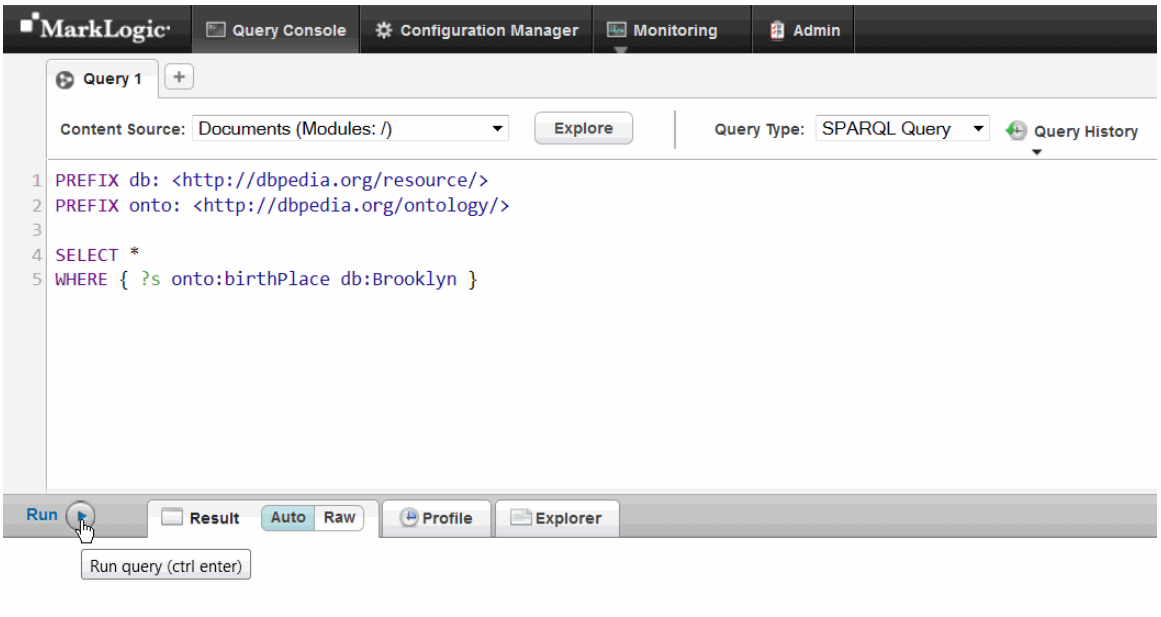


4. クエリウィンドウで、デフォルトのクエリテキストを以下の SPARQL クエリで置換します。

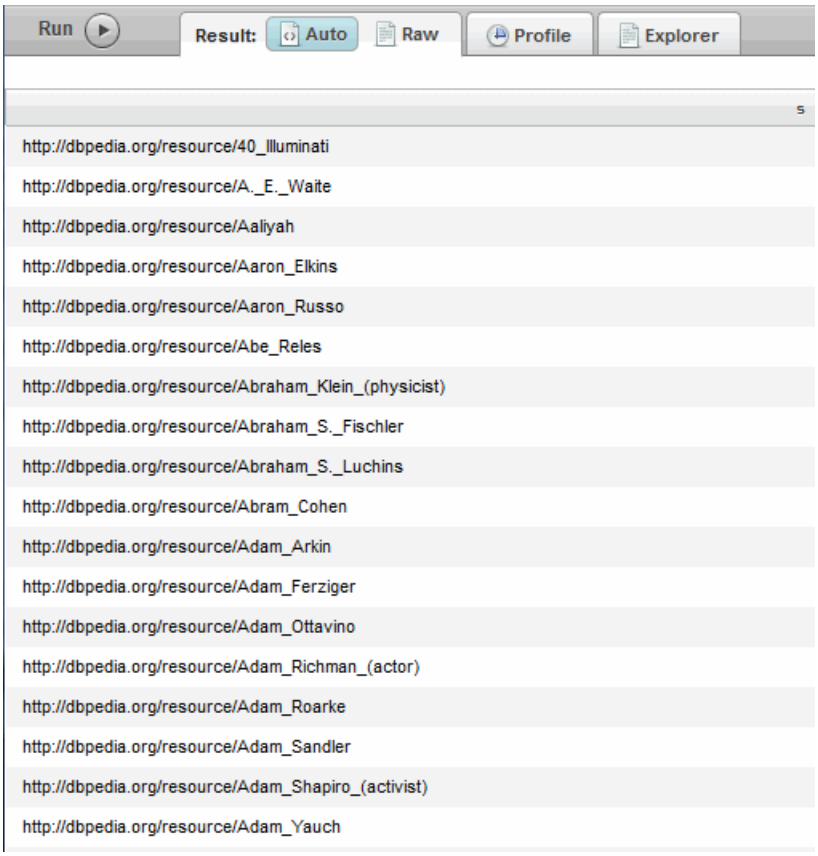
```
PREFIX db: <http://dbpedia.org/resource/>
PREFIX onto: <http://dbpedia.org/ontology/>

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。

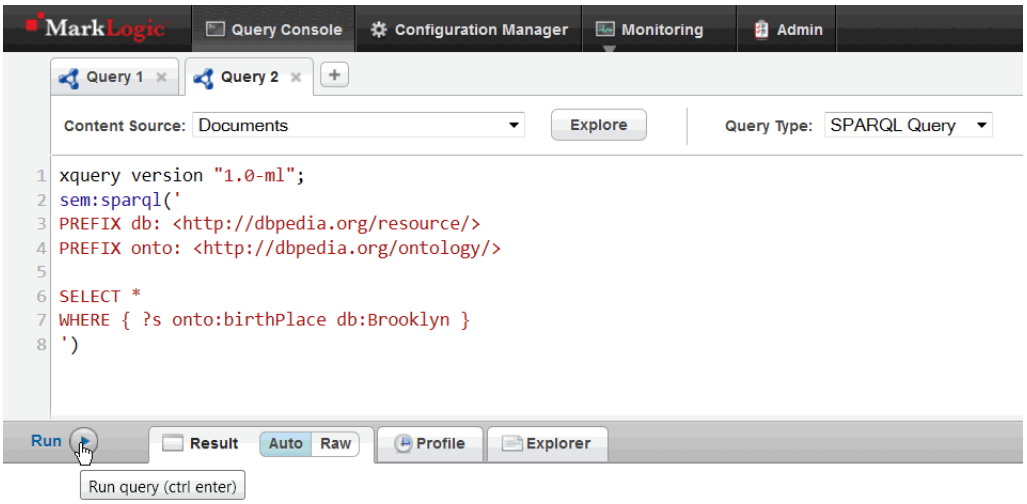
1. ［Content Source］ドロップダウンリストから Documents データベースを選択します。
2. ［Query Type］ドロップダウンリストから、［XQuery］を選択します。
3. クエリウィンドウに、このクエリを入力します。

```
xquery version "1.0-ml";

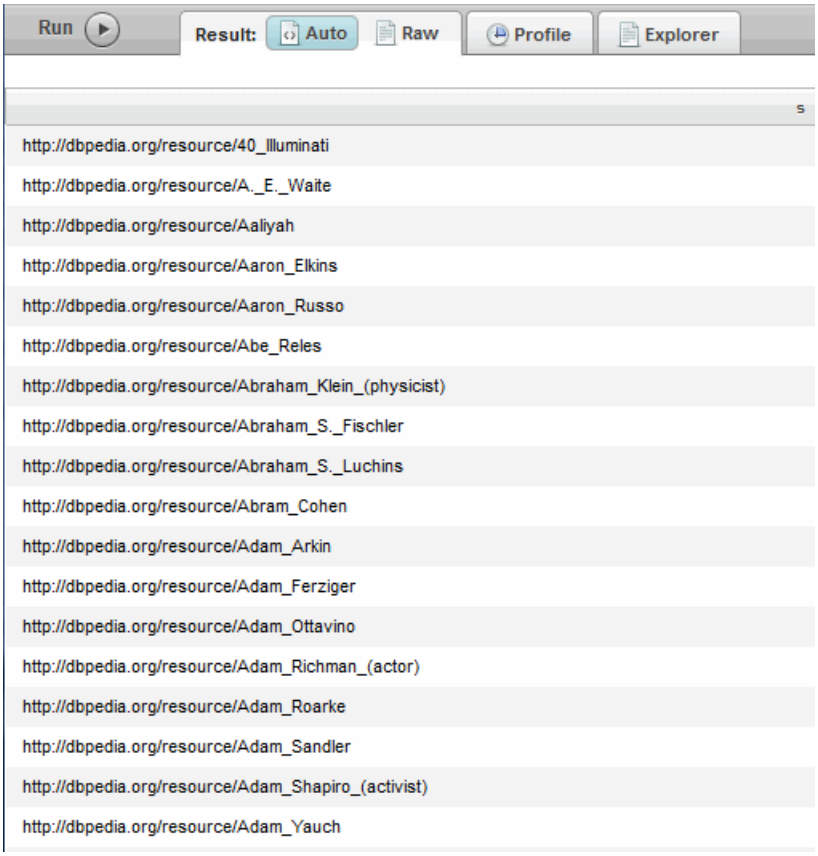
sem:sparql('
PREFIX db: <http://dbpedia.org/resource/>
PREFIX onto: <http://dbpedia.org/ontology/>

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」（29 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」（73 ページ）を参照してください。

3.0 トリプルのロード

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」（168 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 にトリプルが埋め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ロードします。

> **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」（12 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 にロードできます。「サポートされている RDF トリプルのシリアライゼーション」（36 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。

トリプルは、コレクション（collection）と呼ばれる MarkLogic サーバー機能を使用するデフォルトグラフに格納されます。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。

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、コレクション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」（42 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 トリプルのシリアライゼーション
- RDF シリアライゼーションのシンタックスの例
- 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 トリプルのシリアライゼーション

MarkLogic サーバーでは、次に示す RDF データのシリアライゼーション形式をロードできます。

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」（37 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」（38 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」（38 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」（39 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」（40 ページ）を参照してください。	.nt	application/n-triples

形式	説明	ファイル形式	MIME タイプ
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」（40 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」（41 ページ）を参照してください。	.trig	application/trig

3.2.2 RDF シリアライゼーションのシンタックスの例

このセクションでは、次の RDF シリアライゼーション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。

次に例を示します。

```
<http://example.org/item/item22>
<http://example.org/details/shipped>
<http://purl.org/dc/elements/1.1/date> "2013-05-14".
```

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。

次に例を示します。

```
{ "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。

次に例を示します。

```
@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の IRI は、ターム名「acquaintanceOf」を語彙 IRI に付加することで構築されます。

```
<http://example.org/#Jane>
<http://http://purl.org/vocab.org/relationship/acquaintanceOf>
<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/x-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「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 のインストールおよび使用とシステム要件の詳細については、『*Loading Content Into MarkLogic Server Guide*』の「Downloading and Installing mlcp」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. Admin 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、トリプルインデックスをオンにします（さらに、オプションでコレクションレキシコンをオンにします）。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。詳細については、「トリプルインデックスの有効化」（62 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」（23 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。次に例を示します。

```
$ 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 トリプルのシリアライゼーション」（36 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。『*Loading Content Into MarkLogic Server 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」（45 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。『*Loading Content Into MarkLogic Server 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。『*Loading Content into MarkLogic Server Guide*』の「Time vs. Correctness: Understanding -fastload Tradeoffs」を参照してください。デフォルト：`/triplestore`

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
-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ロードすると、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、ロード先になるグラフのパーミッションを継承します

インポートオプションの詳細については、『*Loading Content into MarkLogic Server Guide*』の「Import Options」を参照してください。

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。

次に例を示します。

```
# Windows ユーザーは次を参照のこと。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、『*Understanding and Using Security Guide*』の「Understanding Roles」を参照してください。

次に例を示します。

```
# Windows ユーザーは次を参照のこと。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?oldid=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。

次に例を示します。

```
$ 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。

次に例を示します。

```
<?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>
```

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。

次に例を示します。

```
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
```

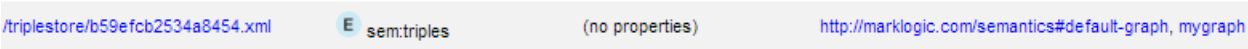
デフォルトでは、sem:rdf-insert はドキュメントをディレクトリ /triplestore/ に配置し、デフォルトグラフを割り当てます。名前付きグラフは、4 番目のパラメータでコレクションとして指定できます。

次に例を示します。

```
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。

次に例を示します。

```
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 8.0-1 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 または同等の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. Admin 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、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オンにします。「トリプルインデックスの有効化」（62 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。次に例を示します。

```
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。

次に例を示します。

```
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。

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」（54 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 トリプルのシリアライゼーション」（36 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. `graph` パラメータを名前付きグラフまたはデフォルト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 トリプルのシリアライゼーション」（36 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。

トリプルがパースされ、デフォルトディレクトリ `/triplestore` にロードされます。

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コレクションに挿入します。

```
# Windows ユーザーは次を参照のこと。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」（54 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。

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：概要と使い方
- 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：概要と使い方

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。

`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。

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」（64 ページ）の説明に従って設定できます。

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。

次に例を示します。

```
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」（62 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

デフォルトでは、データベースを作成するときにトリプルインデックスはオンになっていません。このセクションでは、トリプルインデックス、関連するインデックス、および設定をオンに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。

- データベース設定ページの使用
- Admin API の使用

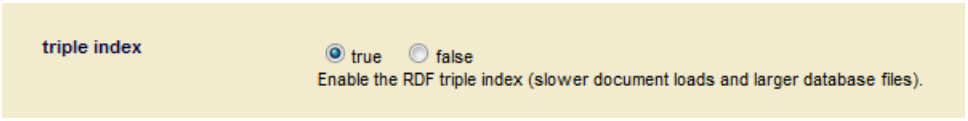
4.2.1 データベース設定ページの使用

トリプルインデックスは、Admin 画面（`http://hostname:8001`）のデータベース設定ページでオンにできます。*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」（23 ページ）を参照してください。

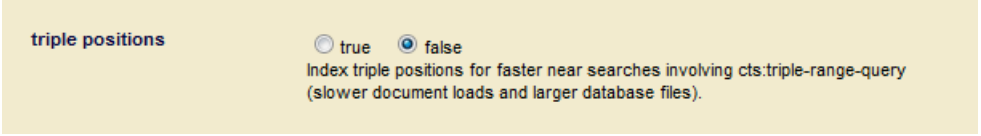
トリプルインデックスおよび関連する設定を構成するには、次の手順を実行します。トリプルポジションインデックス、インメモリのトリプルインデックスサイズ、およびコレクションレキシコンをオンにするには、Admin 画面（`http://hostname:8001`）または Admin API を使用します。詳細については、「Admin API の使用」（63 ページ）を参照してください。

Admin 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。



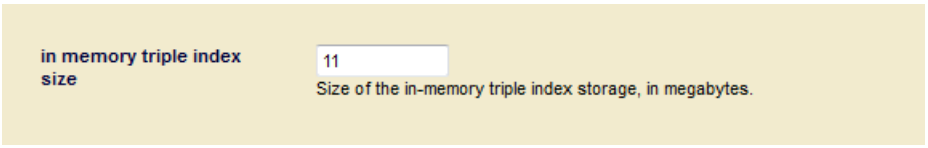
初めてトリプルインデックスをオンにする場合、またはトリプルインデックスをオンにした後でデータベースを再インデックス付けする場合は、有効な `sem:triple` 要素が含まれるドキュメントだけがインデックス付けされます。

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
```

この例は、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)
```

この例は、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 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 $c := admin:database-set-in-memory-triple-index-size($config,
  $Sample-Database, fn:true())
return admin:save-configuration($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。

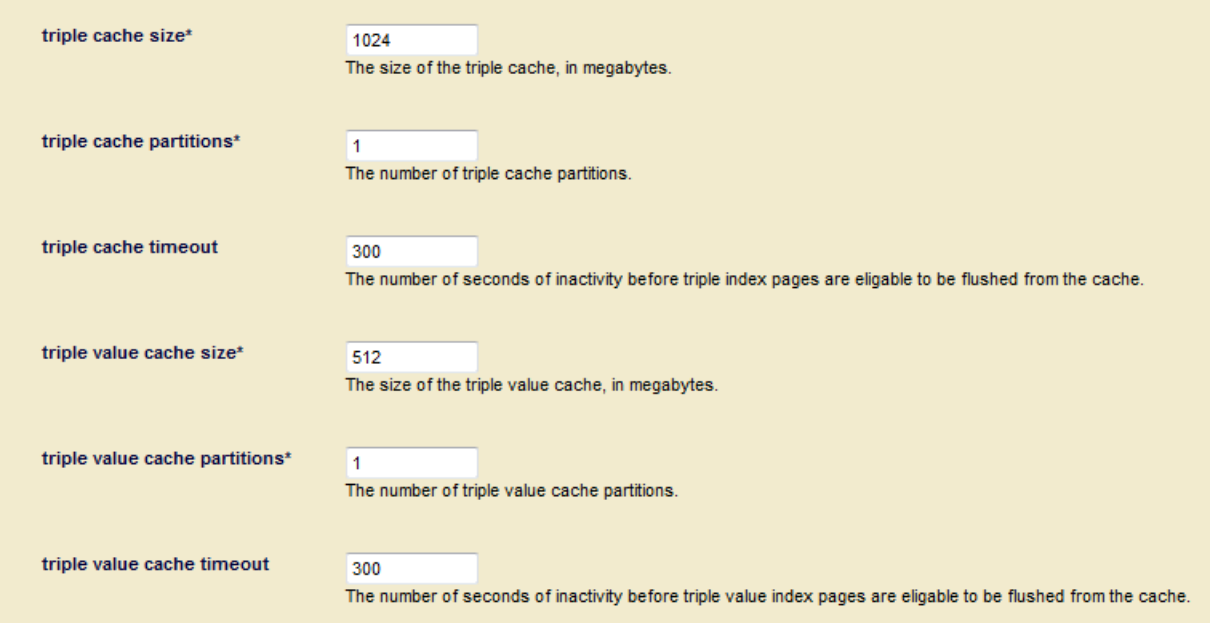
- キャッシュのサイズ設定
- 未使用の値とタイプ
- スケーリングとモニタリング

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。

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、Admin 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。

Admin 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、Admin 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、Admin 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 埋め込みトリプル

トリプルのうち、XML または JSON ドキュメントの一部であり、要素ノード `sem:triples` を持つものは、埋め込みトリプルと呼ばれます。このようなトリプルは、非管理対象トリプル（unmanaged triples）と呼ばれることもあります。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。

> **注：** 埋め込みトリプルは SPARQL Update では修正できません。埋め込みトリプルを修正するには XQuery または JavaScript を使用します。詳細については、「トリプルの更新」（202 ページ）を参照してください。

埋め込み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。

XML ドキュメントの形式で表したシンプルな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:subject>http://example.org/article</sem:subject>
      <sem:predicate>http://example.org/mentions</sem:predicate>
      <sem:object>http://example.org/London</sem:object>
    </sem:triples>
...
  </info>
</article>
```

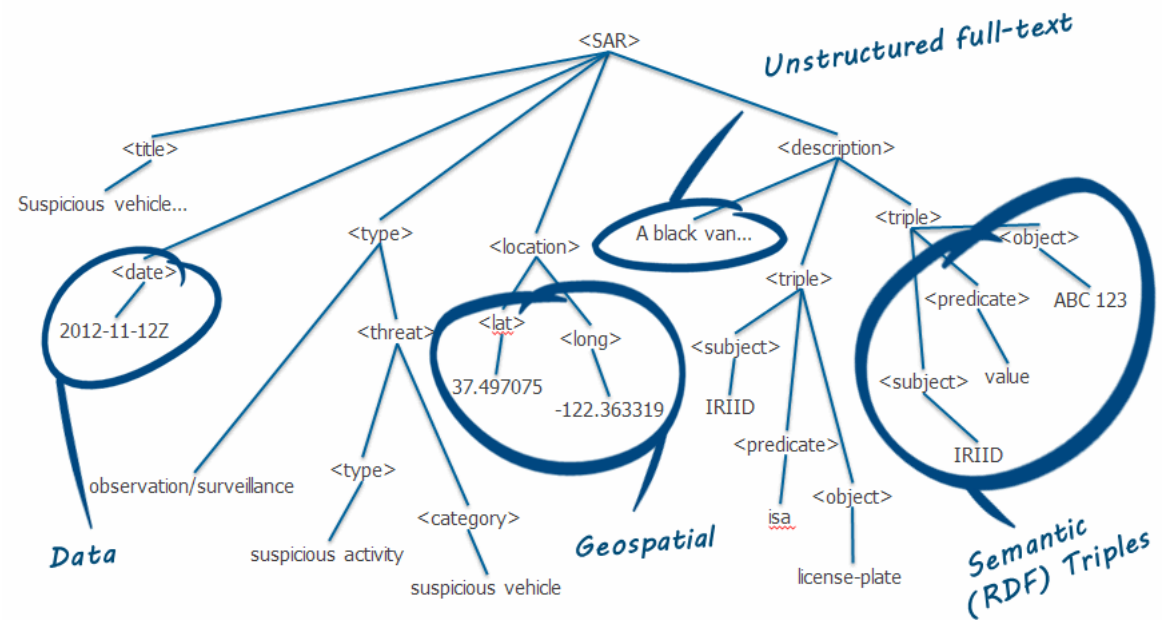
XML または JSON ドキュメントには、多くの種類の情報とともに埋め込みトリプルを格納できます。この例は、疑わしいアクティビティのレポートドキュメントを示したもので、XML とトリプルの両方が含まれます。

```
<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:triples>
        <sem:subject>IRIID</sem:subject>
        <sem:predicate>isa</sem:predicate>
        <sem:object>license-plate</sem:object>
       <sem:triples>
        <sem:subject>IRIID</sem:subject>
        <sem:predicate>value</sem:predicate>
        <sem:object>ABC 123</sem:object>
      </sem:triples>
    </description>
</SAR>
```

MarkLogic データベースに読み込まれた埋め込みトリプルは、トリプルインデックス（triple index）によってインデックスが付けられ、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。`example.json` ドキュメントに埋め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": "John"
 }
}
```

トリプルが XML または JSON ドキュメントに埋め込まれている場合は、トリプルに関する追加情報およびトリプルに関する追加のメタデータを含めることができます（トリプルの日時情報、バイテンポラル情報、ソース）。また、トリプルに関する有用な情報を XML または JSON ファイルに追加することもできます（トリプルの出自など）。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。

5.1 XML ドキュメントでのトリプルの使用方法

埋め込みトリプルを使用すると、ドキュメントと埋め込み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

ドキュメントに埋め込まれたトリプルがある場合、そのドキュメント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することがあります。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。

```
<source>AP Newswire</source>
<sem:triple date="1972-02-21" confidence="100">
  <sem:subject>http://example.org/news/Nixon</sem:subject>
  <sem:predicate>http://example.org/wentTo</sem:predicate>
  <sem:object>China</sem:object>
</sem:triple>
```

また、埋め込みトリプルに注釈を付けることで、情報の信頼度などをさらに提供できます。

5.1.2 複合クエリ

複合クエリは、ドキュメントと任意の埋め込みトリプル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( "//sem:triple/@confidence", ">", 80) ,
```

```
 cts:path-range-query( "//sem:triple/@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" )
  ) )
 ) )
)
```

このクエリは、「ニクソンが訪問した国の情報を含むトリプルをすべて検索します。そのグループの中から、信頼度 80%以上、1974 年 1 月 1 日以降のトリプルだけを返します。さらにそのトリプルは、AP Newswire または BBC が出所でなければなりません」という内容です。

5.1.3 セキュリティ

埋め込みトリプルでは、ドキュメントのセキュリティパーミッションが埋め込み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、『*Understanding and Using Security Guide*』の「Document Permissions」を参照してください。

5.2 バイテンポラル埋め込みトリプル

SPARQL を使用すると、埋め込み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」（29 ページ）を参照してください。

SPARQL と XQuery は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` で制限したり、XQuery のビルトイン関数（全文検索の `cts:contains` など）は SPARQL クエリの一部として呼び出すようにできます。詳細については、「SPARQL クエリにおける XQuery ビルトイン関数の使用」（97 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」（74 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」（115 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 API を使用したトリプルのクエリ」（132 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
- REST API を使用したトリプルのクエリ
- 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://wiki.dbpedia.org/Downloads38/ 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」（25 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、変数と値のバインディングの集まりです。
- 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」（168 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」（79 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」（62 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

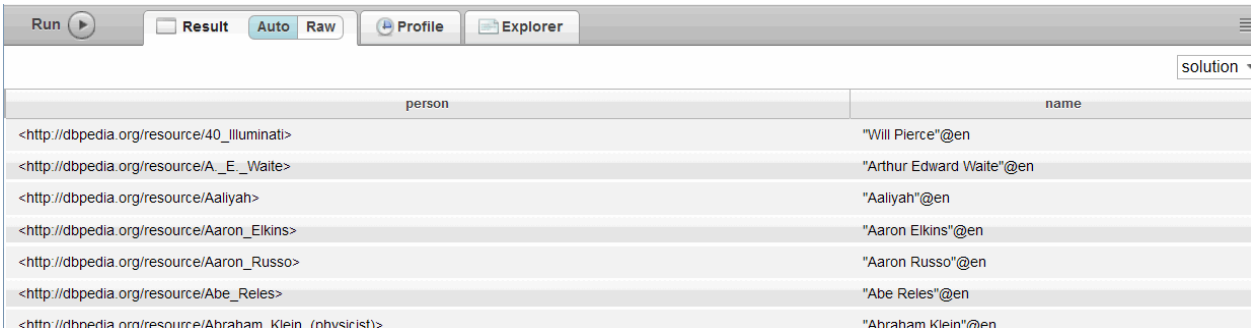
6.1.3.1 Auto 形式と Raw 形式

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":"\"Wil
l
Pierce\"@en"},{"person":"<http://dbpedia.org/resource/A._E._Waite>","n
ame":"\"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>","n
ame":"\"Aaron
Russo\"@en"},{"person":"<http://dbpedia.org/resource/Abe_Reles>","name
":"\"Abe
Reles\"@en"},{"person":"<http://dbpedia.org/resource/Abraham_Klein_(ph
ysicist)>","name":"\"Abraham
Klein\"@en"},{"person":"<http://dbpedia.org/resource/Abraham_S._Fischl
er>","name":"\"Abraham S.
Fischler\"@en"},{"person":"<http://dbpedia.org/resource/Abraham_S._Luc
hins>","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 .
. . . .
```

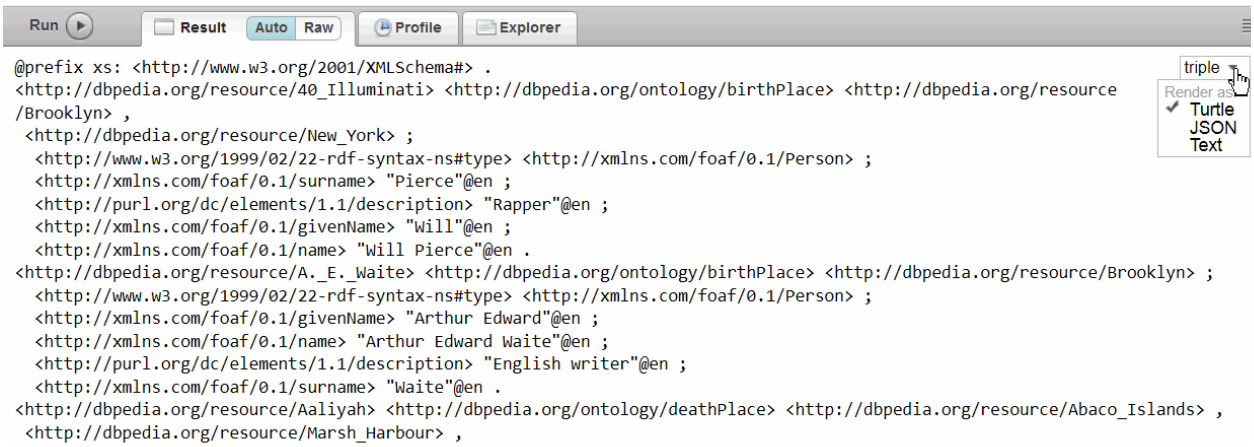
注： トリプルをサブグラフとして返すクエリを実行すると、デフォルトの出力シリアライゼーションは Turtle になります。

6.1.4 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。

6.1.5 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。
- プロパティパス式

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。

6.1.6 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。

6.1.7 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ディングを生成します。

この例は、基本グラフパターンを示したものです。変数がトリプルの主語（?s）と述語（?p）にマッチし、目的語は「Paris」（パリ）です。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れる内容を指定し、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提供します。

```
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」（100 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。

クエリの結果には、目的語の位置に「Paris」がある（birthPlace および deathPlace について）主語および述語の IRI が含まれます。

<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、変数と値のバインディングの集まりです。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」（76 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。

次に例を示します。

```
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 ;
          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。

次に例を示します。

```
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/>

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。

6.1.8 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 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 }
      }
```

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

6.1.8.1 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。

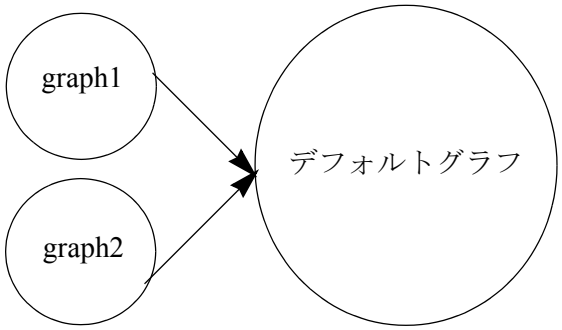
```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT  ?name
FROM    <http://marklogic.com/semantics#info:govtrack/people/>
WHERE   { ?x foaf:name ?name }
LIMIT 10
```

GovTrack データセットについては、「例を実行する準備」（115 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。

6.1.8.2 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」（46 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」（34 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。

次に例を示します。

```
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。

6.1.8.3 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。

次に例を示します。

```
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 Admin 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。

6.1.9 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：変数バインディング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。

6.1.9.1 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ディング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。

次に例を示します。

```
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。

6.1.9.2 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、結果として sem:triple 値のシーケンスがトリプルとして返されます。

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 .}
```

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。

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。

s	p	o
<http://dbpedia.org/resource/Albert_Einstein>	<http://dbpedia.org/ontology/birthPlace>	<http://dbpedia.org/resource/Baden-Württer
<http://dbpedia.org/resource/Albert_Einstein>	<http://dbpedia.org/ontology/birthPlace>	<http://dbpedia.org/resource/German_Empir
<http://dbpedia.org/resource/Albert_Einstein>	<http://dbpedia.org/ontology/deathPlace>	<http://dbpedia.org/resource/Princeton,_Ne
<http://dbpedia.org/resource/Albert_Einstein>	<http://dbpedia.org/ontology/birthPlace>	<http://dbpedia.org/resource/Ulm>
<http://dbpedia.org/resource/Albert_Einstein>	<http://dbpedia.org/ontology/deathPlace>	<http://dbpedia.org/resource/United_States>
<http://dbpedia.org/resource/Albert_Einstein>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://xmlns.com/foaf/0.1/Person>
<http://dbpedia.org/resource/Albert_Einstein>	<http://xmlns.com/foaf/0.1/givenName>	"Albert"
<http://dbpedia.org/resource/Albert_Einstein>	<http://xmlns.com/foaf/0.1/name>	"Albert Einstein"

6.1.9.3 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。結果は、非決定的です。つまり、返される RDF サブグラフの実際の構造は、クエリプロセッサが決定します。

次のクエリは、「Conservationists」と説明されている人々のリストを検索します。

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

DESCRIBE ?person
WHERE {?person dc:description "Conservationist"@en .}
```

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。

クエリパターンから返される IRI を記述するサブグラフは次のようになります。

s	p	o
<http://dbpedia.org/resource/Allen_Bathurst,_9th_Earl_Bathurst>	<http://dbpedia.org/ontology/birthPlace>	<http://dbpedia.org/resource/(
<http://dbpedia.org/resource/Allen_Bathurst,_9th_Earl_Bathurst>	<http://dbpedia.org/ontology/birthPlace>	<http://dbpedia.org/resource/E
<http://dbpedia.org/resource/Allen_Bathurst,_9th_Earl_Bathurst>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://xmlns.com/foaf/0.1/Per
<http://dbpedia.org/resource/Allen_Bathurst,_9th_Earl_Bathurst>	<http://xmlns.com/foaf/0.1/givenName>	"Allen"
<http://dbpedia.org/resource/Allen_Bathurst,_9th_Earl_Bathurst>	<http://xmlns.com/foaf/0.1/name>	"Allen Bathhurst"
<http://dbpedia.org/resource/Allen_Bathurst,_9th_Earl_Bathurst>	<http://xmlns.com/foaf/0.1/surname>	"Bathhurst"
<http://dbpedia.org/resource/Allen_Bathurst,_9th_Earl_Bathurst>	<http://purl.org/dc/elements/1.1/description>	"Conservationist"
<http://dbpedia.org/resource/Allen_Bathurst,_9th_Earl_Bathurst>	<http://dbpedia.org/ontology/birthDate>	"1961-03-11"^^xs:date
<http://dbpedia.org/resource/Charles_Alexander_Sheldon>	<http://www.w3.org/1999/02/22-rdf-syntax-ns#type>	<http://xmlns.com/foaf/0.1/Per
<http://dbpedia.org/resource/Charles_Alexander_Sheldon>	<http://xmlns.com/foaf/0.1/givenName>	"Charles Alexander"
<http://dbpedia.org/resource/Charles_Alexander_Sheldon>	<http://xmlns.com/foaf/0.1/name>	"Charles Alexander Sheldon"
<http://dbpedia.org/resource/Charles_Alexander_Sheldon>	<http://purl.org/dc/elements/1.1/description>	"Conservationist"
<http://dbpedia.org/resource/Charles_Alexander_Sheldon>	<http://xmlns.com/foaf/0.1/surname>	"Sheldon"
<http://dbpedia.org/resource/Charles_Alexander_Sheldon>	<http://dbpedia.org/ontology/birthDate>	"1867-10-17"^^xs:date
<http://dbpedia.org/resource/Charles_Alexander_Sheldon>	<http://dbpedia.org/ontology/deathDate>	"1928-09-21"^^xs:date

6.1.9.4 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。

6.1.10 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
- BIND キーワード
- VALUES セクション

6.1.10.1 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、メールアドレスが含まれるトリプルが存在しない可能性があります。この場合、Query Console ではクエリに対する応答として「your query returned an empty sequence」が返されます。メールアドレスのない人は、結果に表示されません。

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。

6.1.10.2 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。

次に例を示します。

```
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/>

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。

6.1.10.3 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。次に例を示します。

```
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 クエリにおける XQuery ビルトイン関数の使用

関数を使用できる SPARQL クエリでは、SPARQL 関数だけでなくビルトインの XQuery 関数（例えばプレフィックス `fn`、`cts`、または `xdmp` が付いた関数）も使用できます。そのようなクエリには、`FILTER` や `BIND` の他、`SELECT` ステートメント内の式が含まれます。

ビルトイン関数とは、XQuery の「インポートモジュール」を使用せずに呼び出せる関数です。SPARQL クエリで使用する場合、このような XQuery 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。

SPARQL の拡張関数は、`http://www.w3.org/2005/xpath-functions#`*`name`*`/` という形式の IRI 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（例えば `http://www.w3.org/2005/xpath-functions#starts-with/`）です。一般的に `fn`、`xdmp`、および `cts` に関連付けられたプレフィックス IRI の場合（または末尾が「`/`」や「`#`」ではないその他のプレフィックス IRI の場合）、`#` をプレフィックス IRI に付加し、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。

6.1.10.4 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。次に例を示します。

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

=>
true

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

=>
false
```

6.1.10.5 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。次に例を示します。

```
BIND (2014 - ?birthYear AS ?age) .
```

6.1.10.6 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。次に例を示します。

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

これは、パラメータのリストが丸括弧内に含まれている、より長い形式を使用した場合と同じです。

```
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。

次に例を示します。

```
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。

次に例を示します。

```
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.}
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、バインドされていない変数、空白ノード、IRI、または RDF リテラルを順序付けすることもできます。

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

デフォルトの順序は昇順です。

次に例を示します。

```
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" ;
      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。

次に例を示します。

```
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 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 10 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 10
```

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

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、誕生地が London である Politician をクエリし、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}
```

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 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。

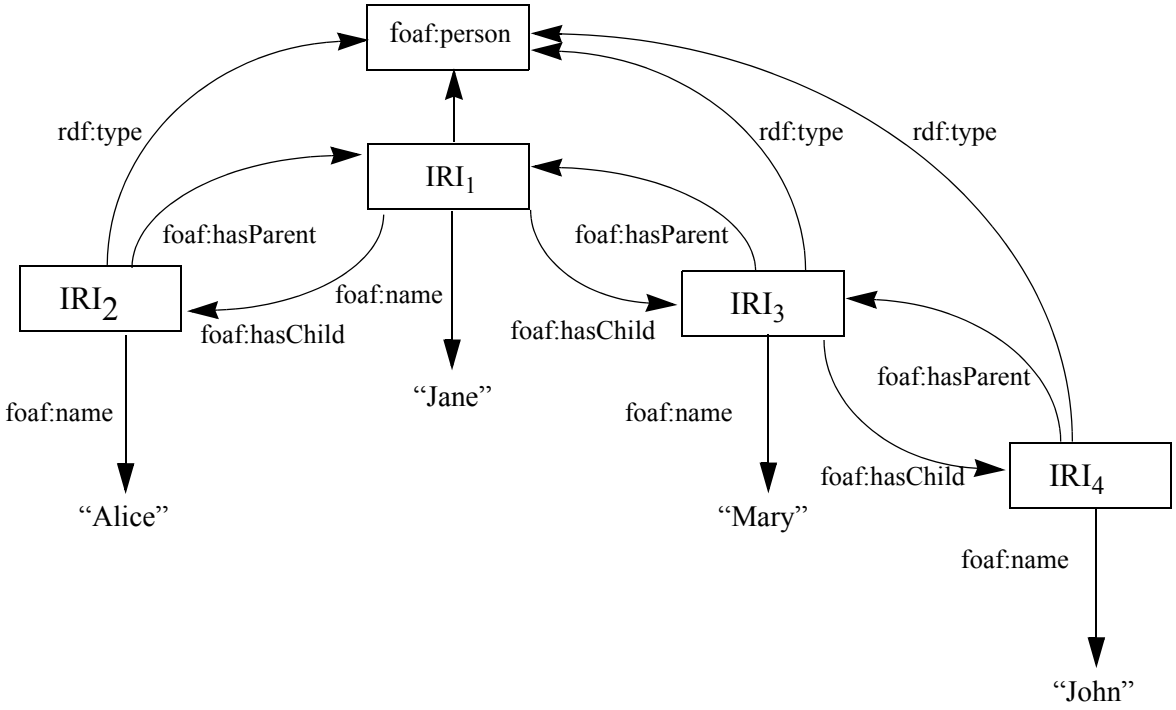
クエリを評価すると、パス式のすべてのマッチが特定され、主語または目的語が適宜バインドされます。記録されるマッチは、グラフを通じたルートごとに 1 つだけです。そのため、任意のパス式に対する重複はありません。

6.1.12.1 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 ;
  foaf:name "John" ;
  <\u003chttp://www.w3.org/1999/02/22-rdf-syntax-ns#type/>
  foaf:Person .
```

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
"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
"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 { "Mary" ^foaf:hasChild ?s }

=>
  s
"Jane"
```

6.1.12.2 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。
反転	`^path`	主語から目的語へパスを反転させます。

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、次のトリプルを追加して `foaf:knows` の概念を表現しています。

```
@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 .
```

アリス（Alice）に接続されている全員の名前を調べるには、`foaf:knows` と「+」パス演算子を使用します。

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

SELECT ?s
WHERE {?s <http://marklogic.com/semantics/”Alice”> .
    ?s foaf:knows+/foaf:name ?name .}
```

```
=>
  s
"Jane"
"John"
"Mary"
"Alice"
```

このクエリは、1 つあるいは複数のパスが存在する foaf:knows によって「Alice」に接続されているすべてのトリプルにマッチします。

foaf:knows と「*」を使用すると、アリスに接続されている全員（アリスを含む）の名前を調べることができます。これはマッチングパスがない場合でも機能します。

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

SELECT ?s
WHERE {?s <http://marklogic.com/semantics/"Alice" .
?s foaf:knows*/foaf:name ?name .}
```

結果は、前出の例と同様です。ただし、「*」パス演算子を使用しているため、アリスに接続されている人数はゼロあるいは複数の可能性があります。

foaf:knows と「?」演算子を使用すると、あるパス要素内に存在する全員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 "Alice" .
?s foaf:knows?/foaf:name ?name .}

=>
"Alice"
```

結果はアリスに接続されている人々について、ゼロあるいは 1 つのマッチになります。

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）。

アリスに接続されていて子供がいる全員を調べる場合は、次の複合クエリを使用できます。

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

SELECT ?s ?name
WHERE {?s foaf:name "Alice" .
?s foaf:knows?/foaf:name ?name .}
FILTER cts:contains(?p, cts:contains("foaf:hasChild"))
}
```

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。

6.1.12.3 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、RDFS ではリソースのすべてのタイプとスーパー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/TR/rdf-schema/>

SELECT ?type
{
   <http://example/thing> rdf:type/rdfs:subClassOf* ?type
}
```

また、すべてのリソースとその推論タイプを推論する場合は次のようにします。

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

SELECT ?x ?type {
?x rdf:type/rdfs:subClassOf* ?type
}
```

推論の詳細については、「推論」（150 ページ）を参照してください。

6.1.13 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)
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/ProductType4
59> .
    ?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
```

6.1.14 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。例えば次のようなものが挙げられます。

- 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検索
- 変数に対するバインディングの使用
- 結果の XML および RDF 表示
- CURIE を扱う
- search API を使用したクエリ

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 8.0-1 以降
- mlcp（MarkLogic Content Pump）。『*Loading Content Into MarkLogic Server Guide*』の「Downloading and Installing mlcp」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、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 MarkLogic REST API Instances」で詳細な情報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をオンにします。「トリプルインデックスの有効化」（62 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」（42 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。

   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'
```

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検索

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、および『*Node.js Application Developer's Guide*』の「セマンティックトリプルのクエリ」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」（115 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。

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 Av \"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 Av \"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 Av \"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」（79 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、バインディングのシーケンスで 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」（99 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ディングを渡すことで、変数を固定値に設定できます。「変数に対するバインディングの使用」（121 ページ）を参照してください。

6.2.2.3 sem:store

`sem:store` オプションは、`sem:sparql`、`sem:sparql-values`、または `sem:sparql-update` を使用する SPARQL クエリのトリプルのソースを示す目的で使用できます。複数の `sem:store` 値を指定した場合は、すべてのソースからのトリプルがマージされてクエリされます。

`sem:store` 値が `sem:sparql`、`sem:sparql-values`、または `sem:sparql-update` のオプションとして指定されていない場合、そのクエリのデフォルトの `sem:store` 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検索できます。

次に例を示します。

```
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ディングの使用

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ディングを使用できます。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ディングを渡すことで、変数を固定値に設定できます。

変数に対するバインディングは、(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ディングを `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>
```

バインディングは、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ディングの例については、「変数のバインディング」（182 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 形式の変数バインディングとして返します。

```
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。

次に例を示します。

```
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 .}
'), "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。

次に例を示します。

```
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")
```

シリアライゼーションオプションは、次の表に示すとおりです。

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」（78 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。

次に例を示します。

```
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。

次に例を示します。

```
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」（59 ページ）を参照してください。

6.2.6 search API を使用したクエリ

このセクションでは、search API を使用して、MarkLogic トリプルストアから 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。

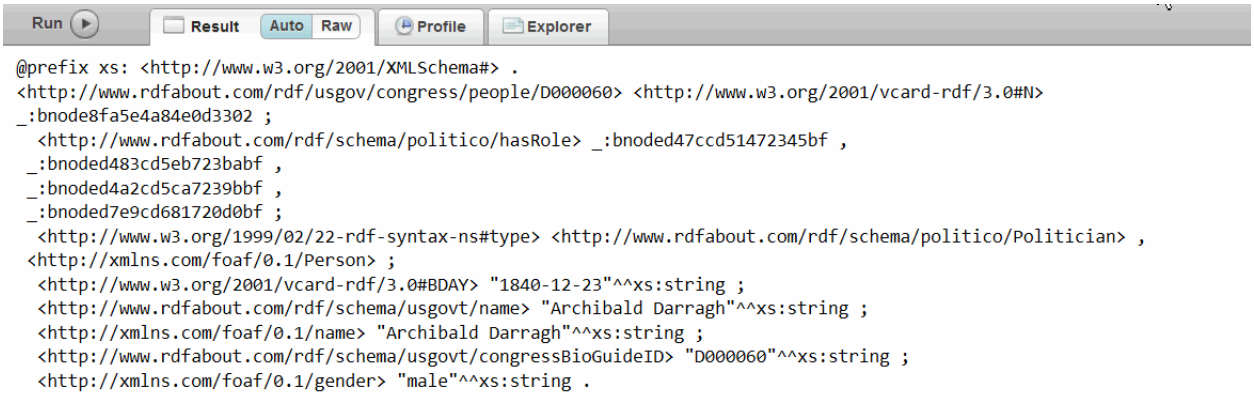
この例では、議員の主語 IRI が主語 IRI の 1 番目のパラメータとして渡されます。

```
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検索アクセスは、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` および `NOT IN` 演算子でシーケンスを構築することで、`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` などのビルトイン関数を効率的に利用できます。

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。

次に例を示します。

```
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 REST API を使用したトリプルのクエリ

このセクションでは、W3C SPARQL プロトコルの実装である SPARQL エンドポイントを使用してトリプルデータをクエリする方法について説明します。このエンドポイントの URL には、次のようなアドレスが割り当てられます。

```
http://host:port/v1/graphs/sparql?
```

このセクションでは、次の内容を取り上げます。

- 前提
- パラメータの指定
- POST リクエストでの SPARQL クエリの実行
- GET リクエストでの SPARQL DESCRIBE クエリ
- サポートされている動詞
- 応答の出力形式
- REST API を使用したトリプルの操作

注： SPARQL エンドポイントを使用するには、rest-reader ロールに対応する読み取りパーミッションが必要であり、現在の環境のセキュリティ要件を満たす必要があります。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。

6.3.1 前提

このセクションの各例は、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」（115 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` または同等の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` に慣れていない場合、またはシステムで `curl` を使用できない場合は、『*REST Application Developer's Guide*』の「Introduction to the curl Tool」を参照してください。

6.3.2 パラメータの指定

`POST:/v1/graphs/sparql` または `GET:/v1/graphs/sparql` を使用する SPARQL エンドポイントでは、次のパラメータがサポートされています。

- query：実行する SPARQL クエリ
- （オプション）default-graph-uri：
  `http://host:port/v1/graphs/sparql?default-graph-uri=<default-graph-uri*>` というアドレスが割り当てられます。
- （オプション）named-graph-uri：
  `http://host:port/v1/graphs/sparql?named-graph-uri=<named-graph-uri*>` というアドレスが割り当てられます。

  `default-graph-uri` は、クエリするグラフ名を選択します。「`*`」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。

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、クエリ自体の外部にあるデータセットを指定する目的で使用できます。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 経由でクエリに複数の 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。

この REST リクエストには、2 つの名前付きグラフ IRI が指定されています。

```
curl --anyauth --user admin:password
"http://localhost:8000/v1/graphs/sparql?named-graph-uri=http://marklog
ic.com/info:govtrack/bills
&named-graph-uri=http://marklogic.com/info:govtrack/people"
-H "Content-type:application/x-www-form-urlencoded"
-H "Accept:application/sparql-results+xml"
-X POST --data-binary 'query=PREFIX%20rdf%3A%20%3Chttp%3A%2F%2F
www.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%0APREFIX%20people%3A%20%3Chttp%3A%2F%2F
www.rdfabout.com%2Frdf%2Fusgov%2Fcongress%2Fpeople%2F%3E%0A
SELECT%20%3Fx%20
WHERE%20%7B%20%3Fx%20rdf%3Atype%20bill%3AHouseBill%20%3B%20bill%3A
cosponsor%20%20people%3AA000358.%20%7D%0A'
```

このクエリは REST によって渡され、2 つの名前付きグラフ http://marklogic.com/info:govtrack/bills および http://marklogic.com/info:govtrack/people に対して検索を実行します。詳細については、「ターゲット RDF グラフ」（84 ページ）を参照してください。

6.3.3 POST リクエストでの SPARQL クエリの実行

POST リクエストを使用して REST API でトリプルをクエリする場合は、次の 2 つの手法を使用できます。

- 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 クエリを示したものです。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。

ファイルの内容は次のとおりです。

```
#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" . }
```

SPARQL エンドポイントでは、パラメータまたは POST のボディとして SPARQL クエリを必要とします。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 ユーザーは次を参照のこと。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
```

注： 読みやすくするため、長いコマンドは 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。コマンドをコピー&ペーストするときは、これらの継続文字や改行を取り除いてください。

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="bnodebnode309771418819f878"
    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
.....
```

クエリの Content-type: には、application/sparql-query が指定されています。また、コンテンツ応答タイプ（Accept:）では出力として application/rdf+xml が指定されています。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」（139 ページ）を参照してください。

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。

POST リクエストを使用する別の方法としては、『*REST Client API*』のセマンティックのドキュメントの説明に従って、フォームエンコードされた手法で URL エンコードされたクエリを含める方法が挙げられます。

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%2F02%2F22-r
df-syntax-ns%23%3E%20%0APREFIX%20bill%3A%20%3Chttp%3A%2F%2Fwww.rdfabou
t.com%2Frdf%2Fschema%2Fusbill%2F%3E%0APREFIX%20people%3A%20%3Chttp%3A%
2F%2Fwww.rdfabout.com%2Frdf%2Fusgov%2Fcongress%2Fpeople%2F%3E%0ASELECT
%20%3Fx%20WHERE%20%7B%20%3Fx%20rdf%3Atype%20bill%3AHouseBill%20%3B%20b
ill%3Acosponsor%20%20people%3AA000358.%20%7D%0A"
-H "Content-type:application/x-www-form-urlencoded"

=>

<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： 見やすくするため、長いコマンドラインは Windows 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。この例を実行するときは、行継続文字や追加の改行を削除してください。

6.3.4 GET リクエストでの SPARQL DESCRIBE クエリ

クエリリクエストパラメータの SPARQL クエリは URL エンコードされている必要があります。この例では、curl が HTTP GET リクエストを送信し、米国議会法案（44）に対して SPARQL の DESCRIBE クエリを実行します。

```
curl -X GET --digest --user "user:password" ^
   -H "accept: application/sparql-results+xml"  ^
   "http://localhost:8050/v1/graphs/sparql?query=PREFIX%20rdf%3A%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%2Fschema
   %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。

6.3.5 サポートされている動詞

SPARQL エンドポイントに対するリクエストでサポートされる動詞は次のとおりです。

動詞	説明	権限
GET	URL エンコードされた SPARQL に基づいてクエリを実行します。名前付きグラフを取得します。	rest-reader
POST	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 クエリに基づいてクエリを実行します。 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。	rest-writer
PUT	SPARQL Update を使用すると、名前付きグラフ内のトリプルが置換されます。空のグラフで PUT を使用すると、グラフが削除されます。	rest-writer
DELETE	SPARQL Update を使用すると、名前付きグラフ内のトリプルが削除されます。	rest-writer
HEAD	コンテンツなしで、名前付きグラフを取得します。	rest-reader

SPARQL Update の詳細については、「REST での SPARQL Update の使用」（186 ページ）を参照してください。

6.3.6 応答の出力形式

REST 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、結果の出力形式が変化することがあります。応答タイプの形式は、クエリのタイプと、REST リクエストの Accept ヘッダタイプで使用されるパラメータによって異なります。Accept ヘッダタイプでは、応答出力で予期される MIME タイプ（MIME type）を設定します。次の表は、さまざまな SPARQL クエリに対する Accept ヘッダタイプおよび応答出力の形式を説明したものです。

クエリタイプ	Accept ヘッダタイプ	応答出力
SELECT	application/sparql-results+xml	W3C 規格の定義に従って、結果を XML で返します。「例：XML として結果を返す」（141 ページ）を参照してください。
SELECT	application/sparql-results+json	W3C 規格の定義に従って、結果を JSON で返します。「例：JSON として結果を返す」（142 ページ）を参照してください。
SELECT	text/html	結果を HTML で返します。「例：HTML として結果を返す」（143 ページ）を参照してください。
CONSTRUCT	application/sparql-results+xml または application/sparql-results+json	サポートされている任意の形式のトリプルとして結果を返します。「サポートされている RDF トリプルのシリアライゼーション」（36 ページ）を参照してください。
DESCRIBE	application/rdf+xml	結果をトリプルとして XML 形式で返します。「GET リクエストでの SPARQL DESCRIBE クエリ」（138 ページ）を参照してください。
ASK	application/sparql-results+xml??? application/sparql-results+json	ブール型の値（true または false）を XML として返します。「例：XML または JSON としてブール型の値を返す」（144 ページ）を参照してください。

SPARQL の SELECT クエリは結果を XML、JSON、または HTML として返すことができますが、SPARQL の CONSTRUCT クエリはサポートされている任意のトリプル形式のトリプルとして結果を返します。SPARQL の DESCRIBE クエリは、クエリによって見つかったトリプルを記述するトリプルを XML 形式で返します。SPARQL の ASK クエリを使用した場合は、ブール型の値（true または false）を返します。

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。このクエリは bills.sparql として保存されていて、結果を 5 件の法案に制約します。

ファイルの内容は次のとおりです。

```
#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
```

6.3.6.1 例：XML として結果を返す

次の例では、SPARQL の 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>
     <bindingname="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>
```

6.3.6.2 例：JSON として結果を返す

次の例では、SPARQL の SELECT 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"}},
...
}]}}
```

6.3.6.3 例：HTML として結果を返す

次の例では、SPARQL の 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><ahref="/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><ahref="/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><ahref="/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>
```

6.3.6.4 例：XML または JSON としてブール型の値を返す

次の例では、SPARQL の ASK クエリが、（以前の例の）Carolyn が Eunice よりも後に誕生したかどうかを判断するために使用されています。

ファイルの内容は次のとおりです。

```
#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).
}
```

SPARQL の ASK クエリが、ブール型の結果を XML として返します。

```
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>
```

同じ SPARQL の ASK クエリで、ブール型の結果が JSON として返されるようにする場合は、次の例のようになります。

```
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}
```

6.3.6.5 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 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。

次に例を示します。

```
<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>
```

6.3.7 REST API を使用したトリプルの操作

次のエンドポイントは、データベース内で参照されている知識（物）への RESTful なアクセスを提供します。

```
http://hostname:port/v1/graphs/things?
```

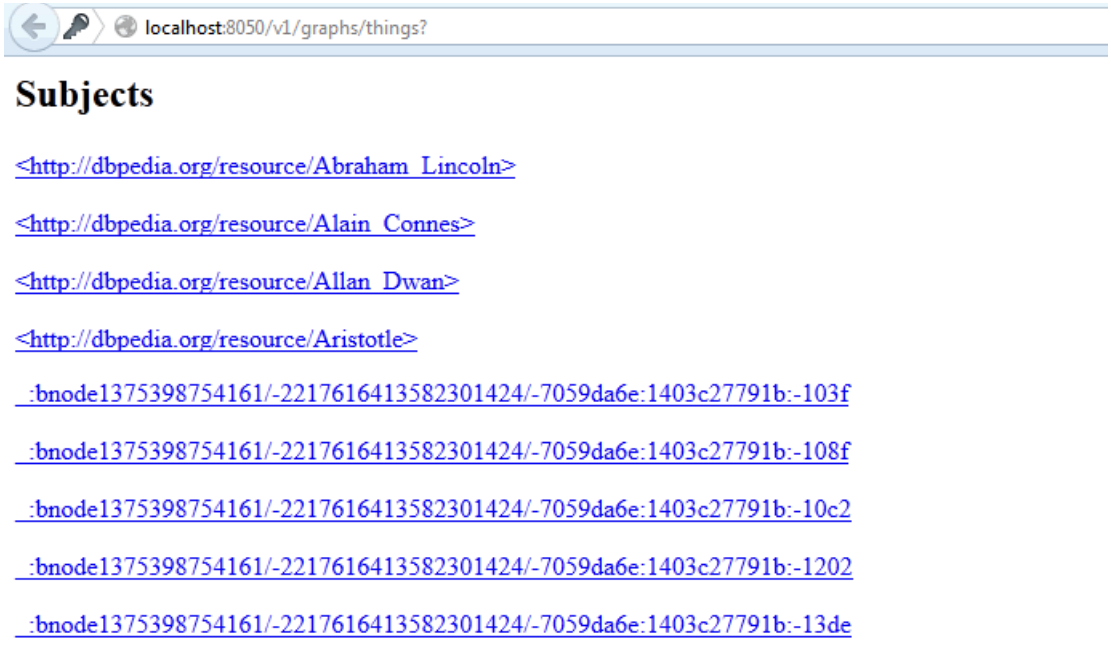
hostname および *port* は、MarkLogic を実行しているホストとポートです。

次に例を示します。

```
http://localhost:8050/v1/graphs/things?
```

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。

Web ブラウザにおける応答の例は、次に示すとおりです。



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。

ノードをドリルダウンすると、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。

例えば、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。

5 triples

_:bnode1375398754161/-2217616413582301424/-7059da6e:1403c27791b:-1145
<http://pervasive.semanticweb.org/ont/2004/06/time#to>
<-7059da6e:1403c27791b:-6a9a> .
_:bnode1375398754161/-2217616413582301424/-7059da6e:1403c27791b:-1145
<http://pervasive.semanticweb.org/ont/2004/06/time#from>
<-7059da6e:1403c27791b:-6a9b> .
_:bnode1375398754161/-2217616413582301424/-7059da6e:1403c27791b:-1145
<http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.rdfabout.com
/rdf/schema/politico/Term> .
_:bnode1375398754161/-2217616413582301424/-7059da6e:1403c27791b:-1145
<http://www.rdfabout.com/rdf/schema/politico/forOffice>
<http://www.rdfabout.com/rdf/usgov/congress/house/33/in> .
_:bnode1375398754161/-2217616413582301424/-7059da6e:1403c27791b:-1145
<http://www.rdfabout.com/rdf/schema/politico/party> "Democrat" .

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。

次に例を示します。

```
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。

localhost:8050/v1/graphs/things?iri=http://dbpedia.org/resource/Abraham_Lincoln

6 triples

<http://dbpedia.org/resource/Abraham_Lincoln> <http://dbpedia.org/ontology/birthPlace> <http://dbpedia.org/resource/
<http://dbpedia.org/resource/Abraham_Lincoln> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://xmlns.co
<http://dbpedia.org/resource/Abraham_Lincoln> <http://purl.org/dc/elements/1.1/description> "16th President of the Un
<http://dbpedia.org/resource/Abraham_Lincoln> <http://xmlns.com/foaf/0.1/name> "Abraham Lincoln"@en .
<http://dbpedia.org/resource/Abraham_Lincoln> <http://dbpedia.org/ontology/birthDate> "1809-02-12"^^xs:date .
<http://dbpedia.org/resource/Abraham_Lincoln> <http://dbpedia.org/ontology/deathDate> "1865-04-15"^^xs:date .

`curl` または HTTP リクエストを発行する同等のコマンドラインツールを使用している場合は、次の `accept` をリクエストヘッダに追加できます。

- パラメータが指定されていない場合は text/html。
- `iri` パラメータが使用されている場合は MIME タイプ。「サポートされている RDF トリプルのシリアライゼーション」（36 ページ）を参照してください。

  注： このエンドポイントでは、`GET` が唯一のサポートされている動詞です。要求した情報を返すには、rest-reader 権限が必要です。

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。

6.4 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、『*Understanding and Using Security Guide*』の「Document Permissions」を参照してください。

7.0 推論

MarkLogic のセマンティックのコンテキスト、および一般的なセマンティック技術において、「推論」のプロセスには、データの組み合わせに基づく新しいファクトとそのデータを理解するためのルールを自動的に発見する処理が含まれます。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。

推論クエリ（inference query）は、自動推論の影響を受ける SPARQL クエリであり、コンピュータプログラムによる自動プロセスです。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、バックワードチェーン推論をサポートしています。

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
- 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、クエリごとに使用するルールセットおよびオントロジー（それぞれ複数可）を指定でき、データベースごとにデフォルトを指定できます。

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

7.1.1 オントロジー

オントロジーとは、データを記述するためのモデルです。使用する情報が増えるほど、データに関して推論されるファクト（データ内の何が他のデータの何と関連があるのか、それはどのような関連なのか）が増えます。セマンティックでは、オントロジー（ontology）は世界の一部を構成するセマンティックモデルを提供するトリプルの集まりです。この知識モデルは、データ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。

例えばオントロジートリプル `ex:color owl:equivalentProperty ex:hue` は、hue と color が等価なプロパティであることを示します。

この SPARQL Update の例では、オントロジートリプルをグラフに挿入します。

```
PREFIX owl: <http://www.w3.org/2002/07/owl/>
PREFIX ex: <http://example.org/>

INSERT DATA
{
GRAPH <http://marklogic.com/semantics/products/inf-1>
{
ex:color owl:equivalentProperty ex:hue .
}
}
```

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。

推論用のオントロジーは、次のようなさまざまな方法で選択できます。

- 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に含めるか除外します。オントロジーは、ディレクトリ別、または cts:query で発見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

ルールセットとは、推論ルール（追加のファクトを推論するために使用できるルール）の集まりです。ルールセットは、クエリ時に既存のトリプルから新しいトリプルを推論するために、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。

クエリ時にルールセットを使用して推論を実行する場合、その推論は「バックワードチェーン」推論とも呼ばれます。ルールはクエリ時に適用されます。つまり各 SPARQL クエリは指定されたルールセットを参照し、その結果とし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。

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現しているトリプルがある場合、MarkLogic はジョンがイングランドに住んでいることを推論します。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）の概念を表すシンプルなルールを次に示します。

```
# 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。

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。

7.1.2.1 ルールセットの使用

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。1 つあるいは複数のルールセットをデータベースと関連付けることができるため、デフォルトでは、そのデータベースに対するクエリにはルールセットが含まれます。クエリ時は、クエリごとに 1 つあるいは複数のルールセットを指定することもできます。

MarkLogic には、標準に基づくルールセット（RDFS、RDFS-Plus、および OWL Horst）が用意されています。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。クエリに合わせたきめ細やかな推論を実行できるように、このような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。このようなルールセットをいくつかインポートするか独自のルールを作成することで、独自のルールセットを作成できます。

Linux でこのような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`）。ここには、`.rules` 拡張子を持つ一連のファイルが格納されています。

```
/opt/MarkLogic/Config/*.rules
```

`.rules` ファイルのそれぞれがルールセットです。テキストエディタで開くと、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大規模なルールセットになっています。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/>

...

rule "domain rdfs2" CONSTRUCT {
  ?x a ?c
} {
  ?x ?p ?y .
  ?p rdfs:domain ?c
}
```

この例で、a は「タイプ」（type of）を意味します（`rdf:type` または `rdfs:type`）。このルールは、2 番目の括弧内のすべてが、トリプル（`p` に定義域 `c` がある、つまり述語 `p` を持つすべてのトリプルについて目的語は定義域 `c` 内にある）にマッチする場合は、1 番目の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。

ルールの作成と使用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。

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、`no-default-ruleset` オプションを使用します。

7.1.2.2 クエリのルールセットの選択

`sem:ruleset-store` を使用すると、SPARQL クエリで使用するルールセットを選択できます。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() )
```

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。

この例では、推論ルール `rdfs:subClassOf` および `rdfs:subPropertyOf` を使用して、`$triples` 内のデータに対して SPARQL クエリを実行します。

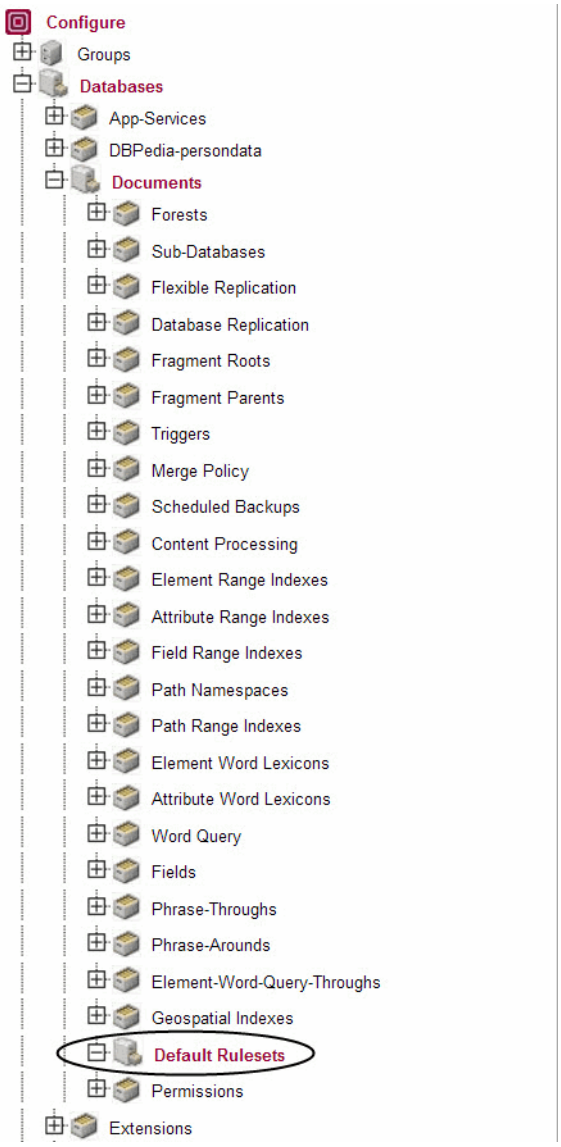
```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

PREFIX skos: <http://www.w3.org/2004/02/skos/core#Concept/>

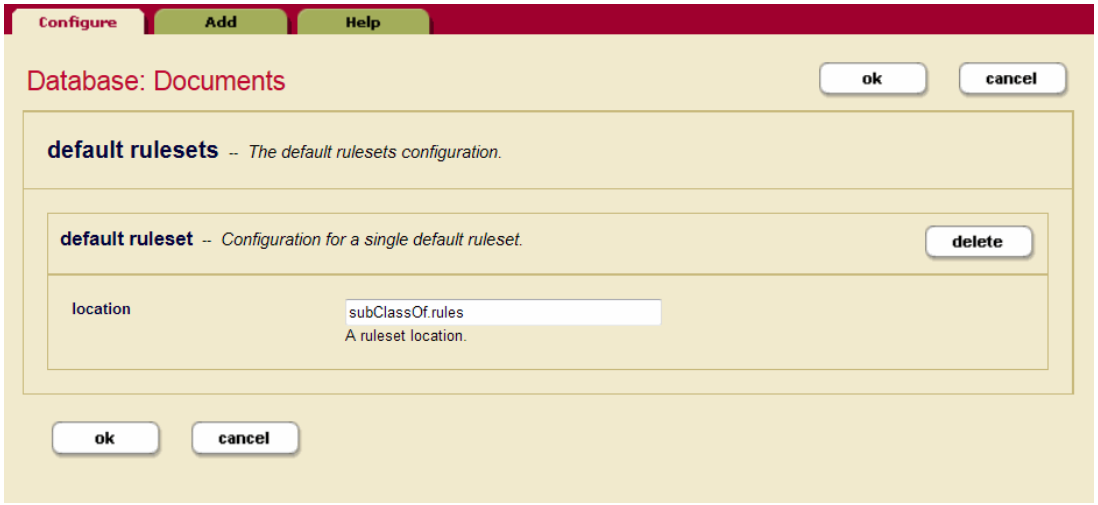
sem:sparql("select * { ?c a skos:Concept; rdfs:label ?l }",(),(),
sem:ruleset-store(("subClassOf.rules","subPropertyOf.rules"),
  ($triples)
)
```

7.1.2.3 データベースのデフォルトルールセットの指定

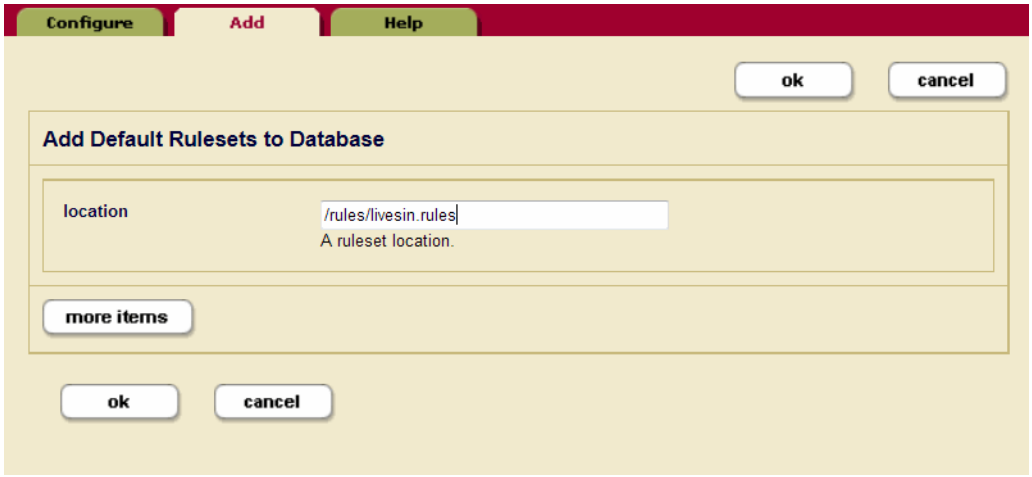
Admin API と XQuery に加え、クエリするデータベースで使用するデフォルトルールセットを設定する場合は、Admin UI も使用できます。データベースのルールセット（複数可）を指定するには、Admin UI の左側にあるナビゲーションで、［Databases］の下にあるデータベース名をクリックします。データベース名をクリックすると、リストが展開します。［Default Rulesets］までスクロールします。



［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。



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。



カスタムルールセットは、Schemas データベース内に配置されます。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`）。

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>
```

7.1.2.4 デフォルトルールセットの上書き

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。

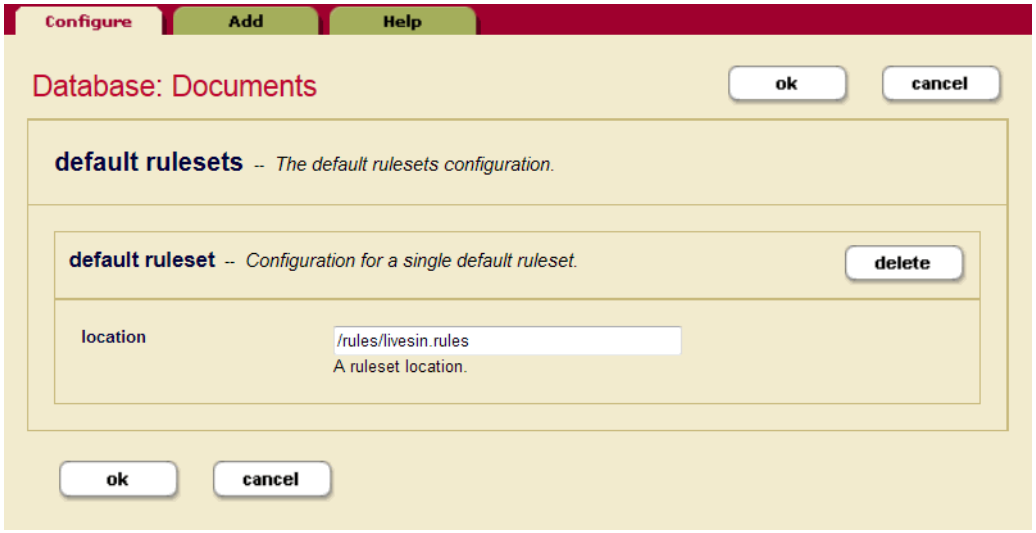
```
xquery version "1.0-ml";
import module namespace sem = "http://marklogic.com/semantics"
  at "/MarkLogic/semantics.xqy";

PREFIX skos: <http://www.w3.org/2004/02/skos/core#Concept/>

sem:sparql("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 ルールセットの作成

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ができます。構築後は、新しいデータセット（データベース（sem:store）および推論トリプルを含むデータセット）に対して検索を実行できます。MarkLogic のルールセットは、拡張子が .rules であり、インストールディレクトリ内にあります。

```
/<MarkLogic_install_dir>/Config/*.rules
```

注： 独自のルールセットを作成すると、Schemas データベース内に格納されます。

推論ルールのシンタックスでは、SPARQL の CONSTRUCT の文法を使用します。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。同一の名前が複数使用されている場合、エラーが発生します。ルールセットファイルのプロローグの import ステートメントには、指定場所にあるルールセットからのすべてのルールが記述されています。

このルールセットは /<MarkLogic Install>/Config ディレクトリに配置されており、rdfs8、rdfs9、rdfs10、rdfs11 という 4 つのルールで構成されています。このルールセットにはプレフィックスが含まれ、各ルールはルール名と CONSTRUCT 節を持ちます。

```
PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:    <http://www.w3.org/2000/01/rdf-schema#>

rule "rdfs8" CONSTRUCT {
  ?c rdfs:subClassOf rdfs:Resource
} {
  ?c a rdfs:Class
}

rule "rdfs9" CONSTRUCT {
  ?x a ?c2
} {
  ?x a ?c1 .
  ?c1 rdfs:subClassOf ?c2 .
  FILTER(?c1!=?c2)
}

rule "rdfs10" CONSTRUCT {
  ?c rdfs:subClassOf ?c
} {
  ?c a rdfs:Class
}

rule "rdfs11" CONSTRUCT {
  ?c1 rdfs:subClassOf ?c3
} {
  ?c1 rdfs:subClassOf ?c2 .
```

```
  ?c2 rdfs:subClassOf ?c3 .
  FILTER(?c1!=?c2 && ?c2!=?c3 && ?c1!=?c3)
}
```

ルールのうち 2 つには FILTER 節も含まれていることに注意してください。

同じディレクトリにあるこのルールセットは、さらに小さなルールセットをインポートして、完全な RDFS ルールセットを近似するルールセットを作成しています。

```
import "rdf.rules"
import "domain.rules"
import "range.rules"
import "subPropertyOf.rules"
import "subClassOf.rules"

PREFIX rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:    <http://www.w3.org/2000/01/rdf-schema#>

# Miscellaneous other axiomatic triples

rule "rdfs properties" CONSTRUCT {
  rdf:type rdfs:domain rdfs:Resource .
  rdf:type rdfs:range rdfs:Class .
}
```

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別の場所から複数回インポートした場合（例えば /<MarkLogic Install>/Config ディレクトリから、および Schemas データベースディレクトリから）、それらのルールセットはそれぞれ異なるルールセットであるとみなされ、重複したルール名がある場合はエラーが発生します。

次の例は、地理的な場所を推論する場合に作成するルールを示したものです。

```
# 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
}
```

Query Console で、xdmp:document-insert を使用して livesIn ルールを Schemas データベースに追加できます。次のコードを実行する前に、コンテンツソースとして Schemas データベースが選択されていることを確認してください。

```
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
  }
'
}
)
```

この例では、Schemas データベースの rules ディレクトリに `livesin.rule` が格納されます（`/rules/livesin.rules`）。

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を確認してから、付属のルールセットの場所を確認します。

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。

- パスを使用する推論
- マテリアライズ

7.2.1 パスを使用する推論

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）を使用して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」（104 ページ）で説明）を使用すると、シンプルな種類の推論が可能です。

例えば、RDFS を使用する場合、「/」プロパティパスを使用することで、この SPARQL クエリで、リソースのすべてのタイプおよびスーパー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/TR/rdf-schema/>
SELECT ?type
{
   <http://example/thing> rdf:type/rdfs:subClassOf* ?type
}
```

また、すべてのリソースとその推論タイプを推論する場合は次のようにします。

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns>
PREFIX rdfs: <http://www.w3.org/TR/rdf-schema/>
SELECT ?x ?type
{
   ?x rdf:type/rdfs:subClassOf* ?type
}
```

アスタリスク（*）が含まれる非列挙プロパティパス式は、パスの主語と目的語を 0 個以上の述語のマッチで接続するパスを探します（この例では `rdfs:subClassOf`）。

プロパティパスの詳細については、「プロパティパス式」（104 ページ）を参照してください。

7.2.2 マテリアライズ

自動推論の代替方法の 1 つがマテリアライズで、フォワードチェーン推論（forward-chaining inference）とも呼ばれます。この方法では、データに対してクエリの一部としてではなく、まとめ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対して最適です。あまり変更されないルールやオントロジーによって推論を実行します。

このような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、そのクエリをデータに対して実行します。

このようなマテリアライズのプロセスは時間がかかることがあり、格納に大量のメモリを必要とします。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。この種の推論は、データ、ルールセット、およびオントロジーが大きく変更されない場合に使用できます。

7.3 REST API による推論の使用

データベースのデフォルトルールセットは、SPARQL REST エンドポイント `POST:/v1/graphs/sparql` および `GET:/v1/graphs/sparql` を使用して SPARQL を実行するときに適用されます。

注： `sem:sparql` または `sem:sparql-update` に対するパラメータ（ルールセットなど）を SPARQL REST エンドポイントに指定することはできません。

`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>
```

REST エンドポイントと curl を使用すると（さらにデータベースのデフォルトルールセットが同じ場合）、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
```

詳細については、「REST API を使用したトリプルのクエリ」（132 ページ）、および『*REST Application Developer's Guide*』の「Querying Triples」を参照してください。

7.4 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。

7.4.1 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。

セマンティック API	説明
sem:store	sem:sparql のクエリ引数では、クエリで使用するトリプルのソースとして sem:store 値を利用できるようになりました。複数の sem:store 値を指定した場合は、すべてのソースからのトリプルがマージされてクエリされます。 sem:store 値には、1 つあるいは複数のオプションと 1 つの cts:query を含めることができます。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。
sem:in-memory-store	トリプルの集まりを表現する sem:store 値を返します。このトリプルの集まりは、引数として渡された sem:triple 値に由来します。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 値を返します。

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）よりも優先して sem:sparql と併用してください。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。

7.4.2 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。
`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。

7.4.3 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、「埋め込みトリプル」（68 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 によるトリプルの挿入、削除、および修正」（202 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。

- SPARQL Update の使用
- SPARQL Update によるグラフの操作
- グラフレベルのセキュリティ
- SPARQL Update によるデータの操作
- 変数のバインディング
- Query Console での SPARQL Update の使用
- XQuery または JavaScript での SPARQL Update の使用
- REST での SPARQL Update の使用

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」（184 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 JavaScript での SPARQL Update の使用」（185 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」（186 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 によるトリプルの挿入、削除、および修正」（202 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、『*Understanding and Using 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。

次に例を示します。

```
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。

次に例を示します。

```
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 操作には sem:sparql-update の $default-permissions パラメータは適用されません。

次に例を示します。

```
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ても、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。

次に例を示します。

```
MOVE <http://marklogic/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 $default-permissions 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ても、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)
```

次に例を示します。

```
ADD <http://marklogic/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を失敗させる場合は、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ても、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。『*Understanding and Using 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、『*Understanding and Using 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 が存在しない場合は、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は、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は、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ディングに基づき、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 は、INSERT（オプション）のない INSERT/DELETE です。INSERT WHERE は、DELETE（オプション）のない INSERT/DELETE 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」（42 ページ）、「sem:rdf-load」（50 ページ）、「グラフストアの処理」（53 ページ）、および「トリプルのロード」（34 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」（175 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 demov: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 操作は、WHERE 節で指定したクエリパターンのバインディングに基づき、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する目的で使用します。DELETE/INSERT WHERE を使用すると、マッチの対象とな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。

固有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 節なしで 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ディング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ディングは、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、従業員が 100 人未満の企業に関する営業データをグラフ <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 )
```

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 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。

次に例を示します。

```
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 変数のバインディング

変数のバインディングは、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ディングを作成するには、バインディングをマッピングする XQuery または JavaScript 関数を作成し、そのバインディングを 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、バインディングの集まりを構築し、そのバインディングと 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)
};

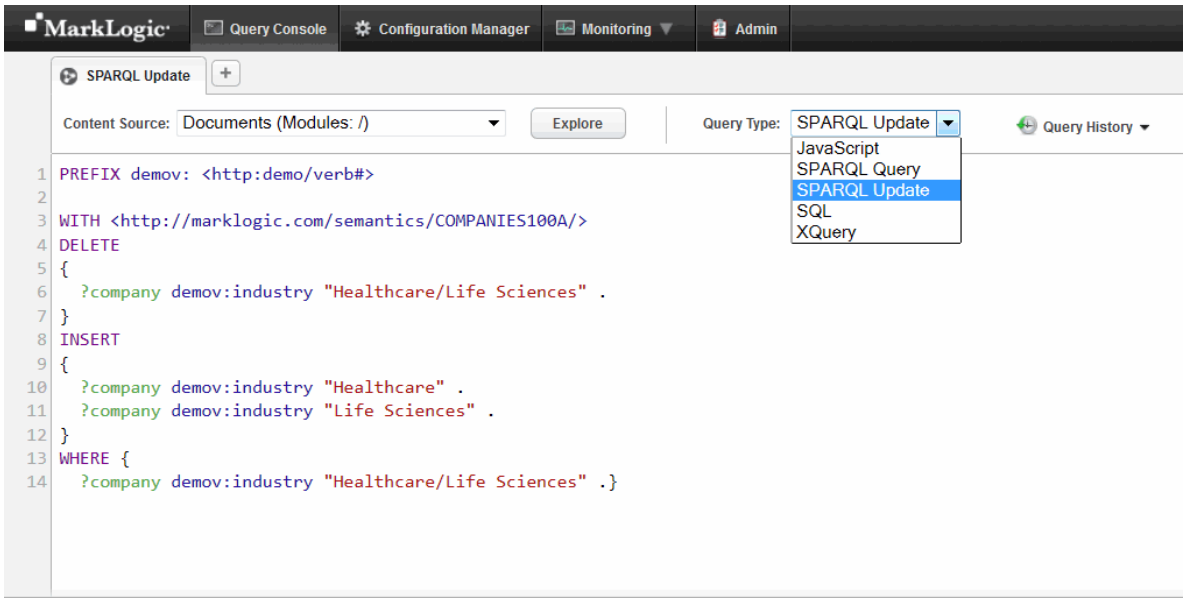
local:add-player-triples(417, "Doolittle", "Sean", "pitcher", 62,
"Athletics"),
local:add-player-triples(215, "Abad", "Fernando", "pitcher", 56,
"Athletics"),
local:add-player-triples(109, "Kazmir", "Scott", "pitcher", 26,
"Athletics"),
```

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。変数のバインディングは、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ディングを使用する例については、「変数に対するバインディングの使用」（121 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 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、『*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『*Node.js Application Developer's Guide*』の「[セマンティックトリプルのクエリ](#)」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)
}
```

8.8 REST での SPARQL Update の使用

このセクションでは、SPARQL Update を使用して、SPARQL エンドポイントからグラフやトリプルデータを管理する方法について説明します。エンドポイントとは、W3C SPARQL プロトコルの実装です。このエンドポイントの URL には、次のようなアドレスが割り当てられます。

```
http://host:port/v1/graphs/sparql?
```

`GET:/v1/graphs/sparql` または `POST:/v1/graphs/sparql` を使用することで、REST で SPARQL Update を使用できます。

- REST で SPARQL Update を使用する方法
- POST を使用した 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

RDF トリプルで REST 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。

8.8.1 REST で SPARQL Update を使用する方法

SPARQL エンドポイントには、3 種類の方法で更新を送信できます。

```
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".
}
```

リクエストのボディに次の形式でクエリを埋め込み、URL に対して `POST:/v1/graphs/sparql` リクエストを実行できます。

```
http://host:port/v1/graphs/sparql

content-type:application/sparql-update
```

または、次のような更新パラメータにクエリを埋め込む形式で、POST:/v1/graphs/sparql リクエストを実行できます。

```
http://host:port/v1/graphs/sparql

content-type:application/x-www-form-urlencoded
```

また、次の形式で更新パラメータにクエリを埋め込み、GET:/v1/graphs/sparql リクエストを実行することもできます。

```
http://host:port/v1/graphs/sparql?update=URLENCODEDQUERY
```

オプションは、GET:/v1/graphs/sparql?query= と POST:/v1/graphs/sparql です。POST を使用する場合は、リクエストのボディに SPARQL を直接配置することも、x-www-form-urlencoded を使用して、更新形式パラメータを URL エンコードされた SPARQL Update に設定することもできます。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。

REST 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」（133 ページ）を参照してください。POST および GET の詳細については、「サポートされている動詞」（139 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 および GET:/v1/graphs/sparql を参照してください。

8.8.2 POST を使用した 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。HTTP リクエストのコンテンツタイプヘッダは application/sparql-update に設定します。

次に例を示します。

```
# Windows ユーザーは次を参照のこと。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://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".}" \
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「\」または「^」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。

SPARQL Update はファイルに含めることもできます。この SPARQL Update は、booktitle.sparql という名前のファイルに記載されています。

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
INSERT DATA
{
<http://example/book1> dc:title "book title" ;
dc:creator "author name" .
}
```

このファイルが含まれるリクエストは、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 URI の HTTP クエリ文字列パラメータとして、オプションの using-graph-uri、using-named-graph-uri、および role-capability パラメータを含めることができます。default-permissions パラメータは、ロールと権限をカンマ区切りの文字列で指定してください。

```
default-permissions=admin,update
default-permissions=admin,execute
```

この例では、SPARQL Update および URL エンコードされたパラメータを指定した 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 に挿入します。パラメータは、アンパサンド（&）文字で区切ります。また、任意の順序で含めることができます。

```
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/sparql-update" \
-H "Accept:application/sparql-results+xml" \
'http://localhost:8000/v1/graphs/sparql?using-named-graph-uri=C1 \
&default-permissions=admin,update&default-permissions=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。

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経由でリクエストの成功または障害を示します（200 または 400）。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。

8.8.3 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 エンコードすることで、HTTP POST メソッド経由で更新プロトコルの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。

> **注：** REST を使用して、パラメータを URL の一部として渡すときは、常に URL を URL エンコードします。

この例では curl を使用し、URL エンコーディングが含まれる SPARQL Update で POST を使用します。

```
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"} }' \
http://localhost:8321/v1/graphs/sparql?
```

次の例は、SPARQL Update とパラメータを使用する POST の例を示したものです。

```
curl --anyauth --user admin:admin -i -X POST \
--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 default-permissions=rest-writer,read \
--data-urlencode default-permissions=rest-writer,update \
--data-urlencode default-permissions=rest-reader,read \
--data-urlencode default-permissions=rest-reader,update \
-H "Content-type:application/x-www-form-urlencoded" \
-H "Accept:application/sparql-results+xml" \
http://localhost:8321/v1/graphs/sparql
```

9.0 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 API、および Java API が、各種のクエリスタイルを使用して MarkLogic のセマンティック機能をサポートします（このガイドのトリプルのロード、セマンティッククエリ、および XQuery によるトリプルの挿入、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

9.1 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 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。

9.1.1 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。

9.1.2 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、『*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、および『*MarkLogic XQuery and XSLT Function Reference*』を参照してください。

9.2 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。次に例を示します。

```
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」（125 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。

9.3 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">
  <a href="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>
```

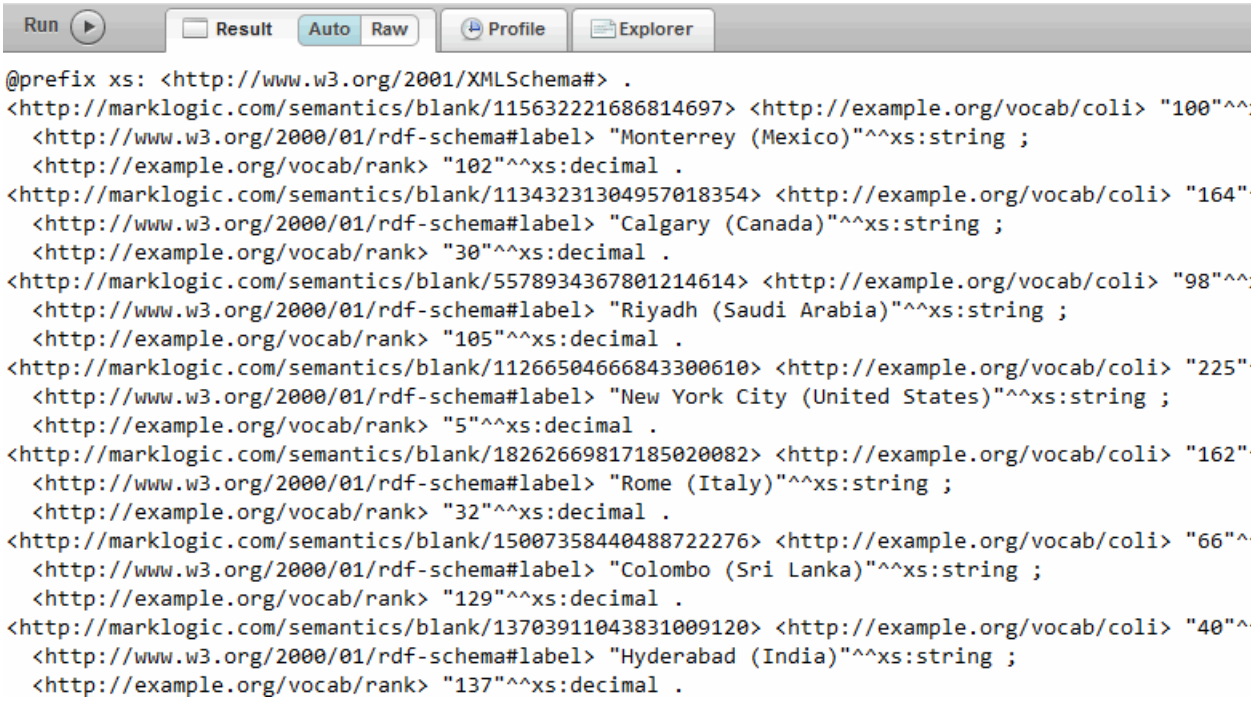
この例では、`sem:rdf-builder` 関数を使用して HTML コンテンツからトリプルを抽出します。この関数は、HTML コードがすでに整形されていて、有用な分類ノード（`@class`）があるということを利用します。

```
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。



9.4 トリプルのパース

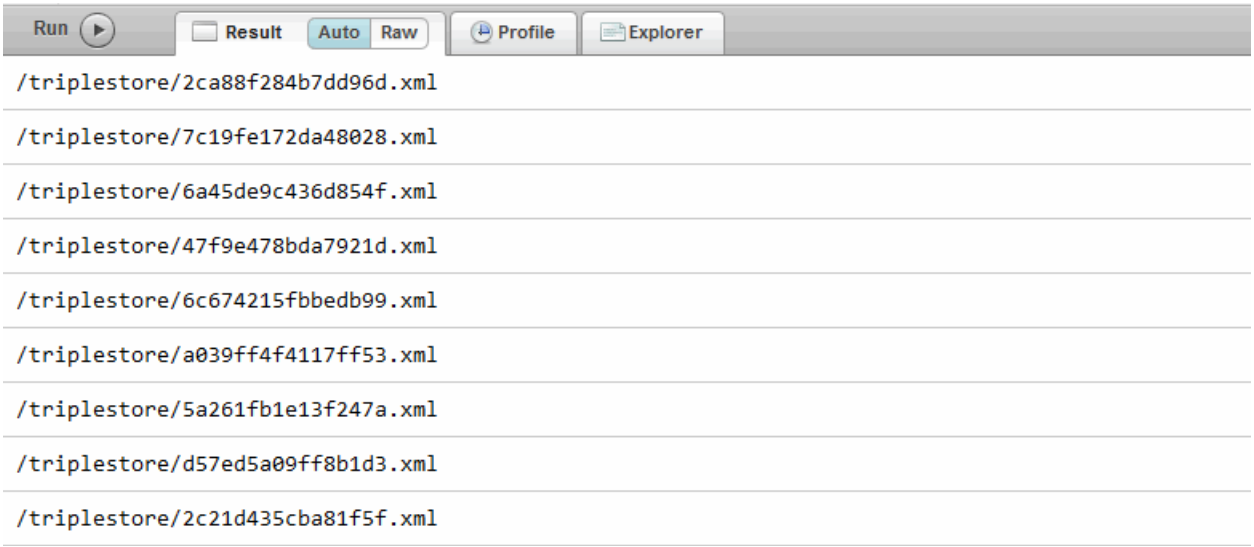
この例では前の例を拡張し、`sem:rdf-insert` 関数を使用してパースしたトリプルをデータベースに挿入します。

```
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。

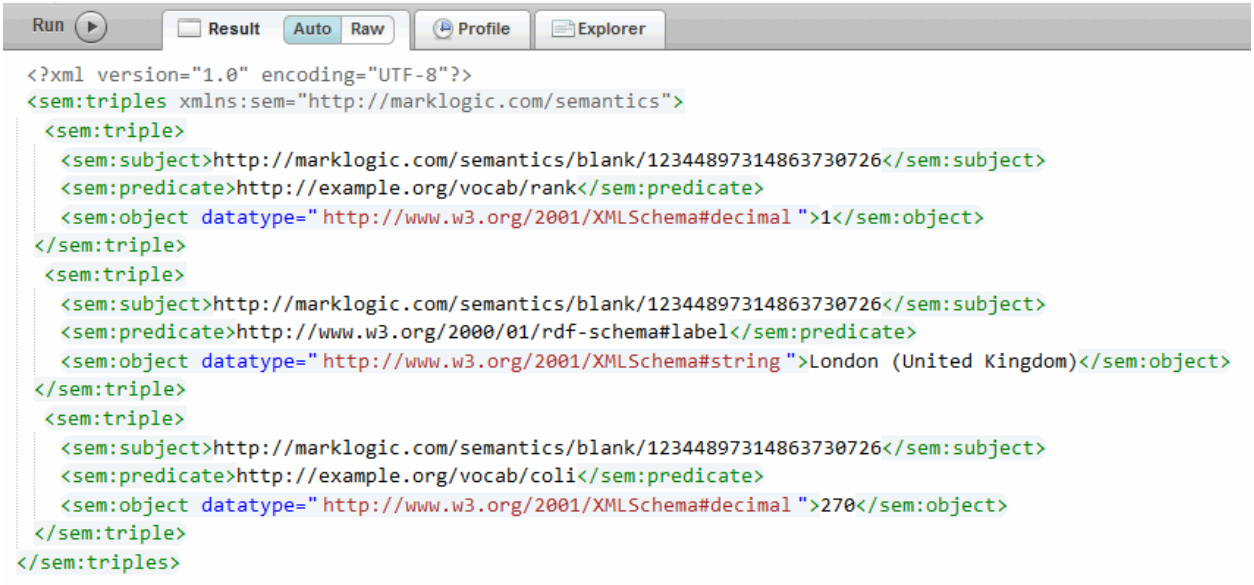


注： XQuery でトリプルを挿入およびパースする詳細については、「XQuery を使用したトリプルのロード」（49 ページ）を参照してください。

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：概要と使い方」（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。

9.5 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

9.5.1 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 .
```

9.5.2 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 .
...
```

10.0 XQuery によるトリプルの挿入、削除、および修正

トリプルは、XQuery または JavaScript で MarkLogic の xdmp ビルトインを使用して修正できます。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」（169 ページ）を参照してください。

「非管理対象」トリプル（*要素ノード*が sem:triples）は、XQuery または JavaScript と xdmp ビルトインを使用した場合のみ修正できます。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。

- トリプルの更新
- トリプルの削除

10.1 トリプルの更新

XQuery または JavaScript の関数を使用し、INSERT/DELETE を使用してノードを置換することにより、データベース内の既存のトリプルを更新できます。管理対象トリプルでは、不正確なデータを修正するために、sem:database-nodes（JavaScript の場合は sem.databaseNode）および xdmp:node-replace（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:subject>http://dbpedia.org/resource/Andr%C3%A9-Marie_Amp%C3%A8r
e
  </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:triples>
```

この例では、sem:rdf-builder 関数がトリプルを構築して sem:database-nodes 関数に渡します。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(., $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 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://marklogic.com/
  semantics","predicate")[0]);
replace.addNode(node.getElementsByTagNameNS("http://marklogic.com/
  semantics","object")[0]);
replace.endElement();
for (var node of nodes) {
xdmp.nodeReplace(node, replace);
};
```

複数のトリプルを更新する場合は、XQuery または JavaScript（管理対象トリプルの場合は SPARQL Update）を使用し、マッチングトリプルを調べてノードを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。

10.2 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 JavaScript によるトリプルの削除
- REST API を使用したトリプルの削除

10.2.1 XQuery または JavaScript によるトリプルの削除

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

10.2.1.1 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。

次に例を示します。

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

sem:graph-delete(sem:iri("mynamedgraph"))
```

JavaScript の場合のコマンドは、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。

10.2.1.2 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。

次に例を示します。

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

let $triples :=
cts:triples(sem:iri("http://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 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(.))
```

10.2.1.3 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。

次に例を示します。

```
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 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")
```

10.2.2 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 ユーザーは次を参照のこと。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 ユーザーは次を参照のこと。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。

次に例を示します。

```
# Windows ユーザーは次を参照のこと。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。