Este documento es una traducción al castellano de la nota del grupo de trabajo del W3C "SKOS Simple Knowledge Organization System Primer", publicada el 18 de agosto de 2009. La presente traducción se concluyó el 20 de noviembre de 2009.

La versión original en inglés es el único documento válido y se encuentra en:
http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/
Puede ver la última versión del documento en inglés en: http://www.w3.org/TR/rdf-sparql-query/

Se ha tratado de respetar al máximo el contenido del documento original en inglés, adaptando la expresión al español para ayudar a una mejor comprensión del mismo. Por tanto, esta traducción puede contener errores, en ningún caso achacables a sus autores originales. Cualquier sugerencia de corrección, duda o comentario sobre la misma puede realizarse dirigiéndose a alguno de sus autores: Juan Antonio Pastor Sánchez y Pedro Manuel Díaz Ortuño.

SPARQL Lenguaje de consulta para RDF

Recomendación del W3C de 15 de enero de 2008

Versión original (en Inglés):
http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/
Última versión:
http://www.w3.org/TR/rdf-sparql-query/
Versión anterior:
http://www.w3.org/TR/2007/PR-rdf-sparql-query-20071112/
Editores:
Eric Prud'hommeaux, W3C <eric@w3.org>
Andy Seaborne, Hewlett-Packard Laboratories, Bristol <andy.seaborne@hp.com>
Traductores (versión en castellano):
Juan Antonio Pastor Sánchez, Universidad de Murcia
Pedro Manuel Díaz Ortuño, Universidad de Murcia

Consulte la sección de **erratas** de este documento, en la que se pueden incluir algunas correciones normativas.

Vea también la sección de **traducciones**.

Resumen

RDF es un formato de datos para grafos dirigidos y etiquetados para representar la información en la Web. Esta especificación define la sintaxis y la semántica del lenguaje de consulta SPARQL para RDF. SPARQL se puede utilizar para expresar consultas que permiten interrogar diversas fuentes de datos, si los datos se almacenan de forma nativa como RDF o son definidos mediante vistas RDF a través de algún sistema middleware. SPARQL contiene las capacidades para la consulta de los patrones obligatorios y opcionales de grafo, junto con sus conjunciones y disyunciones. SPARQL también soporta la ampliación o restricciones del ámbito de las consultas indicando los grafos sobre los que se opera. Los resultados de las consultas SPARQL pueden ser conjuntos de resultados o grafos RDF.

Estado de este documento

Esta sección describe el estado de este documento en el momento de su publicación. Otros documentos pueden reemplazar este documento. Una lista delas publicaciones vigentes del W3C y la última revisión de este informe técnico se puede encontrar en el índice de informes técnicos del W3C en http://www.w3.org/TR/.

Esta es una Recomendación del W3C.

Este documento ha sido revisado por miembros del W3C, por desarrolladores de software y por otros grupos del W3C y partes interesadas, y ha sido remitido por el Director como una Recomendación del W3C. Se trata de un documento estable y puede ser utilizado como material de referencia o citado por otro documento. El papel del W3C en la elaboración de la Recomendación es atraer la atención sobre la especificación y promover un amplio despliegue de la misma. Todo ello mejora la funcionalidad y la interoperabilidad de la Web.

Los comentarios sobre este documento pueden ser enviados a public-rdf-dawg-comments@w3.org, una lista de distribución de correo electrónico. Las preguntas y comentarios sobre SPARQL que no estén relacionadas con esta especificación, incluyendo extensiones y características, pueden discutirse en la lista de distribución public-sparql-dev@w3.org, (archivo público).

Este documento fue elaborado por el Grupo de Trabajo sobre el Acceso a Datos RDF, que es parte de la Actividad para la Web Semántica del W3C. La primera versión de este documento fue un borrador de trabajo del 12 de octubre de 2004, y el grupo de trabajo ha considerado los problemas y comentarios recibidos desde entonces. Dos cambios han sido realizados y registrados desde la publicación de la Recomendación de noviembre de 2007.

El grupo de trabajo, en su Informe sobre la Implementación del Lenguaje de Consulta SPARQL para RDF desmuestra que se han alcanzado los objetivos para implementaciones interoperables, establecidos en la Recomendación Candidata de junio de 2007.

El grupo de trabajo ha pospuesto 12 problemas, incluyendo las funciones de agregación, y el lenguaje de actualización (mantenimiento) de datos.

Este documento fue elaborado por un grupo que opera en el marco de la Política de Patentes del W3C del 5 de febrero 2004. El W3C mantiene una lista pública de cualquier patente divulgada realizada en coordinación con la difusión de los resultados del grupo de trabajo; dicha página también incluye instrucciones para la divulgación de una patente. Una persona que tenga conocimiento actual de una patente que considere que contiene la(s) reivindicación(es) esencial(es) debe revelar la información de conformidad con el artículo 6 de la Política de Patentes del W3C.

Tabla de Contenidos

Anexos

---

1 Introducción

RDF es un formato de datos para grafos dirigidos etiquetados, que permite representar información en la Web. Normalmente, RDF se utiliza, entre otros

usos, para representar información personal, redes sociales, metadatos sobre objetos digitales, así como para proporcionar un medio para la integración de fuentes de información dispares. Esta especificación define la sintaxis y la semántica de SPARQL, un lenguaje de consulta para RDF.

El lenguaje de consulta SPARQL para RDF está diseñado para cumplir con los casos de uso y necesidades identificadas por el Grupo de Trabajo de Acceso a Datos RDF incluidos en el documento Casos de Uso y Requisitos de acceso a datos RDF [UCNR].

El lenguaje de consulta SPARQL está estrechamente relacionado con las siguientes especificaciones:

- La especificación del Protocolo SPARQL para RDF [SPROT] que define el protocolo remoto para enviar consultas SPARQL y recibir los resultados.
- La especificación del Formato XML de los resultados de consultas SPARQL [RESULTS] que define un formato de documento XML para representar los resultados de las consultas SELECT y ASK de SPARQL.

1.1 Estructura del documento

A menos que se indique lo contrario en el epígrafe correspondiente, todas las secciones y anexos de este documento son normativas.

En esta sección del documento, la sección 1, se realiza una introducción a la especificación de lenguaje de consulta SPARQL. Se presenta la organización de este documento de esta especificación, así como las convenciones utilizadas en la misma.

La sección 2 es una introducción al lenguaje de consulta SPARQL en sí, a través de una serie de ejemplos y resultados de consultas. La sección 3 continúa con la introducción a este lenguaje con más ejemplos que demuestran la capacidad de SPARQL para expresar las restricciones sobre los términos RDF que aparecen en los resultados de una consulta.

La sección 4 presenta los detalles de la sintaxis del lenguaje de consulta SPARQL. Se trata de un acompañamiento a la gramática completa del lenguaje y define cómo las construcciones gramaticales representan IRIs, nodos en blanco, literales y variables. Esta sección también define el significado de varias construcciones gramaticales que sirven como simplificación sintáctica de expresiones más verbosas.

La sección 5 muestra los patrones de grafo básicos y de grupo, que son los componentes elementales a partir de los que se construyen patrones de consultas SPARQL más complejas. Las secciones 6, 7 y 8 presentan las construcciones que combinan patrones de grafo SPARQL para crear patrones de grafo más grandes. En particular, la sección 6 muestra la capacidad de hacer partes de una consulta opcional, la sección 7 presenta la capacidad de expresar la disyunción de los patrones de grafos alternativos, y la sección 8 introduce las posibilidades de limitar las porciones de una consulta a grafos fuente concretos. La sección 8 también presenta el mecanismo de SPARQL para la definición de los grafos fuente para una consulta.

La sección 9 define los constructores que afectan a las soluciones de una consulta para ordenar, extraer, proyectar, limitar y eliminar duplicados de una secuencia de soluciones.

La sección 10 define los cuatro tipos de consultas SPARQL que producen resultados de diferentes formas.

La sección 11 define el marco de comprobación de valores extensibles de SPARQL. También muestra las funciones y operadores que pueden usarse para limitar los valores que aparecen en los resultados de una consulta.

La sección 12 es una definición formal de la evluación de patrones de grafo SPARQL y de los modificadores de soluciones.

El apéndice A contiene la definición normativa de la sintaxis de SPARQL, en forma de una gramática expresada en notación EBNF.

1.2 Convenciones del documento

1.2.1 Espacios de nombres

En este documento, a menos que se indique lo contrario se asumen los siguientes vínculos de prefijos de espacios de nombres:

Prefijo	IRI
rdf:	http://www.w3.org/1999/02/22-rdf-syntax-ns#
rdfs:	http://www.w3.org/2000/01/rdf-schema#
xsd:	http://www.w3.org/2001/XMLSchema#
fn:	http://www.w3.org/2005/xpath-functions#

1.2.2 Descripciones de datos

Este documento utiliza el formato de datos Turtle [TURTLE] para mostrar de forma explícita cada tripleta. Turtle permite abreviar IRIs mediante el uso de prefijos:

```
@prefix dc:   <http://purl.org/dc/elements/1.1/> .
@prefix :     <http://example.org/book/> .
:book1  dc:title  "SPARQL Tutorial" .
```

1.2.3 Descripciones de resultados

Los conjuntos de resultados se muestran en forma de tablas.

x	y	z
"Alice"	<http://example/a>	

Un 'vínculo' es un par (variable, término RDF). En este conjuto de resultados, hay tres variables: x, y, z (mostradas en los encabezados de las columnas).

Cada solución se muestra como una file en el cuerpo de la tabla. En el ejemplo hay una única solución, en el que la variable `x` esta vinculada a `"Alice"`, la variable `y` está vinculada con `<http://example/a>`, y la variable `z` no está vinculadas a ningún término RDF. En una solución, la variables no tienen por qué estar vinculadas obligatoriamente.

1.2.4 Terminología

El lenguaje SPARQL incluye IRIs, un subconjunto de Referencias URI RDF que omiten los espacios. Debe tenerse en cuenta que en las consultas SPARQL todas las referencias a IRIs son absolutas; pueden o no incluir un identificador de fragmento [RFC3987, sección 3.1]. Las referencias IRI include a las referencias URI [RFC3986] y URL. En la sintaxis de SPARQL, las formas abreviadas (referencias IRIs relativas y nombres con prefijos) se resuelven para producir IRIs absolutas.

Los siguientes términos son definidos en el documento sobre Conceptos y Sintaxis abstracta de RDF [CONCEPTS] y empleados en SPARL:

- IRI (corresponden al término "`RDF URI reference`" en el documento sobre Conceptos y Sintaxis abstracta de RDF)
- literal
- forma léxica
- literal plano
- etiqueta de lenguaje
- literal tipado
- tipo de dato IRI (corresponds al término "`datatype URI`" en el documento sobre Conceptos y Sintaxis abstracta de RDF)
- nodo en blanco

2 Realizar consultas simples (Informativo)

La mayoría de las formas de consulta en SPARQL contienen un conjunto de patrones de tripleta (triple patterns) denominadas *patrón de grafo básico*. Los patrones de tripleta son similares a las tripletas RDF, excepto que cada sujeto, predicado y objeto puede ser una variable. Un patrón de grafo básico concuerda con un subgrafo de datos RDF cuando los términos RDF (RDF terms) de dicho subgrafo pueden ser sustituidos por las variables y el resultado es un grafo RDF equivalente al subgrafo en cuestión.

2.1 Escribir una consulta simple

El siguiente ejemplo muestra una consulta SPARQL para encontrar el título de un libro en el grafo de datos dado. La consulta consta de dos partes: la cláusula `SELECT` identifica las variables que aparecen en los resultados de la consulta, y la cláusula `WHERE` proporciona el patrón de grafo básico para la concordancia con el gráfo de datos. El patrón de grafo básico de este ejemplo consiste en un único patrón de tripleta con una sola variable (`?title`) en la posición del objeto.

Datos:

```
<http://example.org/book/book1> <http://purl.org/dc/elements/1.1/tit
```

Consulta:

```
SELECT ?title
WHERE
{
  <http://example.org/book/book1> <http://purl.org/dc/elements/1.1/t
}
```

Esta consulta, efectuada sobre los datos indicados anteriormente, tiene una solución:

Resultado de la consulta:

title
"SPARQL Tutorial"

2.2 Concordancias múltiples

El resultado de una consulta es una secuencia de solución, correspondiente a las distintas posibilidades de concordancia del patrón del grafo de la consulta con los datos. Puede haber ninguna, una o múltiples soluciones a una consulta.

Datos:

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name   "Johnny Lee Outlaw" .
_:a  foaf:mbox   <mailto:jlow@example.com> .
_:b  foaf:name   "Peter Goodguy" .
_:b  foaf:mbox   <mailto:peter@example.org> .
_:c  foaf:mbox   <mailto:carol@example.org> .
```

Consulta:

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE
  { ?x foaf:name ?name .
    ?x foaf:mbox ?mbox }
```

Resultados de la consulta:

name	mbox
"Johnny Lee Outlaw"	<mailto:jlow@example.com>
"Peter Goodguy"	<mailto:peter@example.org>

Cada solución proporciona una manera de vincular las variables seleccionadas a los términos RDF para los que el patrón de consulta concuerda con los datos. El conjunto de resultados proporciona todas las posibles soluciones. En el ejemplo anterior, los dos subconjuntos de datos mostrados a continuación proporcionan ambas concordancias.

```
_:a foaf:name  "Johnny Lee Outlaw" .
_:a foaf:box   <mailto:jlow@example.com> .
```

```
_:b foaf:name  "Peter Goodguy" .
_:b foaf:box   <mailto:peter@example.org> .
```

Esta es una concordancia de patrón de grafo básico; todas las variables utilizadas en el patrón de consulta deben estar vinculadas en cada solución.

2.3 Concordancias de literales RDF

Los datos siguientes contienen tres literales RDF:

```
@prefix dt:   <http://example.org/datatype#> .
@prefix ns:   <http://example.org/ns#> .
@prefix :     <http://example.org/ns#> .
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .

:x   ns:p     "cat"@en .
:y   ns:p     "42"^^xsd:integer .
:z   ns:p     "abc"^^dt:specialDatatype .
```

Debe observarse que en Turtle `"cat"@en` es un literal RDF con una forma léxica "cat" y un idioma `en`; `"42"^^xsd:integer` es un literal tipado con el tipo de datos `http://www.w3.org/2001/XMLSchema#integer`; y `"abc"^^dt:specialDatatype` es un literal tipado con el tipo de datos `http://example.org/datatype#specialDatatype`.

Estos datos RDF constituyen el grafo de datos para las consultas de ejemplo de las secciones 2.3.1–2.3.3.

2.3.1 Concordancias de literales de idioma

Las etiquetas de idioma en SPARQL se expresan usando `@` y la etiqueta de idioma, como se define en el documento Best Common Practice 47 [BCP47].

La siguiente consulta no tiene solución porque `"cat"` no es el mismo literal RDF que `"cat"@en`:

```
SELECT ?v WHERE { ?v ?p "cat" }
```

v

sin embargo, la siguiente consulta encontrará una solución, donde la variable `v` se vincula a `:x` porque la etiqueta de idioma se especifica y concuerda con los datos del grafo:

```
SELECT ?v WHERE { ?v ?p "cat"@en }
```

v
<http://example.org/ns#x>

2.3.2 Concondancias de literales con tipos numéricos

Los enteros de una consulta SPARQL indican un literal RDF tipado con el tipo de datos `xsd:integer`. Por ejemplo: `42` es una forma abreviada de `"42"^^<http://www.w3.org/2001/XMLSchema#integer>`.

El patrón en la siguiente consulta tiene una solución con la variable `v` vinculada a `:y`.

```
SELECT ?v WHERE { ?v ?p 42 }
```

v
<http://example.org/ns#y>

La sección 4.1.2 define formas abreviadas de SPARQL para `xsd:float` y `xsd:double`.

2.3.3 Concordancias de literales con tipos de datos arbitrarios

La siguiente consulta tiene una solución con la variable `v` ligada a `:z`. El procesador de consultas no tiene porqué tener conocimiento de los valores en el espacio del tipo de datos. Debido a la correspondencia tanto de la forma léxica como del tipo de datos del IRI, la consulta encuentra una concordancia con el literal indicado.

```
SELECT ?v WHERE { ?v ?p "abc"^^<http://example.org/datatype#specialD
```

v
<http://example.org/ns#z>

2.4 Etiquetas de nodos en blanco en los resultados de las consultas

Los resultados de una consulta pueden contener nodos en blanco. Los nodos en blanco, en los ejemplos de conjuntos de resultados de este documento, se escriben con la forma "_:" seguida de una etiqueta del nodo en blanco.

Las etiquetas de los nodos en blanco se aplican en el ámbito de un conjunto de resultados (como se define en la especificación "Formato XML de los

resultados de consultas SPARQL") o en los grafos resultantes al utilizar la forma de consulta `CONSTRUCT`. El uso de la misma etiqueta en un conjunto de resultados indica el mismo nodo en blanco.

Datos:

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name   "Alice" .
_:b  foaf:name   "Bob" .
```

Consulta:

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
SELECT ?x ?name
WHERE  { ?x foaf:name ?name }
```

x	name
_:c	"Alice"
_:d	"Bob"

Los resultados anteriores también podrían darse con diferentes etiquetas de nodos en blanco, debido a que las etiquetas en los resultados indican si los términos RDF de las soluciones son los mismos o no.

x	name
_:r	"Alice"
_:s	"Bob"

Estos dos resultados tienen la misma información: los nodos en blanco utilizados para la concordancia con la consulta son diferentes en ambas soluciones. No tiene que haber ninguna relación entre una etiqueta `_:a` en el conjunto de resultados y un nodo en blanco en el grafo de datos con la misma etiqueta.

Un programador de aplicaciones no debe esperar que las etiquetas de un nodo en blanco de una consulta hagan referencia a un nodo en blanco específico en los datos.

2.5 Construcción de grafos RDF

SPARQL tiene varias formas de consulta. La forma de consulta `SELECT` devuelve vínculos de variables. La forma `CONSTRUCT` devuelve un grafo RDF. El grafo se construye sobre una plantilla que se usa para generar tripletas RDF basadas en los resultados de la concordancia con el patrón de grafo de la consulta.

Datos:

```
@prefix org:    <http://example.com/ns#> .

_:a  org:employeeName   "Alice" .
_:a  org:employeeId     12345 .

_:b  org:employeeName   "Bob" .
_:b  org:employeeId     67890 .
```

Consulta:

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

CONSTRUCT { ?x foaf:name ?name }
WHERE  { ?x org:employeeName ?name }
```

Resultados:

```
@prefix org: <http://example.com/ns#> .

_:x foaf:name "Alice" .
_:y foaf:name "Bob" .
```

que pueden serializarse con RDF/XML del siguiente modo:

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:foaf="http://xmlns.com/foaf/0.1/"
    >
  <rdf:Description>
    <foaf:name>Alice</foaf:name>
  </rdf:Description>
  <rdf:Description>
    <foaf:name>Bob</foaf:name>
  </rdf:Description>
</rdf:RDF>
```

3 Restricciones de términos RDF (Informativo)

La concordancia con un patrón de grafo produce una secuencia de solución, donde cada solución tiene un conjunto de vínculos de variables con términos RDF. Los filtros `(FILTERS)` SPARQL restringen las soluciones a aquellas para las que la expresión del filtro se evalua como verdadera `(TRUE)`.

Esta sección proporciona una introducción informal a los filtros de SPARQL; su semántica se define en la sección 11. Comprobación de valores. Los ejemplos de esta sección utilizan un mismo grafo de entrada:

Datos:

```
@prefix dc:   <http://purl.org/dc/elements/1.1/> .
@prefix :     <http://example.org/book/> .
@prefix ns:   <http://example.org/ns#> .

:book1  dc:title  "SPARQL Tutorial" .
:book1  ns:price  42 .
:book2  dc:title  "The Semántica Web" .
:book2  ns:price  23 .
```

3.1 Restricciones sobre valores en cadenas de texto

Las funciones de filtro de SPARQL, como `regex` pueden comprobar el valor de literales RDF. `regex` busca concordancias entre literales planos sin etiqueta de lenguaje. `regex` puede usarse para concordancias con formas léxicas de otros literales utilizando la función str.

Consulta:

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
SELECT  ?title
WHERE   { ?x dc:title ?title
          FILTER regex(?title, "^SPARQL")
        }
```

Resultado:

title
"SPARQL Tutorial"

Las concordancias entre expresiones regulares puede realizarse de manera que sea insensibles a las mayúsculas mediante el indicador "`i`".

Consulta:

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
SELECT  ?title
WHERE   { ?x dc:title ?title
          FILTER regex(?title, "web", "i" )
        }
```

Resultado:

title
"The Semantic Web"

El lenguaje de expresiones regulares está definido en el documento Funciones y Operadores de XQuery 1.0 y XPath 2.0 y se basa en las Expresiones regulares de XML Schema.

3.2 Restricciones sobre valores numéricos

Los filtros de SPARQL pueden realizar restricciones sobre expresiones aritméticas.

Consulta:

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
PREFIX  ns:  <http://example.org/ns#>
SELECT  ?title ?price
WHERE   { ?x ns:price ?price .
          FILTER (?price < 30.5)
          ?x dc:title ?title . }
```

Resultado:

title	price
"The Semantic Web"	23

Al limitar el valor de la variable `price`, únicamente `:book2` satisface las condiciones de la consulta, ya que solo `:book2` tiene un precio menor de `30.5`, tal como se indica en la condición del filtro.

3.3 Otras restricciones de términos

Además de los tipos de datos numéricos, SPARQL soporta los tipos `xsd:string`, `xsd:boolean` y `xsd:dateTime` (ver 11.1 Tipos de datos de los operandos). En 11.3 Correspondencia de operadores se enumera un conjunto de funciones de comprobación, incluyendo `BOUND`, `isLITERAL` y `langMATCHES` y acceso, incluyendo `STR`, `LANG` y `DATATYPE`. En 11.5 Funciones constructor se incluye una relación de funciones constructor de XML Schema para convertir valores de un tipo de datos a otro.

4 Sintaxis SPARQL

Esta sección cubre la sintaxis utilizada por SPARQL para términos RDF (RDF terms) y patrones de tripletas (triple patterns). La gramática completa figura en el Anexo A (appendix A).

4.1 Sintaxis de los Términos RDF

4.1.1 Sintaxis para IRIs

La producción IRIref designa el conjunto de IRIs [RFC3987]; los IRIs son una generalización de los URIs [RFC3986] y son totalmente compatibles con URIs y URLs. La producción PrefixedName designa un nombre prefijado. La correspondencia de un nombre prefijado a una IRI se describe a continuación. Las referencias IRI (IRIs absolutas o relativas) son designadas por la producción IRI_REF, en la que los caracteres delimitadores "<" y ">" no forman parte de las referencias IRI. Los IRIs relativos se corresponden con la referencia irelative-ref de la Sección 2.2 ABNF para referencias IRI e IRIs en [RFC3987] y se resuelven a IRIs como se describe a continuación.

Reglas de gramática:

[67]	*IRIref*	::=	IRI_REF | PrefixedName
[68]	*PrefixedName*		PNAME_LN | PNAME_NS
[69]	*BlankNode*	::=	BLANK_NODE_LABEL | ANON
[70]	*IRI_REF*	::=	'<' ([^<>"{}|^`\]-[#x00-#x20])* '>'
[71]	*PNAME_NS*	::=	PN_PREFIX? ':'
[72]	*PNAME_LN*	::=	PNAME_NSPN_LOCAL

El conjunto de términos RDF definidos en la especificación Conceptos y Sintaxis Abstracta de RDF [CONCEPT] incluye referencias RDF URI mientras que los términos de SPARQL contienen IRIs. Las referencias RDF URI que contienen los caracteres "<", ">", '"' (comilla doble), espacio, "{", "}", "|", "\", "^", y "`" no son IRIs. No está definido el comportamiento de una consulta SPARQL frente a las sentencias RDF compuestas de tales referencias URI.

Nombres prefijados.

La palabra clave `PREFIX` asocia una etiqueta de prefijo con un IRI. Un nombre con prefijo está compuesto de una etiqueta de prefijo y una parte local, separados por dos puntos ":". Un nombre con prefijo se correponde (*mapped*) con un IRI mediante la concatenación del IRI asociado con el prefijo y la parte local. La etiqueta de prefijo y la parte local pueden estar vacías. Tenga en cuenta que los nombres locales SPARQL admiten dígitos iniciales, a diferencia de los nombres locales XML.

IRIs Relativos

Los IRIs relativos se combinan con IRIs base conforme a la RFC Identificador de Recurso Uniforme(URI): Sintaxis Genérica [RFC3986], utilizando sólo el algoritmo básico de la Sección 5.2. No se llevan a cabo ni la normalización basada en sintaxis ni la normalización basada en esquema (descritas en las secciones 6.2.2 y 6.2.3 de la RFC 3986). Los caracteres adicionalmente permitidos en las referencias IRI son tratados de la misma manera en la que son tratados los caracteres no reservados en las referencias URI, tal y como se indica en el apartado 6.5 de Identificadores de Recursos Internacionalizados (IRIs) [RFC3987].

La palabra clave `BASE` define la dirección IRI de base para resolver las direcciones IRIs relativas, según la Sección 5.1.1, "URI de base incorporadas al contenido" de la RFC 3986. La Sección 5.1.2, "URI de base de la Entidad encapsulante", define cómo la IRI de base puede provenir de un documento encapsulante, por ejemplo una envoltura SOAP con una directiva xml:base o un documento MIME multiparte con un encabezado Content-Location. La "URI de recuperación" identificada en el punto 5.1.3, "URI de base de la URI de recuperación", es la URL desde la que se recuperó una consulta SPARQL en particular. Si ninguna de las anteriores especifica la URI de base, se utiliza la URI de base por defecto (sección 5.1.4, "URI de base por defecto").

Los siguientes fragmentos son algunas de las diferentes formas deescribir la misma dirección IRI:

```
<http://example.org/book/book1>
```

```
BASE <http://example.org/book/>
<book1>
```

```
PREFIX book: <http://example.org/book/>
book:book1
```

4.1.2 Sintaxis para Literales

La sintaxis general para literales es una cadena de caracteres (entre comillas dobles, `"..."`, o comillas simples, `'...'`), con una etiqueta de idioma (antecedida por `@`) o un tipo de datos IRI o nombre prefijado (antecedido por `^^`) opcionales.

Por comodidad, los enteros pueden escribirse directamente (sin comillas y tipo de datos IRI explícito) y se interpretan como literales del tipo de datos `xsd:integer`; los decimales con '.' en el número pero sin exponente se interpretan como `xsd:decimal`; y los números con exponente se interpretan como `xsd:double`. Los valores del tipo `xsd:boolean` pueden también escribirse como `true` o `false`.

Para facilitar la escritura de valores literales que contengan comillas o que sean largos y contengan caracteres de salto de línea, SPARQL proporciona una construcción adicional con los literales entrecomillados entre tres comillas simples o dobles.

Ejemplos de sintaxis de literales SPARQL:

- `"chat"`
- `'chat'@fr` con la etiqueta de idioma "fr"
- `"xyz"^^<http://example.org/ns/userDatatype>`
- `"abc"^^appNS:appDataType`
- `'''The librarian said, "Perhaps you would enjoy 'War and Peace'."'''`
- `1`, igual a `"1"^^xsd:integer`
- `1.3`, igual a `"1.3"^^xsd:decimal`
- `1.300`, igual a `"1.300"^^xsd:decimal`
- `1.0e6`, igual a `"1.0e6"^^xsd:double`
- `true`, igual a `"true"^^xsd:boolean`
- `false`, igual a `"false"^^xsd:boolean`

Reglas de gramática:

[60]	*RDFLiteral*	::=	String (LANGTAG | ('^^' IRIref))?
[61]	*NumericLiteral*	::=	NumericLiteralUnsigned | NumericLiteralPositive | NumericLiteralNegative
[62]	*NumericLiteralUnsigned*	::=	INTEGER | DECIMAL | DOUBLE
[63]	*NumericLiteralPositive*	::=	INTEGER_POSITIVE | DECIMAL_POSITIVE | DOUBLE_POSITIVE
[64]	*NumericLiteralNegative*	::=	INTEGER_NEGATIVE | DECIMAL_NEGATIVE | DOUBLE_NEGATIVE
[65]	*BooleanLiteral*	::=	'true' | 'false'
[66]	*String*	::=	STRING_LITERAL1 | STRING_LITERAL2 | STRING_LITERAL_LONG1 | STRING_LITERAL_LONG2
[76]	*LANGTAG*	::=	'@' [a-zA-Z]+ ('-' [a-zA-Z0-9]+)*
[77]	*INTEGER*	::=	[0-9]+
[78]	*DECIMAL*	::=	[0-9]+ '.' [0-9]* | '.' [0-9]+
[79]	*DOUBLE*	::=	[0-9]+ '.' [0-9]* EXPONENT | '.' ([0-9])+ EXPONENT | ([0-9])+ EXPONENT
[80]	*INTEGER_POSITIVE*	::=	'+' INTEGER
[81]	*DECIMAL_POSITIVE*	::=	'+' DECIMAL
[82]	*DOUBLE_POSITIVE*	::=	'+' DOUBLE
[83]	*INTEGER_NEGATIVE*	::=	'-' INTEGER
[84]	*DECIMAL_NEGATIVE*	::=	'-' DECIMAL
[85]	*DOUBLE_NEGATIVE*	::=	'-' DOUBLE
[86]	*EXPONENT*	::=	[eE] [+-]? [0-9]+
[87]	*STRING_LITERAL1*	::=	"'" (([^#x27#x5C#xA#xD]) | ECHAR)* "'"
[88]	*STRING_LITERAL2*	::=	'"' (([^#x22#x5C#xA#xD]) | ECHAR)* '"'

Los tokens correspondientes a las producciones INTEGER, DECIMAL, DOUBLE y BooleanLiteral son equivalentes a un literal tipado con el valor léxico del token y el correspondiente tipo de datos (`xsd:integer`, `xsd:decimal`, `xsd:double`, `xsd:boolean`).

4.1.3 Sintaxis para Variables de Interrogación

Las variables de interrrogación en consultas SPARQL tienen alcance global; el uso de un nombre de variable determinado en cualquier lugar de una consulta identifica a la misma variable. Las variables son prefijadas mediante "?" o "$"; la "?" o "$" que no forman parte del nombre de la variable. En una consulta, `$abc` y `?abc` identifican la misma variable. Los posibles nombres para variables se proporcionan en la gramática SPARQL.

Reglas de gramática:

[44]	*Var*	::=	VAR1 | VAR2
[74]	*VAR1*	::=	'?' VARNAME
[75]	*VAR2*	::=	'$' VARNAME
[97]	*VARNAME*	::=	(PN_CHARS_U | [0-9]) (PN_CHARS_U | [0-9] | #x00B7 | [#x0300-#x036F] | [#x203F-#x2040])*

4.1.4 Sintaxis para nodos en blanco

Los nodos en blanco en los patrones de grafo actúan como variables indistintas, no como referencias a nodos en blanco específicos en los datos consultados.

Los nodos en blanco se indican mediante la forma etiquetada, como "`_:abc`", o la forma abreviada "`[]`". Un nodo en blanco que se utiliza en un solo lugar en la sintaxis de la consulta puede indicarse con `[]`. Un único nodo en blanco se utiliza para formar el patrón de tripleta. Las etiquetas de nodo en blanco se escriben de la misma forma que "`_:abc`" para un nodo en blanco con la etiqueta "`abc`". El mismo nodo en blanco no puede utilizarse en dos diferentes patrones de gráfo básicos en la misma consulta.

La construcción `[:p :v]` se puede utilizar en los patrones de tripleta. Crea una etiqueta de nodo en blanco que sirve como sujeto a todas las parejas predicado-objeto contenidas. El nodo en blanco creado también se puede utilizar en otros patrones de tripleta en las posiciones de sujeto y objeto.

Las siguientes dos formas

```
[ :p "v" ] .
```

```
[] :p "v" .
```

asignan una única etiqueta de nodo en blanco (aquí "`b57`") y son equivalentes a escribir:

```
_:b57 :p "v" .
```

Esta etiqueta de nodo en blanco asignada puede usarse como el sujeto u objeto de posteriores patrones de tripletas. Por ejemplo, como un sujeto:

```
[ :p "v" ] :q "w" .
```

que es equivalente a las dos tripletas:

```
_:b57 :p "v" .
_:b57 :q "w" .
```

y como un objeto:

```
:x :q [ :p "v" ] .
```

que es equivalente a las dos tripletas:

```
:x  :q _:b57 .
_:b57 :p "v" .
```

La sintaxis abreviada para los nodos en blanco puede combinarse con otras abreviaturas para sujetos comunes y predicados comunes.

```
  [ foaf:name  ?name ;
    foaf:mbox  <mailto:alice@example.org> ]
```

Es lo mismo que escribir el siguiente patrón de grafo básico para alguna etiqueta de nodo en blanco asignada exclusivamente, "`b18`":

```
  _:b18  foaf:name  ?name .
  _:b18  foaf:mbox  <mailto:alice@example.org> .
```

Reglas gramaticales:

[39]	*BlankNodePropertyList*	::=	'['PropertyListNotEmpty']'
[69]	*BlankNode*	::=	BLANK_NODE_LABEL | ANON
[73]	*BLANK_NODE_LABEL*	::=	'_:' PN_LOCAL
[94]	*ANON*	::=	'[' WS* ']'

4.2 Sintaxis para patrones de tripleta

Los Patrones de Tripleta se escriben como una lista, separada por espacios, de sujeto, predicado y objeto; hay formas abreviadas para escribir algunas construcciones de patrones comunes de tripleta.

Los siguientes ejemplos expresan la misma consulta:

```
PREFIX  dc: <http://purl.org/dc/elements/1.1/>
SELECT  ?title
WHERE   { <http://example.org/book/book1> dc:title ?title }
```

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

SELECT  $title
WHERE   { :book1  dc:title  $title }
```

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

SELECT  $title
WHERE   { <book1>  dc:title  ?title }
```

Reglas gramaticales:

[32]	*TriplesSameSubject*	::=	VarOrTermPropertyListNotEmpty | TriplesNodePropertyList
[33]	*PropertyListNotEmpty*	::=	VerbObjectList (';' (VerbObjectList)?)*
[34]	*PropertyList*	::=	PropertyListNotEmpty?
[35]	*ObjectList*	::=	Object (',' Object)*
[37]	*Verb*	::=	VarOrIRIref | 'a'

4.2.1 Listas de Predicados-Objetos

Los patrones de tripleta con un sujeto común pueden escribirse de forma que el sujeto se escriba una sola vez y se use para más de unpatrón de tripleta empleando la notación ";".

```
    ?x  foaf:name  ?name ;
        foaf:mbox  ?mbox .
```

Es lo mismo que escribir los patrones de tripleta:

```
    ?x  foaf:name  ?name .
    ?x  foaf:mbox  ?mbox .
```

4.2.2 Listas de Objetos

Si los patrones de tripleta comparten tanto el sujeto como el predicado, los objetos pueden separarse mediante ",".

```
    ?x foaf:nick  "Alice" , "Alice_" .
```

es lo mismo que escribir los patrones de tripleta:

```
   ?x  foaf:nick  "Alice" .
   ?x  foaf:nick  "Alice_" .
```

Las listas de objetos pueden combinarse con listas predicado-objeto:

```
   ?x  foaf:name ?name ; foaf:nick  "Alice" , "Alice_" .
```

es equivalente a:

```
   ?x  foaf:name  ?name .
   ?x  foaf:nick  "Alice" .
   ?x  foaf:nick  "Alice_" .
```

4.2.3 Colecciones RDF

Las colecciones RDF pueden escribirse como patrones de tripleta usando la sintaxis "(elemento1 elemento2 ...)". La forma "`()`" es una alternativa para la IRI `http://www.w3.org/1999/02/22-rdf-syntax-ns#nil`. Cuando se usa con una colección de elementos, como en `(1 ?x 3 4)`, se asignan a la colección patrones de tripleta con nodos en blanco. El nodo en blanco al principio de la colección puede usarse como un sujeto u objeto en otros patrones de tripleta. Los nodos en blanco asignados por la sintaxis de la colección no aparecen en otras partes de la consulta.

```
(1 ?x 3 4) :p "w" .
```

es azúcar sintáctico para (cabe señalar que `b0`, `b1`, `b2` y `b3` no aparecen en otras partes de la consulta):

```
    _:b0  rdf:first  1 ;
          rdf:rest   _:b1 .
    _:b1  rdf:first  ?x ;
          rdf:rest   _:b2 .
    _:b2  rdf:first  3 ;
          rdf:rest   _:b3 .
    _:b3  rdf:first  4 ;
          rdf:rest   rdf:nil .
    _:b0  :p         "w" .
```

Las colecciones RDF pueden ser anidadas y pueden involucrar otras formas sintácticas:

```
(1 [:p :q] ( 2 ) ) .
```

es azúcar sintáctico para:

```
    _:b0  rdf:first  1 ;
          rdf:rest   _:b1 .
    _:b1  rdf:first  _:b2 .
    _:b2  :p         :q .
    _:b1  rdf:rest   _:b3 .
    _:b3  rdf:first  _:b4 .
    _:b4  rdf:first  2 ;
          rdf:rest   rdf:nil .
    _:b3  rdf:rest   rdf:nil .
```

Reglas gramaticales:

```
[40]   Collection  ::=  '(' GraphNode+ ')'
[92]   NIL         ::=  '(' WS* ')'
```

4.2.4 rdf:type

Se puede utilizar la palabra clave "`a`" como un predicado en un patrón de tripleta; es una alternativa para la IRI `http://www.w3.org/1999/02/22-rdf-syntax-ns#type`. Esta palabra clave es sensible a las mayúsculas.

```
?x  a   :Class1 .
[ a :appClass ] :p "v" .
```

es azúcar sintáctico para:

```
?x    rdf:type  :Class1 .
_:b0  rdf:type  :appClass .
_:b0  :p        "v" .
```

5 Patrones de grafo

SPARQL se basa en la correspondencia entre patrones de grafo. Existen varios modos para formar patrones de grafo más complejos mediante la combinación de patrones más pequeños:

- Patrones de grafo básicos, donde un conjunto de patrones de tripleta debe concordar
- Patrones de grafo de grupo, donde un conjunto de patrones de grafo deben concordar totalmente
- Patrones de grafo opcionales, donde patrones adicionales pueden ampliar una solución
- Patrones de grafo alternativos, donde se intenta encontrar correspondencias con dos o más patrones
- Patrones sobre grafos con nombre, donde los patrones concuerdan con grafos con nombre

En esta sección se describen las dos formas de combinar patrones a partir de su conjunción: patrones de grafo básicos, que combinan patrones de tripleta, y patrones de grafo de grupo, que combinan todos los demás patrones de grafo.

El patrón de grafo más externo de una consulta se denomina patrón de consulta. Gramaticalmente se identifica como `GroupGraphPattern` en

```
[13]   WhereClause  ::=  'WHERE'?
                          GroupGraphPattern
```

5.1 Patrones de grafo básicos

Los patrones de grafos básicos son conjuntos de patrones de tripleta. La concordancia entre patrones de grafo SPARQL se define mediante la combinación de los resultados de concordancia de patrones de grafo básicos.

Una secuencia de patrones de tripleta interrumpidos por un filtro consta de un solo patrón de grafo básico. Cualquier patrón de grafo termina un patrón de grafo básico.

5.1.1 Etiquetas de nodos en blanco

Al utilizar nodos en blanco con la forma `_:abc`, las etiquetas afectan al patrón de grafo básico. En cualquier consulta, una etiqueta puede usarse solo en un patrón de grafo básico.

5.1.2 Ampliando la concordancia de patrones de grafo básicos

SPARQL ha sido definido para encontrar correspondencias entre grafos RDF con una vinculación simple. SPARQL puede ampliarse a otro tipo de vinculación si se dan ciertas condiciones tal y como se describe más adelante.

5.2 Patrones de grafo de grupo

En una expresión de consulta SPARQL, un patrón de grafo de grupo se delimitan mediante llaves: `{}`. Por ejemplo, el siguiente patrón de consulta es un patrón de grafo de grupo con un solo patrón de grafo básico.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  {
          ?x foaf:name ?name .
          ?x foaf:mbox ?mbox .
       }
```

Puede obtenerse la misma solución a partir de una consulta que agrupe los patrones de tripleta en dos patrones de grafo básico. Por ejemplo, la sigiuente consulta tiene una estructura diferente pero arroja la misma solución que en el ejemplo anterior:

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  { { ?x foaf:name ?name . }
         { ?x foaf:mbox ?mbox . }
       }
```

Reglas gramaticales:

[20]	*GroupGraphPattern*	::=	'{' TriplesBlock? ((GraphPatternNotTriples | Filter) '.'? TriplesBlock?)* '}'
[21]	*TriplesBlock*	::=	TriplesSameSubject ('.' TriplesBlock?)?
[22]	*GraphPatternNotTriples*	::=	OptionalGraphPattern | GroupOrUnionGraphPattern | GraphGraphPattern

5.2.1 Patrón de grupo vacío

El patrón de grupo:

```
{ }
```

concuerda con cualquier grafo (incluido el grafo vacío) con una solución en la que no se vincula ninguna variable. Por ejemplo:

```
SELECT ?x
WHERE {}
```

concuerda con una solución en la que variable `x` no está vinculada.

5.2.2 Ámbito de los filtros

Una restricción, expresada con `FILTER`, es es una restricción de soluciones sobre el grupo completo en el que aparece el filtro. Los siguientes patrones tienen todos las mismas soluciones:

```
{   ?x foaf:name ?name .
    ?x foaf:mbox ?mbox .
    FILTER regex(?name, "Smith")
}
```

```
{   FILTER regex(?name, "Smith")
    ?x foaf:name ?name .
    ?x foaf:mbox ?mbox .
}
```

```
{   ?x foaf:name ?name .
    FILTER regex(?name, "Smith")
    ?x foaf:mbox ?mbox .
}
```

5.2.3 Ejemplos de patrones de grafo de grupo

```
{
  ?x foaf:name ?name .
  ?x foaf:mbox ?mbox .
}
```

es un grupo de un patrón de grafo básico que consta de dos patrones de tripleta.

```
{
  ?x foaf:name ?name . FILTER regex(?name, "Smith")
  ?x foaf:mbox ?mbox .
}
```

es un grupo de un patrón de grafo básico y un filtro que consta de dos patrones de tripleta; los filtros no rompen el patrón de grafo básico en dos.

```
{
  ?x foaf:name ?name .
  {}
  ?x foaf:mbox ?mbox .
}
```

es un grupo de tres elementos, un patrón de grafo básico con un patrón de tripleta, un grupo vacío y otro patrón de grafo básico con un patrón de tripleta.

6 Inclusión de valores opcionales

Los patrones de grafo básicos permiten que las aplicaciones realicen consultas donde todo el patrón debe concordar para obtener una solución. Para cada solución de una consulta conteniendo solo un patrón de grafo de grupo con al menos un patrón de grafo básico, cada variable está vinculada a un término RDF en una solución. Sin embargo, las estructuras completas regulares no pueden ser asumidas en todos los grafos RDF. Es útil poder disponer de las consultas que permitan añadir información a la solución donde la información esté disponible, pero sin rechazar la solución debido a que una parte del patrón de consulta no concuerde. Una concordancia opcional proporciona esta posibilidad: si la parte opcional no concuerda, no se crea ningún vínculo pero no se elimina la solución.

6.1 Concordancia de patrones opcionales

Puede especificarse sintácticamente partes opcionales de un patrón de grafo con la cláusula OPTIONAL aplicándose a patrones de grafo:

```
pattern OPTIONAL { pattern }
```

La forma sintáctica:

```
{ OPTIONAL { pattern } }
```

es equivalente a:

```
{ { } OPTIONAL { pattern } }
```

Regla gramatical:

```
[23]   OptionalGraphPattern ::= 'OPTIONAL'GroupGraphPattern
```

La cláusula `OPTIONAL` es asociativa por la izquierda:

```
pattern OPTIONAL { pattern } OPTIONAL { pattern }
```

es equivalente a:

```
{ pattern OPTIONAL { pattern } } OPTIONAL { pattern }
```

En una concordancia opcional, el patrón de grafo opcional concuerda con un grafo, y en consecuencia definiendo y añadiendo vínculos a una o más soluciones, o conservando una solución sin cambios y por lo tanto sin añadir vínculos adicionales.

Datos:

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

_:a  rdf:type        foaf:Person .
_:a  foaf:name       "Alice" .
_:a  foaf:mbox       <mailto:alice@example.com> .
_:a  foaf:mbox       <mailto:alice@work.example> .

_:b  rdf:type        foaf:Person .
_:b  foaf:name       "Bob" .
```

Consulta:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE  { ?x foaf:name  ?name .
         OPTIONAL { ?x  foaf:mbox  ?mbox }
       }
```

Con los datos anteriores, el resultado de la consulta es:

name	mbox
"Alice"	<mailto:alice@example.com>
"Alice"	<mailto:alice@work.example>
"Bob"	

No hay ningún valor de `mbox` en la solución donde el nombre es `"Bob"`.

Esta consulta encuentra los nombres de personas en los datos. Si hay una tripleta con predicado `mbox` y el mismo sujeto, una solución también contendrá

el objeto de dicha tripleta. En este ejemplo, únicamente se proporciona un patrón de tripleta en la parte opcional de la concordancia de la consulta, sin embargo, por regla general, la parte opcional puede ser cualquier patrón de grafo. Los patrones de grafo opcionales deben concordar entre sí para que afecten a la solución de la consulta.

6.2 Restricciones sobre concordancias de patrones opcionales

Las restricciones pueden indicarse en un patrón de grafo opcional. Por ejemplo:

```
@prefix dc:   <http://purl.org/dc/elements/1.1/> .
@prefix :     <http://example.org/book/> .
@prefix ns:   <http://example.org/ns#> .

:book1  dc:title  "SPARQL Tutorial" .
:book1  ns:price  42 .
:book2  dc:title  "The Semantic Web" .
:book2  ns:price  23 .
```

```
PREFIX  dc:  <http://purl.org/dc/elements/1.1/>
PREFIX  ns:  <http://example.org/ns#>
SELECT  ?title ?price
WHERE   { ?x dc:title ?title .
          OPTIONAL { ?x ns:price ?price . FILTER (?price < 30) }
        }
```

title	price
"SPARQL Tutorial"	
"The Semantic Web"	23

No aparece el precio para el libro titulado "SPARQL Tutorial" debido a que el patrón de grafo opcional no dio lugar a una solución con la variable "`price`".

6.3 Múltiples patrones de grafos opcionales

Los patrones de grafo se definen de forma recursiva. Un patrón de grafo puede tener ninguno o más patrones de grafo, y cualquier parte de un patrón de consulta puede tener una parte opcional. En el siguiente ejemplo hay dos patrones de grafo opcionales.

Datos:

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name       "Alice" .
_:a  foaf:homepage   <http://work.example.org/alice/> .

_:b  foaf:name       "Bob" .
_:b  foaf:mbox       <mailto:bob@work.example> .
```

Consulta:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox ?hpage
WHERE  { ?x foaf:name  ?name .
         OPTIONAL { ?x foaf:mbox ?mbox } .
         OPTIONAL { ?x foaf:homepage ?hpage }
       }
```

Resultados de la consulta:

name	mbox	hpage
"Alice"		<http://work.example.org/alice/>
"Bob"	<mailto:bob@work.example>	

7 Concordancias alternativas

SPARQL proporciona un mecanismo para combinar patrones de grafo de modo que alguno de los patrones de grafo alternativos concuerden. Si más de una alternativa concuerda se obtienen todas las soluciones de patrones posibles.

Los patrones alternativos se especifican sintácticamente mediante la clausula `UNION`.

Datos:

```
@prefix dc10:  <http://purl.org/dc/elements/1.0/> .
@prefix dc11:  <http://purl.org/dc/elements/1.1/> .

_:a  dc10:title     "SPARQL Query Language Tutorial" .
_:a  dc10:creator   "Alice" .

_:b  dc11:title     "SPARQL Protocol Tutorial" .
_:b  dc11:creator   "Bob" .

_:c  dc10:title     "SPARQL" .
_:c  dc11:title     "SPARQL (updated)" .
```

Consulta:

```
PREFIX dc10:  <http://purl.org/dc/elements/1.0/>
PREFIX dc11:  <http://purl.org/dc/elements/1.1/>

SELECT ?title
WHERE  { { ?book dc10:title  ?title } UNION { ?book dc11:title  ?tit
```

Resultados de la consulta:

title
"SPARQL Protocol Tutorial"
"SPARQL"
"SPARQL (updated)"

```
"SPARQL Query Language Tutorial"
```

Esta consulta busca los títulos de los libros en los datos, si el título se registra utilizando las propiedades Dublin Core de la versión 1.0 o la versión 1.1. Para determinar exactamente como se almacena la información, una consulta podría utilizar diferentes variables para las dos alternativas:

```
PREFIX dc10:  <http://purl.org/dc/elements/1.0/>
PREFIX dc11:  <http://purl.org/dc/elements/1.1/>

SELECT ?x ?y
WHERE  { { ?book dc10:title ?x } UNION { ?book dc11:title  ?y } }
```

x	y
	"SPARQL (updated)"
	"SPARQL Protocol Tutorial"
"SPARQL"	
"SPARQL Query Language Tutorial"	

Esto devolverá los resultados con la variable `x` vinculada a las soluciones de la parte izquierda de la `UNION`, y con la variable `y` vinculada a las soluciones de la parte derecha. Si ninguna parte del patrón de `UNION` concuerda, entonces el patrón de grafo no concordará.

El patrón de `UNION` combina los patrones de grafo; cada pattern combines graph patterns; cada posibilidad alternativa puede contener más de un patrón de tripleta:

```
PREFIX dc10:  <http://purl.org/dc/elements/1.0/>
PREFIX dc11:  <http://purl.org/dc/elements/1.1/>

SELECT ?title ?author
WHERE  { { ?book dc10:title ?title .  ?book dc10:creator ?author }
         UNION
         { ?book dc11:title ?title .  ?book dc11:creator ?author }
       }
```

author	title
"Alice"	"SPARQL Protocol Tutorial"
"Bob"	"SPARQL Query Language Tutorial"

En esta consulta solo concuerda un libro si ambos predicados, título y creador, provienen de la misma versión de Dublin Core.

Grammar rule:

```
[25]   GroupOrUnionGraphPattern  ::=  GroupGraphPattern ( 'UNION' GroupGraphPattern )*
```

8 Conjuntos de datos RDF

El modelo de datos RDF expresa la información como grafos formados de tripletas con sujeto, predicado y objeto. Muchos sistemas de almacenamiento de datos RDF mantienen múltiples grafos y registra información sobre cada uno ellos, permitiendo que una aplicación realice consultas que involucren más de un grafo.

Una consulta SPARQL se ejecuta en un *Conjunto de datos RDF* que representa una colección de grafos. Un conjunto de datos RDF comprende un grafo, el grafo por defecto (default graph), que no tiene un nombre, y cero o más grafos con nombre (named graph), donde cada nombre de grafo se identifica mediante una referencia IRI. Puede suceder que en una consulta SPARQL diferentes partes del patrón de consulta concuerdan con diferentes grafos, tal y como se describe en la sección 8.3 Consultas a conjuntos de datos.

Un conjunto de datos RDF puede contener cero grafos con nombre; un conjunto de datos RDF también puede contener un grafo por defecto. Una consulta no tiene que implicar necesariamente la concordancia con el grafo por defecto; la correspondencia de una consulta puede comprender únicamente a los grafos con nombre.

El grafo utilizado para la concordancia de un patrón de grafo básico se denomina *grafo activo*. En la sección anterior todas las consultas se ejecutaban frente a un único grafo, el grafo por defecto de un conjunto de datos RDF es el grafo activo. La cláusula `GRAPH` se utiliza, como parte de la consulta, para definir como grafo activo a uno de los grafos con nombre del conjunto de datos.

8.1 Ejemplos de conjuntos de datos RDF

La definición del conjunto de datos RDF no limita las relaciones del grafo por defecto y de los grafos con nombre. La información puede estar repetida en diferentes grafos; pudiendo descubrirse relaciones entre grafos. Existen dos convenciones de utilidad:

- tener información en el grafo por defecto sobre la procedencia de los grafos con nombre
- incluir dicha información tanto en los grafos con nombre como en el grafo por defecto.

Ejemplo 1:

```
# Default graph
@prefix dc: <http://purl.org/dc/elements/1.1/> .

<http://example.org/bob>    dc:publisher  "Bob" .
<http://example.org/alice>  dc:publisher  "Alice" .
```

```
# Named graph: http://example.org/bob
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Bob" .
_:a foaf:mbox <mailto:bob@oldcorp.example.org> .
```

```
# Named graph: http://example.org/alice
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Alice" .
_:a foaf:mbox <mailto:alice@work.example.org> .
```

En el ejemplo anterior, el grafo por defecto contiene los nombres de los editores de dos grafos con nombre. En este ejemplo, las tripletas en los grafos con nombre no son visibles en el grafo por defecto.

Ejemplo 2:

Los datos RDF pueden combinarse mediante la unión RDF [RDF-MT] de grafos. Una convención posible de los grafos en un conjunto de datos RDF es disponer de un grafo por defecto que sea el resultado de la unión RDF que incluya la información de algunos o todos los grafos con nombre.

En el siguiente ejemplo, los grafos con nombre contienen las misma tripletas que el ejemplo anterior. El conjunto de datos RDF incluye una unión RDF de los grafos con nombre en el grafo por defecto, re-etiquetando los nodos en blanco para distinguirlos.

```
# Default graph
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:x foaf:name "Bob" .
_:x foaf:mbox <mailto:bob@oldcorp.example.org> .

_:y foaf:name "Alice" .
_:y foaf:mbox <mailto:alice@work.example.org> .
```

```
# Named graph: http://example.org/bob
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Bob" .
_:a foaf:mbox <mailto:bob@oldcorp.example.org> .
```

```
# Named graph: http://example.org/alice
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Alice" .
_:a foaf:mbox <mailto:alice@work.example> .
```

En una unión RDF, los nodos en blanco en el grafo resultante no se comparten en los grafos originales.

8.2 Especificación de conjuntos de datos RDF

Una consulta SPARQL puede especificar el conjunto de datos a utilizar en la concordancia mediante las cláusulas `FROM` y `FROM NAMED` para describir el conjunto de datos. Si una consulta proporciona dicha descripción, entonces es usado en vez de cualquier conjunto de datos que usaría el servicio de consulta si no se proporcionara la misma. El conjunto de datos RDF también puede ser especificado en una petición del protocolo SPARQL, en cuyo caso la descripción del protocolo anularía cualquier descripción de la propia consulta. Un servicio de consulta puede rechazar una petición de una consulta si la descripción del conjunto de datos no es aceptable para el servicio.

Las cláusulas `FROM` y `FROM NAMED` permiten a la consulta especificar el conjunto de datos mediante una referencia; indican que el conjunto de datos debe incluir los grafos obtenidos a partir de las representaciones de los recursos identificados por las referencias IRI suministradas (por ejemplo, la forma absoluta de las referencias IRI suministradas). El conjunto de datos resultante de aplicar las cláusulas `FROM` y `FROM NAMED` es:

- un grafo por defecto consistente en la unión RDF de los grafos referenciados en las cláusulas `FROM`, y
- un conjunto de pares (IRI, grafo) de cada cláusula `FROM NAMED`.

No se utiliza la cláusula `FROM`, pero hay una o más cláusulas `FROM NAMED`, entonces el conjunto de datos incluirá un grafo vacío para el grafo por defecto.

Reglas gramaticales:

[9]	*DatasetClause*	::=	'FROM' (DefaultGraphClause | NamedGraphClause)
[10]	*DefaultGraphClause*	::=	SourceSelector
[11]	*NamedGraphClause*	::=	'NAMED' SourceSelector
[12]	*SourceSelector*	::=	IRIref

8.2.1 Especificación del grafo por defecto

Cada cláusula `FROM` contiene una referencia IRI que indica el grafo que ha de utilizarse como grafo por defecto. Esta asignación no se realiza considerando el grafo como un grafo con nombre.

En este ejemplo, el conjunto de datos RDF contienen un sencillo grafo por defecto y ningún grafo con nombre:

```
# Default graph (stored at http://example.org/foaf/aliceFoaf)
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name     "Alice" .
_:a  foaf:mbox     <mailto:alice@work.example> .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT  ?name
FROM    <http://example.org/foaf/aliceFoaf>
WHERE   { ?x foaf:name ?name }
```

name
"Alice"

Si la consulta proporciona más de una cláusula `FROM`, proporcionando más de una referencia IRI para indicar el grafo por defecto, entonces el grafo por defecto se basa en la unión RDF de los grafos obtenidos a partir de las representaciones de los recursos identificados por las IRIs suministradas.

8.2.2 Especificación de grafos con nombre

Una consulta puede proporcionar IRIs para grafos con nombre en el conjunto de datos RDF utilizando la cláusula `FROM NAMED`. Cada IRI se utiliza para proporcionar un grafo con nombre al conjunto de datos RDF. Usando la misma IRI en dos o más cláusulas `FROM NAMED` tiene como consecuencia que un grafo con nombre con dicha IRI aparezca en el conjunto de datos.

```
# Graph: http://example.org/bob
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Bob" .
_:a foaf:mbox <mailto:bob@oldcorp.example.org> .
```

```
# Graph: http://example.org/alice
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Alice" .
_:a foaf:mbox <mailto:alice@work.example> .
```

```
...
FROM NAMED <http://example.org/alice>
FROM NAMED <http://example.org/bob>
...
```

La sintaxis de `FROM NAMED` sugiere que las referencias IRI identifican los grafos correspondientes, pero en un conjunto de datos RDF la relación entre una IRI y un grafo es indirecta. La referencia IRI identifica un recurso, y el recurso es representado por un grafo (o de un modo más preciso: a través de un documento que serializa un grafo). For más detalles véase [WEBARCH].

8.2.3 Combinación de FROM y FROM NAMED

Las cláusulas `FROM` y `FROM NAMED` pueden usarse en la misma consulta.

```
# Default graph (stored at http://example.org/dft.ttl)
@prefix dc: <http://purl.org/dc/elements/1.1/> .

<http://example.org/bob>    dc:publisher  "Bob Hacker" .
<http://example.org/alice>  dc:publisher  "Alice Hacker" .
```

```
# Named graph: http://example.org/bob
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Bob" .
_:a foaf:mbox <mailto:bob@oldcorp.example.org> .
```

```
# Named graph: http://example.org/alice
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Alice" .
_:a foaf:mbox <mailto:alice@work.example.org> .
```

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

El conjunto de datos RDF para esta consulta contiene un grafo por defecto y dos grafos con nombre. La cláusula `GRAPH` se describe más adelante.

Las acciones requeridas para construir el conjunto de datos no son determinadas únicamente por la descripción del conjunto de datos. Si se proporciona dos veces una misma referencia IRI en una descripción de conjunto de datos, The actions required to construct the dataset are not determined by the dataset description alone. If an IRI is given twice in a dataset description, ya sea usando dos cláusulas `FROM`, o una cláusula `FROM` y una clásula `FROM NAMED`, entonces no se asume que se hagan únicamente uno o dos intentos para obtener un grafo RDF asociado con la IRI. Por tanto, no se pueden hacer suposiciones sobre la identidad del nodo en blanco en las tripletas que se han obtenida a partir de las dos ocurrencias en la descripción del conjunto de datos. En general, no se pueden hacer suposiciones acerca de la equivalencia de los grafos.

8.3 Consultas a conjuntos de datos

Al consultar una colección de grafos, la cláusula `GRAPH` se utiliza para encontrar concordancias entre patrones y grafos con nombre. La cláusula `GRAPH` puede proporcionar una referencia IRI para seleccionar un grafo o para usar una

variable que se aplicará en las referencias IRI de todos grafos con nombre del conjunto de datos de la consulta.

El uso de la cláusula `GRAPH` cambia el grafo activo para la concordancia con patrones de grafo básicos que forman parte de la consulta. Fuera del uso de la cláusula `GRAPH`, se buscan concordancias del patrones de grafo básicos con el grafo por defecto.

Los siguientes dos grafos se utilizarán en los ejemplos:

```
# Named graph: http://example.org/foaf/aliceFoaf
@prefix  foaf:     <http://xmlns.com/foaf/0.1/> .
@prefix  rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix  rdfs:     <http://www.w3.org/2000/01/rdf-schema#> .

_:a  foaf:name     "Alice" .
_:a  foaf:mbox     <mailto:alice@work.example> .
_:a  foaf:knows    _:b .

_:b  foaf:name     "Bob" .
_:b  foaf:mbox     <mailto:bob@work.example> .
_:b  foaf:nick     "Bobby" .
_:b  rdfs:seeAlso  <http://example.org/foaf/bobFoaf> .

<http://example.org/foaf/bobFoaf>
     rdf:type      foaf:PersonalProfileDocument .
```

```
# Named graph: http://example.org/foaf/bobFoaf
@prefix  foaf:     <http://xmlns.com/foaf/0.1/> .
@prefix  rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix  rdfs:     <http://www.w3.org/2000/01/rdf-schema#> .

_:z  foaf:mbox     <mailto:bob@work.example> .
_:z  rdfs:seeAlso  <http://example.org/foaf/bobFoaf> .
_:z  foaf:nick     "Robert" .

<http://example.org/foaf/bobFoaf>
     rdf:type      foaf:PersonalProfileDocument .
```

Reglas gramaticales:

```
[24]  GraphGraphPattern  ::=  'GRAPH' VarOrIRIref GroupGraphPattern
```

8.3.1 Acceso a grafos con nombre

La siguiente consulta busca concordancias de patrones de grafo en cada uno de los grafos con nombre del conjunto de datos, y forma las soluciones que contienen la variable `src` vinculada a las referencias IRI de los grafos que concuerdan. El patrón de grafo concuerda con el grafo activo que es cada uno de los grafos con nombre del conjunto de datos.

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

SELECT ?src ?bobNick
FROM NAMED <http://example.org/foaf/aliceFoaf>
FROM NAMED <http://example.org/foaf/bobFoaf>
WHERE
  {
    GRAPH ?src
    { ?x foaf:mbox <mailto:bob@work.example> .
      ?x foaf:nick ?bobNick
    }
  }
```

El resultado de la consulta devuelve el nombre de los grafos donde se encontró la información, así como el valor del nick de Bob:

src	bobNick
<http://example.org/foaf/aliceFoaf>	"Bobby"
<http://example.org/foaf/bobFoaf>	"Robert"

8.3.2 Restricciones por IRIs de grafos

La consulta puede restringir la concordancia a aplicar sobre un grafo específico proporcionando la referencia IRI del mismo. De esta forma se establece el grafo activo como el grafo nombrado por dicha IRI. La consulta busca el nick de Bob, tal y como se indica, en el grafo `http://example.org/foaf/bobFoaf`.

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

SELECT ?nick
FROM NAMED <http://example.org/foaf/aliceFoaf>
FROM NAMED <http://example.org/foaf/bobFoaf>
WHERE
  {
     GRAPH data:bobFoaf {
         ?x foaf:mbox <mailto:bob@work.example> .
         ?x foaf:nick ?nick }
  }
```

lo que devuelve una única solución:

nick
"Robert"

8.3.3 Restricciones por IRIs de grafos posibles

Una variable utilizada en una cláusula `GRAPH` puede usarse también en otra cláusula `GRAPH` o en el patrón de grafo para la concordancia con el grafo por defecto del conjunto de datos.

La siguiente consulta usa el grafo con IRI `http://example.org/foaf/aliceFoaf` para encontrar el documento de perfil para Bob; si es así concuerda otro patrón con el grafo. El patrón en la segunda cláusula `GRAPH` encuentra el nodo en blanco (variable `w`) para la persona con el mismo buzón de correo electrónico (indicado por la variable `mbox`) tal y como se encuentra en la primera cláusula `GRAPH` (variable `whom`), debido a que el nodo en blanco usado para la concordancia con la variable `whom` del fichero FOAF de Alice no es el mismo que el nodo en blanco del documento de perfil (ya que se encuentran en grafos diferentes).

```
PREFIX  data:  <http://example.org/foaf/>
PREFIX  foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX  rdfs:  <http://www.w3.org/2000/01/rdf-schema#>

SELECT ?mbox ?nick ?ppd
FROM NAMED <http://example.org/foaf/aliceFoaf>
FROM NAMED <http://example.org/foaf/bobFoaf>
WHERE
{
  GRAPH data:aliceFoaf
  {
    ?alice foaf:mbox <mailto:alice@work.example> ;
           foaf:knows ?whom .
    ?whom  foaf:mbox ?mbox ;
           rdfs:seeAlso ?ppd .
    ?ppd  a foaf:PersonalProfileDocument .
  } .
  GRAPH ?ppd
  {
      ?w foaf:mbox ?mbox ;
         foaf:nick ?nick
  }
}
```

mbox	nick	ppd
<mailto:bob@work.example>	"Robert"	<http://example.org/foaf/bobFoaf>

Cualquier tripleta del fichero FOAF de Alice, dando el `nick` de Bob no se utiliza para proporcionar un nick para Bob, debido a que el patrón en el que participa la variable `nick` está restringida por `ppd` a un documento de perfil personal en particular.

8.3.4 Grafos con nombre y grafos por defecto

Los patrones de consulta pueden involucrar al grafo por defecto y a grafos con nombre. En este ejemplo, un agregador ha leido un recurso web en dos ocasiones diferentes. Cada vez que el agregador lee un grafo, el sistema local proporciona una referencia IRI. Los grafos con casi iguales, pero la dirección de correo electrónico de "Bob" ha cambiado.

En este ejemplo, el grafo por defecto se utiliza para registrar la información de procedencia y los datos RDF actualmente leídos se almacenan en dos grafos separados, para los que el sistema proporciona IRIs diferentes. El conjunto de

datos RDF consiste en dos grafos con nombre y la información sobre los mismos.

Conjunto de datos RDF:

```
# Default graph
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix g:  <tag:example.org,2005-06-06:> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

g:graph1 dc:publisher "Bob" .
g:graph1 dc:date "2004-12-06"^^xsd:date .

g:graph2 dc:publisher "Bob" .
g:graph2 dc:date "2005-01-10"^^xsd:date .
```

```
# Graph: locally allocated IRI: tag:example.org,2005-06-06:graph1
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Alice" .
_:a foaf:mbox <mailto:alice@work.example> .

_:b foaf:name "Bob" .
_:b foaf:mbox <mailto:bob@oldcorp.example.org> .
```

```
# Graph: locally allocated IRI: tag:example.org,2005-06-06:graph2
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Alice" .
_:a foaf:mbox <mailto:alice@work.example> .

_:b foaf:name "Bob" .
_:b foaf:mbox <mailto:bob@newcorp.example.org> .
```

Esta consulta encuentra direcciones de correo electrónico, detallando el nombre de la persona y la fecha en la que se descubrió la información.

```
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 }
  }
```

Los resultados muestran que la dirección de correo electrónico para "Bob" ha cambiado.

name	mbox	date
"Bob"	<mailto:bob@oldcorp.example.org>	"2004-12-06"^^xsd:date
"Bob"	<mailto:bob@newcorp.example.org>	"2005-01-10"^^xsd:date

La referencia IRI para el tipo de dato fecha (date) se ha abreviado en los resultados para mayor claridad.

9 Secuencias de soluciones y modificadores

Los patrones de consulta generarán una colección desordenada de soluciones, siendo cada solución una función parcial de las variables a los términos RDF. Estas soluciones son tratadas entonces como una secuencia (una secuencia de solución), inicialmente sin orden específico; los modificadores de secuencia son aplicados entonces para crear otra secuencia. Finalmente, esta última secuencia se usa para generar uno de los resultados de una forma de consulta SPARQL..

Un modificador de secuencia de solución puede ser alguno de los siguientes:

- Modificador de orden: ordenar las soluciones
- Modificador de proyección: escoger ciertas variables
- Modificador Distinct: garantizar que las soluciones en la secuencia son únicas
- Modificador Reduced: permitir la eliminación del algunas soluciones no únicas
- Modificador Offset: controlar donde comienzan las soluciones en la secuencia global de soluciones
- Modificador Limit: restringir el número de soluciones

Los modificadores se aplican en el orden dado por la lista anterior.

Reglas gramaticales:

[5]	*SelectQuery*	::=	'SELECT' ('DISTINCT' | 'REDUCED')? (Var+ | '*') DatasetClause* WhereClause SolutionModifier
[14]	*SolutionModifier*	::=	OrderClause? LimitOffsetClauses?
[15]	*LimitOffsetClauses*	::=	(LimitClause OffsetClause? | OffsetClause LimitClause?)
[16]	*OrderClause*	::=	'ORDER' 'BY' OrderCondition+

9.1 ORDER BY

La clausula `ORDER BY` establece el orden de una secuencia de solución.

Después de la cláusula `ORDER BY` hay una secuencia de elementos de comparación de orden, compuesta de una expresión y un modificador de orden opcional (`ASC()` o `DESC()`). Cada comparador de orden es ascendente (indicado por el modificador `ASC()` o por ningún modificador) o descendente (indicado por el modificador `DESC()`).

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

SELECT ?name
WHERE { ?x foaf:name ?name }
ORDER BY ?name
```

```
PREFIX     :    <http://example.org/ns#>
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX xsd:     <http://www.w3.org/2001/XMLSchema#>

SELECT ?name
WHERE { ?x foaf:name ?name ; :empId ?emp }
ORDER BY DESC(?emp)
```

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

SELECT ?name
WHERE { ?x foaf:name ?name ; :empId ?emp }
ORDER BY ?name DESC(?emp)
```

El operador "<" (vea las secciones 11.3 Conversión de operadores y 11.3.1 Extensibilidad de Operadores) define el orden relativo de pares de `numerics`, `simple literals`, `xsd:strings`, `xsd:booleans` y `xsd:dateTimes`. Los pares de IRIs se ordenan comparándolos como `simple literals`.

SPARQL también establece un orden entre algunos tipos de términos RDF que no podrían ser ordenados de otra forma:

1. (Prioridad más baja) no hay ningún valor asignado a la variable oexpresión en esta solución;
2. Nodos en blanco;
3. IRIs;
4. Literales RDF.

Un literal simple (plain literal) es inferior a un literal RDF con tipo `xsd:string` de la misma forma léxica.

SPARQL no define un orden general de todos los posibles términos RDF. A continuación se incluyen algunos ejemplos de pares de términos para los que no está definido el orden relativo:

- "a" y "a"@en_gb (un literal simple y un literal con una etiqueta de idioma)
- "a"@en_gb y "b"@en_gb (dos literales con etiquetas de idioma)
- "a" y "a"^^xsd:string (un literal simple y un xsd:string)
- "a" y "1"^^xsd:integer (un literal simple y literal con un tipo de datos nativo (compatible))
- "1"^^my:integer y "2"^^my:integer (dos tipos de datos no-nativos)
- "1"^^xsd:integer y "2"^^my:integer (un tipo de datos nativo y un tipo de datos no nativo)

Esta lista de vínculos de variables es ascendente:

Término RDF	Razón

Término RDF	Razón
	Los resultados no asignados son ordenados primeros.
_:z	Los nodos en blanco siguen a los no asignados.
_:a	No hay un orden relativo de los nodos en blanco.
<http://script.example/Latin>	Los IRIs siguen a los nodos en blanco.
<http://script.example/Кириллица>	El caracter en la 23° posición, "K", tiene un punto decódigo Unicode 0x41A, que es mayor que 0x4C ("L").
<http://script.example/漢字>	El caracter en la 23° posición, "漢", tiene un punto decódigo Unicode 0x6F22, que es mayor que 0x41A ("K").
"http://script.example/Latin"	Los literales simples siguen a los IRIs.
"http://script.example/Latin"^^xsd:string	Los tipos xsd:strings siguen a los literales simples.

El orden ascendente de dos soluciones con respecto a un comparador de ordenación se establece mediante la sustitución de los vinculos de la solución en las expresiones y la comparación de ellas con el operador "<". El orden descendente es el inverso del orden ascendente.

El orden relativo de dos soluciones es el orden relativo de las dos soluciones con respecto al primer comparador de ordenación en la secuencia. Para soluciones en las que las sustituciones de los vinculos de soluciones producen el mismo término RDF, el orden es el orden relativo de las dos soluciones con respecto al siguiente comparador de ordenación. El orden relativo de las dos soluciones es indefinido si ninguna expresión de orden evaluada para las dos soluciones produce distintos términos RDF.

La ordenación de una secuencia de soluciones siempre da lugar a una secuencia con el mismo número de soluciones en ella.

La utilización del modificador `ORDER BY` en una secuencia de solución para una consulta del tipo `CONSTRUCT` o `DESCRIBE` no tiene efecto directo ya que únicamente `SELECT` devuelve una secuencia de resultados. Cuando se utiliza en combinación con los modificadores `LIMIT` y `OFFSET`, `ORDER BY` puede ser usado para devolver resultados generados desde una porción diferente de la secuencia de soluciones. Una consulta `ASK` no incluye los modificadores `ORDER BY`, `LIMIT` u `OFFSET`.

Reglas gramaticales:

[16]	*OrderClause*	::=	'ORDER' 'BY' OrderCondition+
[17]	*OrderCondition*	::=	(('ASC' | 'DESC') BrackettedExpression) | (Constraint | Var)
[18]	*LimitClause*	::=	'LIMIT' INTEGER
[19]	*OffsetClause*	::=	'OFFSET' INTEGER

9.2 Projección

La secuencia de soluciones puede ser transformada en una secuencia que implica sólo un subconjunto de variables. Para cada solución de la secuencia, se forma una nueva solución con una selección determinada de las variables utilizando la forma de consulta SELECT.

El siguiente ejemplo muestra una consulta para extraer sólo los nombres de las personas descritas en un grafo RDF utilizando las propiedades FOAF:

```
@prefix foaf:        <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name       "Alice" .
_:a  foaf:mbox       <mailto:alice@work.example> .

_:b  foaf:name       "Bob" .
_:b  foaf:mbox       <mailto:bob@work.example> .
```

```
PREFIX foaf:       <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE
 { ?x foaf:name ?name }
```

name
"Bob"
"Alice"

9.3 Soluciones duplicadas

Una secuencia de soluciones sin modificador de consulta `DISTINCT` o `REDUCED` conservará las soluciones duplicadas.

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .

_:x    foaf:name   "Alice" .
_:x    foaf:mbox   <mailto:alice@example.com> .

_:y    foaf:name   "Alice" .
_:y    foaf:mbox   <mailto:asmith@example.com> .

_:z    foaf:name   "Alice" .
_:z    foaf:mbox   <mailto:alice.smith@example.com> .
```

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?name WHERE { ?x foaf:name ?name }
```

name
"Alice"
"Alice"
"Alice"

Los modificadores `DISTINCT` y `REDUCED` controlan si los duplicados se incluyen en los resultados de la consulta.

9.3.1 DISTINCT

El modificador `DISTINCT` elimina soluciones duplicadas. Específicamente, se elimina del conjunto de soluciones cada solución que vincule las mismas variables con el mismo término RDF como otra solución.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?name WHERE { ?x foaf:name ?name }
```

name
"Alice"

Obsérvese que, por el orden de modificadores de las secuencias de soluciones, los duplicados son eliminados antes de que se apliquen los modificadores limit u offset.

9.3.2 REDUCED

Mientras que el modificador `DISTINCT` asegura la eliminación de lassoluciones duplicadas del conjunto de soluciones, el modificador `REDUCED` simplemente les permite ser eliminadas. La cardinalidad de unconjunto de vinculos de variables en un conjunto de soluciones `REDUCED` de al menos uno y no mayor que la cardinalidad de los conjuntos desoluciones sin modificadores `DISTINCT` o `REDUCED`. Por ejemplo, utilizando los datosanteriores, la consulta

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT REDUCED ?name WHERE { ?x foaf:name ?name }
```

puede tener uno, dos (aquí mostradas) o tres soluciones:

name
"Alice"
"Alice"

9.4 OFFSET

El modificador `OFFSET` produce que las soluciones generadas empiecen después de un número especificado de so luciones. El valor de cero para `OFFSET` no tiene efecto.

El uso de los modificadores `LIMIT` y `OFFSET` para seleccionar diferentes subconjuntos de soluciones de la consulta no será útil a menos que el orden sea predecible mediante el modificador `ORDER BY`.

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

SELECT  ?name
WHERE   { ?x foaf:name ?name }
ORDER BY ?name
LIMIT   5
OFFSET  10
```

9.5 LIMIT

La cláusula `LIMIT` establece un límite superior al número de soluciones devueltas. Si el número de soluciones reales es mayor que el límite, entonces como mucho serán devueltas el número límite de soluciones.

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

SELECT ?name
WHERE { ?x foaf:name ?name }
LIMIT 20
```

El valor de cero para `LIMIT` causaría que no se devolvieran resultados. Un límite no puede ser negativo.

10 Formas de consulta

SPARQL tiene cuatro formas de consulta. Estas formas de consulta usan las soluciones que concuerdan con el patrón para formar los conjuntos de resultados o grafos RDF. Las formas de consulta son:

SELECT
: Devuelve todo, o un subconjunto de las variables vinculadas en una concordancia con un patrón de búsqueda

CONSTRUCT

Devuelve un grafo RDF construido mediante la sustitución de variables en un conjunto de plantillas de tripleta.

ASK

Devuelve un valor booleano indicando si se encuentra o no una concordancia para un patrón de consulta.

DESCRIBE

Devuelve un grafo RDF que describe los recursos encontrados.

Puede usarse el Formato XML de los resultados de variables vinculadas SPARQL para serializar el conjunto de resultados de una consulta `SELECT` o el resultado booleano de una consulta `ASK`.

10.1 SELECT

La forma de los resultados de SELECT devuelve las variables y sus vinculaciones directamente. La sintaxis `SELECT *` es una abreviación que selecciona todas las variables de una consulta.

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .

_:a    foaf:name   "Alice" .
_:a    foaf:knows  _:b .
_:a    foaf:knows  _:c .

_:b    foaf:name   "Bob" .

_:c    foaf:name   "Clare" .
_:c    foaf:nick   "CT" .
```

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
SELECT ?nameX ?nameY ?nickY
WHERE
  { ?x foaf:knows ?y ;
       foaf:name ?nameX .
    ?y foaf:name ?nameY .
    OPTIONAL { ?y foaf:nick ?nickY }
  }
```

nameX	nameY	nickY
"Alice"	"Bob"	
"Alice"	"Clare"	"CT"

Se puede acceder a los conjuntos de resultados mediante una API local, pero también pueden serializarse tanto en XML como en un grafo RDF. El documento Formato XML de los resultados de consultas SPARQL describe un formato XML y proporciona este ejemplo:

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head>
    <variable name="nameX"/>
    <variable name="nameY"/>
    <variable name="nickY"/>
  </head>
  <results>
    <result>
      <binding name="nameX">
        <literal>Alice</literal>
      </binding>
      <binding name="nameY">
        <literal>Bob</literal>
      </binding>
   </result>
    <result>
      <binding name="nameX">
        <literal>Alice</literal>
      </binding>
      <binding name="nameY">
        <literal>Clare</literal>
      </binding>
      <binding name="nickY">
        <literal>CT</literal>
      </binding>
    </result>
  </results>
</sparql>
```

Reglas gramaticales:

```
[5]   SelectQuery  ::=  'SELECT' ( 'DISTINCT' |
                        'REDUCED' )? ( Var+ | '*' )
                        DatasetClause*
                        WhereClauseSolutionModifier
```

10.2 CONSTRUCT

La forma de consulta `CONSTRUCT` devuelve un único grafo RDF especificado por un plantilla de grafo. El resultado es un grafo RDF formado al tomar cada solución de la consulta de la secuencia de solución, sustituyendo las variables en la plantilla de grafo, y combinando las tripletas en un único grafo RDF mediante la unión de conjuntos.

Si cualquier instancia produce una tripleta que contenga una variable no vinculada o una construcción RDF no válida, como un literal en las posiciones del sujeto o precidado, entonces esa tripleta no se incluye en la salida del grafo RDF. La plantilla del grafo puede contener tripletas sin variables (conocidas como base o tripletas explícitas), y éstas también aparecen en la salida del grafo RDF devuelta por la forma de consulta CONSTRUCT.

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .

_:a    foaf:name   "Alice" .
_:a    foaf:mbox   <mailto:alice@example.org> .
```

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>
CONSTRUCT   { <http://example.org/person#Alice> vcard:FN ?name }
WHERE       { ?x foaf:name ?name }
```

crea las propiedades vcard para la información FOAF:

```
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .

<http://example.org/person#Alice> vcard:FN "Alice" .
```

10.2.1 Plantillas con nodos en blanco

Una plantilla puede crear un grafo RDF conteniendo nodos en blanco. Las etiquetas de de los nodos en blanco pertenecen al ámbito de la plantilla para cada solución. Si la misma etiqueta aparece dos veces en una plantilla, entonces existe un nodo en blanco creado para cada solución, pero al mismo tiempo hay diferentes nodos en blanco para las tripletas generadas por diferentes soluciones de la consulta.

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .

_:a    foaf:givenname   "Alice" .
_:a    foaf:family_name "Hacker" .

_:b    foaf:firstname   "Bob" .
_:b    foaf:surname     "Hacker" .
```

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>

CONSTRUCT { ?x  vcard:N _:v .
            _:v vcard:givenName ?gname .
            _:v vcard:familyName ?fname }
WHERE
 {
    { ?x foaf:firstname ?gname } UNION  { ?x foaf:givenname   ?gname
    { ?x foaf:surname   ?fname } UNION  { ?x foaf:family_name ?fname
 }
```

la anterior consulta crea propiedades vcard correspondiente a la información de FOAF:

```
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .

_:v1 vcard:N          _:x .
_:x vcard:givenName  "Alice" .
_:x vcard:familyName "Hacker" .

_:v2 vcard:N          _:z .
_:z vcard:givenName  "Bob" .
_:z vcard:familyName "Hacker" .
```

El uso de la variable `x` en la plantilla, que en este ejemplo se vincula con los nodos en blanco con etiquetas `_:a` y `_:b` de los datos, produce diferentes etiquetas de nodos en blanco (`_:v1` y `_:v2`) en el grafo RDF resultante.

10.2.2 Acceso a los grafos en el conjunto de datos RDF

Empleando `CONSTRUCT`, es posible extraer partes de la totalidad de los grafos del conjunto destino de datos RDF. Este primer ejemplo devuelve el grafo (si se encuentra en el conjunto de datos) etiquetado con la referencia IRI `http://example.org/aGraph`; en caso contrario, devuelve un grafo vacío.

```
CONSTRUCT { ?s ?p ?o } WHERE { GRAPH <http://example.org/aGraph> { ?
```

El acceso al grafo puede estar condicionado por otra información. Por ejemplo, si el grafo por defecto contiene metadatos sobre los grafos con nombre en el conjunto de datos, entonces una consulta como la siguiente puede extraer un grafo basado en información sobre un grafo con nombre:

```
PREFIX  dc: <http://purl.org/dc/elements/1.1/>
PREFIX app: <http://example.org/ns#>
CONSTRUCT { ?s ?p ?o } WHERE
 {
   GRAPH ?g { ?s ?p ?o } .
   { ?g dc:publisher <http://www.w3.org/> } .
   { ?g dc:date ?date } .
   FILTER ( app:customDate(?date) > "2005-02-28T00:00:00Z"^^xsd:date
 }
```

donde `app:customDate` identifica una función de extensión para transformar los datos al formato del término RDF `xsd:dateTime`.

Regla gramatical:

[6]	*ConstructQuery*	::=	'CONSTRUCT' ConstructTemplate DatasetClause* WhereClauseSolutionModifier

10.2.3 Modificadores de solución y CONSTRUCT

Los modificadores de solución de una consulta afectan a los resultados de una consulta `CONSTRUCT`. En este ejemplo, el grafo de salida de la plantilla `CONSTRUCT` está formado a partir de dos de las soluciones de la correspondencia del patrón de grafo. La consulta produce un grafo con los nombres de las personas de los dos sitios web más consultados. Las tripletas del grafo RDF no están ordenadas.

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix site: <http://example.org/stats#> .

_:a foaf:name "Alice" .
_:a site:hits 2349 .

_:b foaf:name "Bob" .
_:b site:hits 105 .

_:c foaf:name "Eve" .
_:c site:hits 181 .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX site: <http://example.org/stats#>

CONSTRUCT { [] foaf:name ?name }
WHERE
{ [] foaf:name ?name ;
     site:hits ?hits .
}
ORDER BY desc(?hits)
LIMIT 2
```

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:x foaf:name "Alice" .
_:y foaf:name "Eve" .
```

10.3 ASK

Las aplicaciones puede utilizar la forma `ASK` para comprobar si un patrón de consulta tiene solución. No se devuelve información alguna sobre las posibles soluciones a la consulta, únicamente si existe o no una solución.

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name       "Alice" .
_:a  foaf:homepage   <http://work.example.org/alice/> .

_:b  foaf:name       "Bob" .
_:b  foaf:mbox       <mailto:bob@work.example> .
```

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
ASK  { ?x foaf:name  "Alice" }
```

```
yes
```

El Formato XML de los resultados de consultas SPARQL estructura estos resultados del siguiente modo:

```
<?xml version="1.0"?>
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
  <head></head>
  <results>
    <boolean>true</boolean>
  </results>
</sparql>
```

Sobre los mismos datos, la siguiente consulta no devuelve ninguna concordancia al no mercionarse el `mbox` de Alice.

```
PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
ASK  { ?x foaf:name  "Alice" ;
          foaf:mbox  <mailto:alice@work.example> }
```

```
no
```

Regla gramatical:

```
[8]   AskQuery  ::=  'ASK' DatasetClause* WhereClause
```

10.4 DESCRIBE (Informativo)

La forma `DESCRIBE` devuelve como resultado un único grafo RDF con datos sobre recursos. Estos datos no son prescritos por una consulta SPARQL, donde el cliente que realiza la consulta tendría que conocer la estructura RDF en la fuente de datos, pero, en cambio, está determinado por el procesador de consultas SPARQL. El patrón de consulta es usado para crear un conjunto de resultados. La forma `DESCRIBE` toma cada uno de los recursos identificados en una solución, junto con otros recursos directamente nombrados mediante una IRI, y construye un único grafo RDF tomando una "descripción" que puede provenir de cualquier información disponible, incluyendo el conjunto destino de datos RDF objeto. La descripción está determinada por el servicio de consulta. La sintaxis `DESCRIBE *` es una abreviación que describe todas las variables en una consulta.

10.4.1 IRIs explícitas

La cláusula `DESCRIBE` por sí misma puede tomar IRIs para identificar recursos. Precisamente, la consulta `DESCRIBE` más simple es una referencia IRI en la cláusula `DESCRIBE`:

```
DESCRIBE <http://example.org/>
```

10.4.2 Identificación de recursos

Los recursos descritos también pueden obtenerse a partir de las vinculaciones de una variable de la consulta en el conjunto de resultados. Esto permite describir recursos identificados mediante IRI o nodo en blanco en el conjunto de datos:

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
DESCRIBE ?x
WHERE    { ?x foaf:mbox <mailto:alice@org> }
```

La propiedad `foaf:mbox` se define como una propiedad funcional inversa en el vocabulario FOAF. Si se trata como tal, esta consulta devolverá información sobre más de una persona. Sin embargo, si el patrón de consulta tiene múltiples soluciones, los datos RDF para cada una es la unión de todas las descripciones de grafos RDF.

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
DESCRIBE ?x
WHERE    { ?x foaf:name "Alice" }
```

Se pueden proporcionar más de una IRI o variable:

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
DESCRIBE ?x ?y <http://example.org/>
WHERE    {?x foaf:knows ?y}
```

10.4.3 Descripciones de recursos

El RDF obtenido está determinado por la información del editor. Se trata de la información útil que el servicio dispone sobre un recurso. Puede incluir información sobre otros recursos: por ejemplo, los datos RDF de un libro también puede incluir detalles sobre su autor.

Un consulta sencilla como

```
PREFIX ent:  <http://org.example.com/employees#>
DESCRIBE ?x WHERE { ?x ent:employeeId "1234" }
```

podría devolver una descripción de los empleados y otros detalles potencialmente útiles:

```
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
@prefix vcard:  <http://www.w3.org/2001/vcard-rdf/3.0> .
@prefix exOrg:  <http://org.example.com/employees#> .
@prefix rdf:    <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl:    <http://www.w3.org/2002/07/owl#>

_:a     exOrg:employeeId    "1234" ;

        foaf:mbox_sha1sum   "ABCD1234" ;
        vcard:N
         [ vcard:Family       "Smith" ;
           vcard:Given        "John"  ] .

foaf:mbox_sha1sum  rdf:type  owl:InverseFunctionalProperty .
```

que incluye el nodo en blanco determinado por el vocabulario vcard vcard:N. Otros posibles mecanismos para decidir qué información devolver incluyen las Descripciones Concisas Acotadas [CBD].

Para un vocabulario como FOAF, donde los recursos son típicamente nodos en blanco, devolver suficiente información para identificar un nodo como la propiedad funcional inversa `foaf:mbox_sha1sum` así como información como el nombre y otros detalles registrados podría resultar apropiado. En el ejemplo, se devuelve una concordancia con la cláusula WHERE, pero esto no es preciso.

Regla gramatical:

```
[7]   DescribeQuery  ::=  'DESCRIBE' ( VarOrIRIref+ | '*' )
                           DatasetClause*
                           WhereClause?
                           SolutionModifier
```

11 Comprobación de valores

Las cláusulas `FILTER` de SPARQL restringen las soluciones de una concordancia de patrón de grafo conforme a una determinada expresión. Específicamente, los términos `FILTER` eliminan todas las soluciones que, cuando se sustituye en la expresión, dan lugar a un valor efectivo booleano de `false` o producen un error. Los valores booleanos efectivos se definen en la Sección 11.2.2 *Valor booleano efectivo* y los errores en el apartado 2.3.1 *Tipos de errores* de la especificación XQuery 1.0: Un lenguaje de consulta XML [XQUERY]. Estos errores no tienen efectos fuera de la evaluación `FILTER`.

Los literales de RDF puede tener un tipo de datos IRI:

```
@prefix a:          <http://www.w3.org/2000/10/annotation-ns#> .
@prefix dc:         <http://purl.org/dc/elements/1.1/> .

_:a   a:annotates   <http://www.w3.org/TR/rdf-sparql-query/> .
_:a   dc:date       "2004-12-31T19:00:00-05:00" .

_:b   a:annotates   <http://www.w3.org/TR/rdf-sparql-query/> .
_:b   dc:date       "2004-12-31T19:01:00-05:00"^^<http://www.w3.org/
```

El objeto del primer triplete `dc:date` no tiene información de tipo. El segundo tiene un tipo de datos `xsd:dateTime`.

Las expresiones SPARQL se construyen de acuerdo a la gramática yproporcionan acceso a funciones (nombradas por la IRI) y operadores de funciones (invocadas mediante palabras clave y símbolos en la gramática SPARQL). Los operadores SPARQL pueden ser usados para comparar los valores de los literales tipados:

```
PREFIX a:      <http://www.w3.org/2000/10/annotation-ns#>
PREFIX dc:     <http://purl.org/dc/elements/1.1/>
PREFIX xsd:    <http://www.w3.org/2001/XMLSchema#>

SELECT ?annot
WHERE { ?annot  a:annotates  <http://www.w3.org/TR/rdf-sparql-query/
        ?annot  dc:date      ?date .
        FILTER ( ?date > "2005-01-01T00:00:00Z"^^xsd:dateTime ) }
```

Los operadores SPARQL se listan el la sección 11.3 y se asocian con sus producciones en la gramática.

Además, SPARQL permite invocar funciones arbitrarias, incluyendo un subconjunto de las funciones de conversión (casting functions) de XPath, listadas en la sección 11.5. Estas funciones son invocadas por su nombre (un IRI) dentro de una consulta SPARQL. Por ejemplo:

```
... FILTER ( xsd:dateTime(?date) < xsd:dateTime("2005-01-01T00:00:00
```

En esta sección se utilizan las convenciones de texto siguientes:

- Los operadores de XPath se etiquetan con el prefijo `op:`. Los operadores XPath no tienen ningún espacio de nombres; `op:` es una convención de etiquetado.
- Los operadores introducidos por esta especificación se indican con la clase (CSS) SPARQLoperator

11.1 Tipos de datos de los operandos

Las funciones y operadores SPARQL se aplican sobre términos RDF y variables SPARQL . Un subconjunto de estas funciones y operadores se toma del documento XQuery 1.0 y XPath 2.0 Funciones y Operadores [FUNCOP] y tienen como argumentos y tipos de valor devuelto el valor tipado de XML Schema. Los `literales tipados` RDF pasados como argumentos a estas funciones y operadores se convierten a valores tipados de XML Schema con un valor de cadena del tipo `forma léxica` y un tipo de datos atómico correspondiente al tipo de datos IRI. Los valores tipados devueltos se reconvierten de la misma forma a `literales tipados` RDF.

SPARQL tiene operadores adicionales que operan sobre los subconjuntos específicos de términos RDF. En referencia a un tipo, los siguientes términos denotan un `literal tipado` con el correspondiente tipo de datos IRI de XML Schema [XSDT]:

- `xsd:integer`
- `xsd:decimal`
- `xsd:float`
- `xsd:double`
- `xsd:string`
- `xsd:boolean`
- `xsd:dateTime`

Los siguientes términos identifican adicionalmente tipos usados en comprobaciones de valores en SPARQL:

- numérico denota `literales tipados` con tipos de datos `xsd:integer`, `xsd:decimal`, `xsd:float`, y `xsd:double`.
- literal simple denota un `literal simple` sin `etiqueta de idioma`.
- término RDF denota los tipos `IRI`, `literal`, y `nodo en blanco`.
- variable denota una variable SPARQL .

Los siguientes tipos se derivan de los tipos numéricos y son argumentos válidos para funciones y operadores que acepten argumentos numéricos :

- `xsd:nonPositiveInteger`
- `xsd:negativeInteger`
- `xsd:long`
- `xsd:int`
- `xsd:short`
- `xsd:byte`
- `xsd:nonNegativeInteger`
- `xsd:unsignedLong`
- `xsd:unsignedInt`
- `xsd:unsignedShort`
- `xsd:unsignedByte`
- `xsd:positiveInteger`

Las extensiones de lenguaje SPARQL pueden tratar tipos adicionales como derivados de tipos de datos de XML Schema.

11.2 Evaluación de filtros

SPARQL proporciona un subconjunto de funciones definidos por XQuery en la sección Correspondencia de Operadores. La sección 2.2.3 Procesamiento de expresiones de la especificación XQuery 1.0 describe la invocación de funciones XPath. Las siguientes reglas describen lasdiferencias en los modelos de datos y ejecución entre XQuery y SPARQL:

- A diferencia de XPath/XQuery, las funciones SPARQL no procesan secuencias de nodos. Al interpretar la semántica de las funciones XPath, asume que cada argumento es una secuencia de un solo nodo.
- Las functiones invocadas con un argumento de un tipo erróneo producirán un error de tipo. Los argumentos de valor booleano efectivo (etiquetados "xsd:boolean (EBV)" en la tabla de correspondencia de operadores siguiente), son forzados a `xsd:boolean` usando las reglas EBV de la sección 11.2.2.
- Además de la función BOUND, todas las funciones y operadores operan sobre términos RDF y producirán un error de tipo si algún argumento no está ligado (unbound).
- Cualquier expresión distinta de logical-or (`||`) o logical-and (`&&`) que encuentre un error producirá ese error.
- Un logical-or que encuentre un error en sólo una ramificación devolverá TRUE si la otra ramificación es TRUE y un error si la otra ramificación es FALSE.

- Un logical-and que encuentre un error en sólo una ramificación devolverá un error si la otra ramificación es TRUE y FALSE si la otra ramificación es FALSE.
- Un logical-or o logical-and que encuentre errores en ambas ramificaciones producirá *cualquiera* de los errores.

La tabla de verdad de logical-and y logical-or para verdadero (`T`), falso (`F`), y error (`E`) es la siguiente:

A	B	A || B	A && B
T	T	T	T
T	F	T	F
F	T	T	F
F	F	F	F
T	E	T	E
E	T	T	E
F	E	E	F
E	F	E	F
E	E	E	E

11.2.1 Invocación

SPARQL define una sintaxis para la invocación de funciones yoperadores sobre una lista de argumentos. Estos se invocan de la siguiente forma:

- Se evalúan las expresiones de argumentos, produciendo valores de argumentos. El orden de evaluación de argumentos no está definido.
- Los argumentos numéricos se promueven como sea necesario para satisfacer los tipos esperados por esa función u operador.
- Se invoca la función u operador sobre los valores de argumentos.

Si cualquiera de estos pasos falla, la invocación genera un error. Los efectos de los errores se definen en la sección 11.2 Evaluación de filtro.

11.2.2 Valor Booleano Efectivo

El valor efectivo booleano (effective boolean value), o valor EBV, se utiliza para calcular los argumentos de las funciones lógicas logical-and, logical-or, y fn:not, y para evaluar el resultado de una expresión `FILTER`.

Las reglas de Effective Boolean Value de XQuery se basan en la definición de fn:boolean de XPath. Las siguientes reglas reflejan las reglas las reglas para `fn:boolean` aplicadas a los tipos de argumentos presentes en las consultas de SPARQL:

- El valor EBV de cualquier literal cuyo tipo es `xsd:boolean` o numérico es falso si la forma léxica no es válida para ese tipo de datos (por ejemplo "abc"^^xsd:integer).
- Si el argumento es un literal tipado con un tipo de datos `xsd:boolean`, el valor EBV es el valor de ese argumento.
- Si el argumento es un literal simple o un literal tipado con un tipo de datos `xsd:string`, el valor EBV es falso si el valor del operando tiene una longitud cero; de lo contrario, el valor EBV es verdadero.
- Si el argumento es de un tipo numérico o un literal tipado con un tipo de datos derivado de un tipo numérico el valor EBV es falso si el valor del operandpo es "NaN" o numéricamente igual a cero; de lo contrario el valor EBV es true.
- Todos los otros argumentos, incluidos los argumentos no ligados, producirán un error de tipo.

Un valor EBV `true` se representa como un literal tipado con un tipo de datos `xsd:boolean` y un valor léxico de "true"; un EBV false se representa como un literal tipado con un tipo de datos `xsd:boolean` y un valor léxico de "false".

11.3 Conversión de Operadores

La gramática SPARQL identifica un conjunto de operadores (por ejemplo, &&, *, isIRI) utilizados para la construcción de restricciones. La siguiente tabla asocia cada una de estas producciones gramaticales con los operandos apropiados y una función de operador definida en el documento XQuery 1.0 y XPath 2.0 Funciones and Operadores [FUNCOP] o los operadores SPARQL especificados en la sección 11.4. En la selección de la definición del operador para un conjunto dado de parámetros, se aplica la definición con los parámetros más específicos. Por ejemplo, en la evaluación de `xsd:integer = xsd:signedInt`, se aplica la definición para `=` con dos parámetros `numeric`, en lugar de una con dos términos RDF. La tabla está dispuesta de forma que el candidato más viable es el más específico. Los operadores invocados sin los operandos apropiados producirán un error de tipo.

SPARQL sigue el esquema de XPath para las promociones de tipo numéricoy la sustitución de subtipo para los argumentos de los operadores numéricos. Las reglas de convesión de operadores de XPath para operandos numéricos (`xsd:integer`, `xsd:decimal`, `xsd:float`, `xsd:double`, y tipos derivados de un tipo numérico) se aplican también a los operadores SPARQL (véase la especificación XML Path Language (XPath) 2.0 [XPATH20] para definiciones de promociones de tipos numéricos y sustitución de subtipos). Algunos de los operadores se asocian con expresiones de función anidadas, por ejemplo `fn:not(op:numeric-equal(A, B))`. Tenga en cuenta que de acuerdo con las definiciones de XPath, `fn:not` y `op:numeric-equal` producirán un error si su argumento es un error.

La colación para la función `fn:compare` se define mediante XPath y se identifica mediante `http://www.w3.org/2005/xpath-functions/collation/codepoint`. Esta colación permite la comparación de cadenas basadas en valores de punto de código. La equivalencia de puntos de código de cadenas se puede probar con la equivalencia de término RDF.

Operadores Unarios SPARQL

Operador	Tipo(A)	Función	Tipo del resultado
Operadores Unarios XQuery			
! A	xsd:boolean (EBV)	fn:not(A)	xsd:boolean
+ A	numérico	op:numeric-unary-plus(A)	numerico
- A	numérico	op:numeric-unary-minus (A)	numerico
Comprobaciones SPARQL, definidas en la sección 11.4			
BOUND(A)	variable	bound(A)	xsd:boolean
isIRI(A) isURI(A)	término RDF	isIRI(A)	xsd:boolean
isBLANK(A)	término RDF	isBlank(A)	xsd:boolean
isLITERAL(A)	término RDF	isLiteral(A)	xsd:boolean
Descriptores de acceso SPARQL, definidos en la sección 11.4			
STR(A)	literal	str(A)	literal simple
STR(A)	IRI	str(A)	literal simple
LANG(A)	literal	lang(A)	literal simple
DATATYPE (A)	literal tipado	datatype(A)	IRI
DATATYPE (A)	literal simple	datatype(A)	IRI

Operadores SPARQL Binarios

Operador	Tipo(A)	Tipo(B)	Función	Tipo de resultado
Conectivas Lógicas, definidas en la sección 11.4				
A || B	xsd:boolean (EBV)	xsd:boolean (EBV)	logical-or(A, B)	xsd:boolean
A && B	xsd:boolean (EBV)	xsd:boolean (EBV)	logical-and(A, B)	xsd:boolean
Comprobaciones XPath				
A = B	numérico	numérico	op:numeric-equal (A, B)	xsd:boolean
A = B	literal simple	literal simple	op:numeric-equal (fn:compare(A, B), 0)	xsd:boolean
A = B	xsd:string	xsd:string	op:numeric-equal (fn:compare(STR (A), STR(B)), 0)	xsd:boolean
A = B	xsd:boolean	xsd:boolean	op:boolean-equal (A, B)	xsd:boolean
A = B	xsd:dateTime	xsd:dateTime		xsd:boolean

			op:dateTime-equal(A, B)	
A != B	numérico	numérico	fn:not(op:numeric-equal(A, B))	xsd:boolean
A != B	literal simple	literal simple	fn:not(op:numeric-equal(fn:compare(A, B), 0))	xsd:boolean
A != B	xsd:string	xsd:string	fn:not(op:numeric-equal(fn:compare(STR(A), STR(B)), 0))	xsd:boolean
A != B	xsd:boolean	xsd:boolean	fn:not(op:boolean-equal(A, B))	xsd:boolean
A != B	xsd:dateTime	xsd:dateTime	fn:not(op:dateTime-equal(A, B))	xsd:boolean
A < B	numérico	numérico	op:numeric-less-than(A, B)	xsd:boolean
A < B	literal simple	literal simple	op:numeric-equal(fn:compare(A, B), -1)	xsd:boolean
A < B	xsd:string	xsd:string	op:numeric-equal(fn:compare(STR(A), STR(B)), -1)	xsd:boolean
A < B	xsd:boolean	xsd:boolean	op:boolean-less-than(A, B)	xsd:boolean
A < B	xsd:dateTime	xsd:dateTime	op:dateTime-less-than(A, B)	xsd:boolean
A > B	numérico	numérico	op:numeric-greater-than(A, B)	xsd:boolean
A > B	literal simple	literal simple	op:numeric-equal(fn:compare(A, B), 1)	xsd:boolean
A > B	xsd:string	xsd:string	op:numeric-equal(fn:compare(STR(A), STR(B)), 1)	xsd:boolean
A > B	xsd:boolean	xsd:boolean	op:boolean-greater-than(A, B)	xsd:boolean
A > B	xsd:dateTime	xsd:dateTime	op:dateTime-greater-than(A, B)	xsd:boolean
A <= B	numérico	numérico	logical-or(op:numeric-less-than(A, B), op:numeric-equal(A, B))	xsd:boolean
A <= B	literal simple	literal simple	fn:not(op:numeric-equal(fn:compare(A, B), 1))	xsd:boolean
A <= B	xsd:string	xsd:string	fn:not(op:numeric-equal(fn:compare	xsd:boolean

			(STR(A), STR(B)), 1))	
A <= B	xsd:boolean	xsd:boolean	fn:not(op:boolean-greater-than(A, B))	xsd:boolean
A <= B	xsd:dateTime	xsd:dateTime	fn:not (op:dateTime-greater-than(A, B))	xsd:boolean
A >= B	numérico	numérico	logical-or (op:numeric-greater-than(A, B), op:numeric-equal(A, B))	xsd:boolean
A >= B	literal simple	literal simple	fn:not(op:numeric-equal(fn:compare (A, B), -1))	xsd:boolean
A >= B	xsd:string	xsd:string	fn:not(op:numeric-equal(fn:compare (STR(A), STR(B)), -1))	xsd:boolean
A >= B	xsd:boolean	xsd:boolean	fn:not(op:boolean-less-than(A, B))	xsd:boolean
A >= B	xsd:dateTime	xsd:dateTime	fn:not (op:dateTime-less-than(A, B))	xsd:boolean
Aritmética XPath				
A * B	numérico	numérico	op:numeric-multiply(A, B)	numérico
A / B	numérico	numérico	op:numeric-divide (A, B)	numérico; pero xsd:decimal si ambos operandos son xsd:integer
A + B	numérico	numérico	op:numeric-add (A, B)	numérico
A - B	numérico	numérico	op:numeric-subtract(A, B)	numérico
Tests SPARQL, definidos en la sección 11.4				
A = B	término RDF	término RDF	RDFterm-equal(A, B)	xsd:boolean
A != B	término RDF	término RDF	fn:not(RDFterm-equal(A, B))	xsd:boolean
sameTERM(A)	término RDF	término RDF	sameTerm(A, B)	xsd:boolean
langMATCHES (A, B)	literal simple	literal simple	langMatches(A, B)	xsd:boolean
REGEX (STRING, PATTERN)	literal simple	literal simple	fn:matches (STRING, PATTERN)	xsd:boolean

SPARQL Trinary Operators

Operador	Tipo (A)	Tipo (B)	Tipo (C)	Función	Tipo de resultado
Tests SPARQL Tests, definidos en la sección 11.4					
REGEX(STRING, PATTERN, FLAGS)	literal simple	literal simple	literal simple	fn:matches (STRING, PATTERN, FLAGS)	xsd:boolean

Los argumentos de función de tipo xsd:boolean marcados con "(EBV)" son forzados a xsd:boolean por la evaluación del valor booleano efectivo de ese argumento.

11.3.1 Extensibilidad de operadores

Las extensiones del lenguaje SPARQL pueden proporcionar conversiones adicionales entre operadores y operadores de funciones; esto equivale a agregar filas a la tabla anterior. Ningún operador adicional puede producir un resultado que sustituya a cualquier otro resultado, a excepción de un error de tipo, en la semántica definida anteriormente. La consecuencia de esta regla es que las extensiones SPARQL producirán *al menos* las mismas soluciones que una implementación no extendida, y pueden, en algunas consultas, producir más soluciones.

Las conversiones adicionales del operador '<' están destinadas a controlar el orden relativo de los operandos, en particular, cuando se utiliza en una cláusula `ORDER BY`.

11.4 Definición de Operadores

Esta sección define los operadores introducidos por el Lenguaje deconsulta SPARQL. Los ejemplos muestran el comportamiento de los operadores que se invocan por las construcciones gramaticales apropiadas.

11.4.1 bound

```
xsd:boolean   BOUND (variable var)
```

Devuelve `true` si `var` está ligado a un valor. Devuelve false en caso contrario. Las variables con el valor NaN o INF se consideran ligadas.

Datos:

```
@prefix foaf:        <http://xmlns.com/foaf/0.1/> .
@prefix dc:          <http://purl.org/dc/elements/1.1/> .
@prefix xsd:         <http://www.w3.org/2001/XMLSchema#> .

_:a  foaf:givenName  "Alice".

_:b  foaf:givenName  "Bob" .
_:b  dc:date         "2005-04-04T04:04:04Z"^^xsd:dateTime .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc:   <http://purl.org/dc/elements/1.1/>
PREFIX xsd:   <http://www.w3.org/2001/XMLSchema#>
SELECT ?name
 WHERE { ?x foaf:givenName  ?givenName .
         OPTIONAL { ?x dc:date ?date } .
         FILTER ( bound(?date) ) }
```

Resultado de la consulta:

givenName
"Bob"

Podemos comprobar que un patrón de grafo no se expresa por la definición de un patrón de grafo `OPTIONAL` que introduce una variable y comprobar que la variable no está ligada (`not bound`). Esto se llama "negación por fracaso" (*Negation as Failure*) en programación lógica.

Esta consulta filtra las personas con un nombre (`name`) pero sin fecha (`date`) expresada:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc:   <http://purl.org/dc/elements/1.1/>
SELECT ?name
 WHERE { ?x foaf:givenName  ?name .
         OPTIONAL { ?x dc:date ?date } .
         FILTER (!bound(?date)) }
```

Resultado de la consulta:

name
"Alice"

Como la fecha (`dc:date`) de Bob era conocida, `"Bob"` no era una solución para la consulta.

11.4.2 isIRI

```
 xsd:boolean   ISIRI (RDF term term)
 xsd:boolean   ISURI (RDF term term)
```

Devuelve `true` si `term` es un IRI. Devuelve `false` en caso contrario. `ISURI` es una alternativa ortográfica del operador `ISIRI`.

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name       "Alice".
_:a  foaf:mbox       <mailto:alice@work.example> .

_:b  foaf:name       "Bob" .
_:b  foaf:mbox       "bob@work.example" .
```

Esta consulta filtra las personas con un nombre (`name`) y un buzón de correo (`mbox`) que es un IRI:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
 WHERE { ?x foaf:name  ?name ;
            foaf:mbox  ?mbox .
         FILTER isIRI(?mbox) }
```

Resultado de la consulta:

name	mbox
"Alice"	<mailto:alice@work.example>

11.4.3 isBlank

```
 xsd:boolean    ISBLANK (RDF term term)
```

Devuelve `true` si `term` es un nodo en blanco. Devuelve `false` en caso contrario.

```
@prefix a:          <http://www.w3.org/2000/10/annotation-ns#> .
@prefix dc:         <http://purl.org/dc/elements/1.1/> .
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .

_:a   a:annotates   <http://www.w3.org/TR/rdf-sparql-query/> .
_:a   dc:creator    "Alice B. Toeclips" .

_:b   a:annotates   <http://www.w3.org/TR/rdf-sparql-query/> .
_:b   dc:creator    _:c .
_:c   foaf:given    "Bob".
_:c   foaf:family   "Smith".
```

Esta consulta filtra a las personas con un `dc:creator` que usa predicados del vocabulario FOAF para expresar el nombre:

```
PREFIX a:      <http://www.w3.org/2000/10/annotation-ns#>
PREFIX dc:     <http://purl.org/dc/elements/1.1/>
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>

SELECT ?given ?family
 WHERE { ?annot  a:annotates  <http://www.w3.org/TR/rdf-sparql-query
         ?annot  dc:creator   ?c .
         OPTIONAL { ?c  foaf:given   ?given ; foaf:family  ?family }
         FILTER isBlank(?c)
       }
```

Resultado de la consulta:

given	family
"Bob"	"Smith"

En este ejemplo, hay dos objetos de predicados `foaf:knows`, pero solo uno (`_:c`) era un nodo en blanco.

11.4.4 isLiteral

```
xsd:boolean   ISLITERAL (RDF term term)
```

Devuelve `true` if `term` is a literal. Devuelve `false` en caso contrario.

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name       "Alice".
_:a  foaf:mbox       <mailto:alice@work.example> .

_:b  foaf:name       "Bob" .
_:b  foaf:mbox       "bob@work.example" .
```

Esta consulta es similar a la de la sección 11.4.2 excepto que filtra personas con un nombre (`name`) y un buzón de correo (`mbox`) que es un literal. Podría usarse para buscar datos erróneos. (`foaf:mbox` puede tener sólo una IRI como su objeto).

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
 WHERE { ?x foaf:name  ?name ;
           foaf:mbox  ?mbox .
         FILTER isLiteral(?mbox) }
```

Resultados de la consulta:

name	mbox
"Bob"	"bob@work.example"

11.4.5 str

```
literal simple   STR (literal ltrl)
literal simple   STR (IRI rsrc)
```

Devuelve the `forma lexica` de `ltrl` (un literal); Devuelve la representación de punto de código de `rsrc` (una IRI). Es útil para examinar partes de una IRI, por ejemplo, el nombre del host.

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name       "Alice".
_:a  foaf:mbox       <mailto:alice@work.example> .

_:b  foaf:name       "Bob" .
_:b  foaf:mbox       <mailto:bob@home.example> .
```

Esta consulta selecciona las personas que usan su dirección `work.example` en su perfil foaf:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
 WHERE { ?x foaf:name  ?name ;
           foaf:mbox  ?mbox .
         FILTER regex(str(?mbox), "@work.example") }
```

Resultados de la consulta:

name	mbox
"Alice"	<mailto:alice@work.example>

11.4.6 lang

```
 literal simple   LANG (literal ltrl)
```

Devuelve la `etiqueta de idioma` de `ltrl`, si la tiene. Devuelve `""` si `ltrl` no tiene `etiqueta de idioma`. Tenga en cuenta que el modelo de datos RDF no incluye literales con una `etiqueta de idioma` vacía.

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name       "Robert"@EN.
_:a  foaf:name       "Roberto"@ES.
_:a  foaf:mbox       <mailto:bob@work.example> .
```

Esta consulta encuentra el `foaf:name` español y `foaf:mbox`:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
 WHERE { ?x foaf:name  ?name ;
           foaf:mbox  ?mbox .
         FILTER ( lang(?name) = "ES" ) }
```

Resultados de la consulta:

name	mbox
"Roberto"@ES	<mailto:bob@work.example>

11.4.7 datatype

```
 IRI   DATATYPE (typed literal typedLit)
 IRI   DATATYPE (literal simple simpleLit)
```

Devuelve el `tipo de datos IRI` de `typedLit`; Devuelve `xsd:string` si el parámetro es un literal simple.

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .
@prefix eg:         <http://biometrics.example/ns#> .
@prefix xsd:        <http://www.w3.org/2001/XMLSchema#> .

_:a  foaf:name       "Alice".
_:a  eg:shoeSize     "9.5"^^xsd:float .

_:b  foaf:name       "Bob".
_:b  eg:shoeSize     "42"^^xsd:integer .
```

Esta consulta encuentra el `foaf:name` y el `foaf:shoeSize` de cualquier persona con una talla de pie que sea un entero:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>
PREFIX eg:   <http://biometrics.example/ns#>
SELECT ?name ?shoeSize
 WHERE { ?x foaf:name  ?name ; eg:shoeSize  ?shoeSize .
         FILTER ( datatype(?shoeSize) = xsd:integer ) }
```

Resultados de la consulta:

name	shoeSize
"Bob"	42

11.4.8 logical-or

```
xsd:boolean   xsd:boolean left || xsd:boolean right
```

Devuelve un `OR` lógico de `left` y `right`. Tenga en cuenta que `logical-or` opera sobre el valor booleano efectivo de sus argumentos.

Nota: véase la sección 11.2, Evaluación de filtros, para el tratamiento de errores del operador `||`.

11.4.9 logical-and

```
xsd:boolean   xsd:boolean left && xsd:boolean right
```

Devuelve un `AND` lógico de `left` y `right`. Tenga en cuenta que `logical-and` opera sobre el valor booleano efectivo de sus argumentos.

Nota: véase la sección 11.2, Evaluación de filtros, para el tratamiento de errores del operador `&&`.

11.4.10 RDFterm-equal

```
xsd:boolean   RDF term term1 = RDF term term2
```

Devuelve `true` si `term1` y `term2` son el mismo término RDF según se define en Resource Description Framework (RDF): Conceptos y sintaxis abstracta [CONCEPTS]; produce un error de tipo si los argumentos son ambos literales pero no son el mismo término RDF [*]; Devuelve `false` en caso contrario. `term1` y `term2` son lo mismo si cualquiera de lo siguiente es cierto:

- `term1` y `term2` son IRIs equivalentes como se define en 6.4 RDF URI References de [CONCEPTS].
- `term1` y `term2` son literales equivalentes como se define en 6.5.1 Literal Equality de [CONCEPTS].
- `term1` y `term2` son el mismo nodo en blanco como se describe en 6.6 Blank Nodes de [CONCEPTS].

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name       "Alice".
_:a  foaf:mbox       <mailto:alice@work.example> .

_:b  foaf:name       "Ms A.".
_:b  foaf:mbox       <mailto:alice@work.example> .
```

Esta consulta encuentra las personas que tienen múltiples tripletas `foaf:name`:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name1 ?name2
 WHERE { ?x foaf:name  ?name1 ;
            foaf:mbox  ?mbox1 .
         ?y foaf:name  ?name2 ;
            foaf:mbox  ?mbox2 .
         FILTER (?mbox1 = ?mbox2 && ?name1 != ?name2)
       }
```

Resultados de la consulta:

name1	name2
"Alice"	"Ms A."
"Ms A."	"Alice"

En esta consulta de documentos que fueron anotados el día de Año nuevo (2004 o 2005), los términos RDF no son los mismos, pero tienen valores equivalentes:

```
@prefix a:          <http://www.w3.org/2000/10/annotation-ns#> .
@prefix dc:         <http://purl.org/dc/elements/1.1/> .

_:b   a:annotates   <http://www.w3.org/TR/rdf-sparql-query/> .
_:b   dc:date       "2004-12-31T19:00:00-05:00"^^<http://www.w3.org/
```

```
PREFIX a:      <http://www.w3.org/2000/10/annotation-ns#>
PREFIX dc:     <http://purl.org/dc/elements/1.1/>
PREFIX xsd:    <http://www.w3.org/2001/XMLSchema#>

SELECT ?annotates
WHERE { ?annot  a:annotates  ?annotates .
        ?annot  dc:date      ?date .
        FILTER ( ?date = xsd:dateTime("2005-01-01T00:00:00Z") ) }
```

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

* La invocación de RDFterm-equal sobre dos literales tipados evalúa la equivalencia de valores. Una implementación extendida puede dar soporte a tipos de datos adicionales. Una implementación que procese una consulta que compruebe la equivalencia sobre tipos de datos no soportados (y forma léxica no idéntica y tipo de datos IRI) devuelve un error, indicando que es incapaz de determinar si los valores son o no equivalentes. Por ejemplo, una implementación no extendida producirá un error cuando evalúe tanto `"iiii"^^my:romanNumeral = "iv"^^my:romanNumeral` como `"iiii"^^my:romanNumeral != "iv"^^my:romanNumeral`.

11.4.11 sameTerm

```
xsd:boolean   SAMETERM (RDF term term1, RDF term term2)
```

Devuelve `true` si `term1` y `term2` son el mismo término RDF según se define en Resource Description Framework (RDF): Conceptos y sintaxis abstracta [CONCEPTS]; devuelve `false` en caso contrario.

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name       "Alice".
_:a  foaf:mbox       <mailto:alice@work.example> .

_:b  foaf:name       "Ms A.".
_:b  foaf:mbox       <mailto:alice@work.example> .
```

Esta consulta encuentra las personas que tienen múltiples tripletas `foaf:name`:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name1 ?name2
 WHERE { ?x foaf:name  ?name1 ;
            foaf:mbox  ?mbox1 .
         ?y foaf:name  ?name2 ;
            foaf:mbox  ?mbox2 .
         FILTER (sameTerm(?mbox1, ?mbox2) && !sameTerm(?name1, ?name
       }
```

Resultados de la consulta:

name1	name2
"Alice"	"Ms A."
"Ms A."	"Alice"

A diferencia de `RDFterm-equal`, `sameTerm` puede ser usado para evaluar la no equivalencia de literales tipados con tipos de datos no soportados:

```
@prefix :           <http://example.org/WMterms#> .
@prefix t:          <http://example.org/types#> .

_:c1  :label         "Container 1" .
_:c1  :weight        "100"^^t:kilos .
_:c1  :displacement  "100"^^t:liters .

_:c2  :label         "Container 2" .
_:c2  :weight        "100"^^t:kilos .
_:c2  :displacement  "85"^^t:liters .

_:c3  :label         "Container 3" .
_:c3  :weight        "85"^^t:kilos .
_:c3  :displacement  "85"^^t:liters .
```

```
PREFIX  :       <http://example.org/WMterms#>
PREFIX  t:      <http://example.org/types#>

SELECT ?aLabel1 ?bLabel
WHERE { ?a  :label         ?aLabel .
        ?a  :weight        ?aWeight .
        ?a  :displacement ?aDisp .

        ?b  :label         ?bLabel .
        ?b  :weight        ?bWeight .
        ?b  :displacement ?bDisp .

        FILTER ( sameTerm(?aWeight, ?bWeight) && !sameTerm(?aDisp, ?
```

aLabel	bLabel
"Container 1"	"Container 2"
"Container 2"	"Container 1"

La comprobación de cajas con el mismo peso también puede hacerse con el operador '=' (RDFterm-equal) puesto que la evaluación de `100"^^t:kilos = "85"^^t:kilos` producirá un error, eliminando esa potencial solución.

11.4.12 langMatches

```
xsd:boolean   LANGMATCHES (literal simple language-tag, literal simp
```

Devuelve `true` si `language-tag` (primer argumento) se corresponde con `language-range` (segundo argumento) según el esquema básico de filtrado definido en [RFC4647] sección 3.3.1. El argumento `language-range` es un rango de idioma de base según el documento sobre Concordancia de

etiquetas de idioma [RFC4647] sección 2.1. Un `language-range` de "*" se corresponde con cualquier cadena `language-tag` no vacía.

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

_:a  dc:title         "That Seventies Show"@en .
_:a  dc:title         "Cette Série des Années Soixante-dix"@fr .
_:a  dc:title         "Cette Série des Années Septante"@fr-BE .
_:b  dc:title         "Il Buono, il Bruto, il Cattivo" .
```

Esta consulta usa `langMatches` y `lang` (descrito en la sección 11.2.3.8) para encontrar los títulos en francés para el espectáculo conocido en inglés como "That Seventies Show":

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title
 WHERE { ?x dc:title  "That Seventies Show"@en ;
            dc:title  ?title .
         FILTER langMatches( lang(?title), "FR" ) }
```

Resultados de la consulta:

title
"Cette Série des Années Soixante-dix"@fr
"Cette Série des Années Septante"@fr-BE

El idioma `langMatches( lang( ?v ), "*" )` no filtrará los literales sin una etiqueta de idioma puesto que `lang( ?v )` devolverá una cadena vacía, por lo que

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title
 WHERE { ?x dc:title  ?title .
         FILTER langMatches( lang(?title), "*" ) }
```

informará de todos los títulos con una etiqueta de idioma:

title
"That Seventies Show"@en
"Cette Série des Années Soixante-dix"@fr
"Cette Série des Années Septante"@fr-BE

11.4.13 regex

```
 xsd:boolean   REGEX (literal simple text, literal simple pattern)
 xsd:boolean   REGEX (literal simple text, literal simple pattern, l
```

Invoca la función fn:matches de XPath para filtrar el argumento `text` contra una expresión regular `pattern`. El lenguaje de expresiones regulares se define

en la especificación XQuery 1.0 y XPath 2.0 Funciones y operadores [FUNCOP], sección 7.6.1 Sintaxis de Expresiones Regulares [FUNCOP].

```
@prefix foaf:       <http://xmlns.com/foaf/0.1/> .

_:a  foaf:name       "Alice".
_:b  foaf:name       "Bob" .
```

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
 WHERE { ?x foaf:name  ?name
         FILTER regex(?name, "^ali", "i") }
```

Resultados de la consulta:

name
"Alice"

11.5 Funciones constructoras

SPARQL importa un subconjunto de funciones constructoras de XPath definidas en XQuery 1.0 y XPath 2.0 Funciones y operadores [FUNCOP], sección 17.1 Conversión (casting) de tipos primitivos a tipos primitivos. Los constructores SPARQL comprenden todos los constructores XPath para los tipos de datos de operandos SPARQL más los tipos de datos adicionales impuestos por el modelo de datos RDFI. La conversión en SPARQL se realiza invocando a la función constructora para el tipo destino sobre un operando del tipo origen.

XPath define sólo las conversiones desde un tipo de datos XML Schema a otro. El resto de conversiones se definen de la siguiente forma:

- La conversión de un IRI a un `xsd:string` produce un literal tipado con una forma léxica de los puntos de código que comprenden la IRI, y un tipo de datos de `xsd:string`.
- La conversión de un literal simple a cualquier tipo de datos XML Schema se define como el producto de la conversión de un `xsd:string` con el valor de cadena igual a la forma léxica del literal al tipo de datos destino.

La siguiente tabla resume las operaciones de conversión que siempre se permiten (Y), nunca se permiten (N) y dependen del valor de la froma léxica (M). Por ejemplo, una operación de conversión desde un `xsd:string` (la primera fila) a un `xsd:float` (la segunda columna) es dependiente del valor de la forma léxica (M).

bool = xsd:boolean
dbl = xsd:double
flt = xsd:float
dec = xsd:decimal
int = xsd:integer
dT = xsd:dateTime
str = xsd:string

IRI = IRI
ltrl = `literal simple`

Desde \ A	str	flt	dbl	dec	int	dT	bool
str	Y	M	M	M	M	M	M
flt	Y	Y	Y	M	M	N	Y
dbl	Y	Y	Y	M	M	N	Y
dec	Y	Y	Y	Y	Y	N	Y
int	Y	Y	Y	Y	Y	N	Y
dT	Y	N	N	N	N	Y	N
bool	Y	Y	Y	Y	Y	N	Y
IRI	Y	N	N	N	N	N	N
ltrl	Y	M	M	M	M	M	M

11.6 Comprobación de valores extensibles

Una regla gramatical PrimaryExpression puede ser una llamada a una función de extensión nombrada por una IRI. Una función de extensión toma un cierto numero de términos RDF como argumentos y devuelve un término RDF. Las semánticas de estas funciones se identifican mediante la IRI que identifica la función.

La interoperabilidad de las consultas SPARQL que usan funciones de extensión será problamente limitada.

Como ejemplo, considere una función llamada `func:even`:

```
xsd:boolean   func:even (numeric value)
```

Esta función será invocada en un filtro (FILTER) como:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX func: <http://example.org/functions#>
SELECT ?name ?id
WHERE { ?x foaf:name  ?name ;
          func:empId   ?id .
        FILTER (func:even(?id)) }
```

Como segundo ejemplo, considere una función `aGeo:distance` que calcula la distancia entre dos puntos, que se usa aquí para encontrar lugares cerca de Grenoble:

```
xsd:double   aGeo:distance (numeric x1, numeric y1, numeric x2, nume
```

```
PREFIX aGeo: <http://example.org/geo#>

SELECT ?neighbor
WHERE { ?a aGeo:placeName "Grenoble" .
        ?a aGeo:location ?axLoc .
        ?a aGeo:location ?ayLoc .

        ?b aGeo:placeName ?neighbor .
        ?b aGeo:location ?bxLoc .
        ?b aGeo:location ?byLoc .

        FILTER ( aGeo:distance(?axLoc, ?ayLoc, ?bxLoc, ?byLoc) < 10
      }
```

Una función de extensión puede usarse para evaluar algunos tipos de datos de aplicación no soportados por la especificación base de SPARQL; podría ser una transformación entre formatos de tipos de datos, por ejemplo a un término RDF XSDdateTime desde otro formato de fecha.

12 Definición of SPARQL

Esta sección define el comportamiento correcto para la evaluación de los partrones de grafo y modificadores de solución, dadas una cadena de consulta y un conjunto de datos RDF. Esto no implica que una implementación de SPARQL deba utilizar el proceso definido aquí.

El resultado de una ejecución SPARQL se define por una serie de pasos, comenzando por la consulta SPARQL como una cadena, convirtiendo dicha cadena en una forma de sintaxis abstracta, y convirtiéndola a su vez en una consulta abstracta SPARQL que incluye los operadores del álgebra de SPARQL. Esta consulta abstracta es evaluada sobre un conjunto de datos RDF.

12.1 Definiciones iniciales

12.1.1 Términos RDF

SPARQL está definido en términos de IRIs [RFC3987]. Las referencias IRIs son un subcojunto de las referencias URI RDF que omite los espacios.

Definición: Término RDF

Sea I el conjunto de todas las referencias IRIs.
Sea RDF-L el conjunto de todos los Literales RDF
Sea RDF-B el conjunto de todos los nodos en blanco de los grafos RDF

El conjunto de los **Términos RDF**, RDF-T, es la unión de I, RDF-L y RDF-B.

Esta definición de **Término RDF** recoge de forma conjunta varias nociones básicas del modelo de datos RDF, pero actualizada para referirse a IRIs en vez de referencias URI RDF.

12.1.2 Conjunto de datos RDF

> **Definición:** Conjunto de datos RDF
>
> Un conjunto de datos RDF es un conjunto:
> { G, (<u_1>, G_1), (<u_2>, G_2), . . . (<u_n>, G_n) }
> donde G y cada G_i son grafos, y cada <u_i> es una IRI. Cada <u_i> es diferente.
>
> G se denomina grafo por defecto. (<u_i>, G_i) se denomina grafo con nombre.

> **Definición:** Grafo activo
>
> El **grafo activo** es el grafo del conjunto de datos utilizado para la concordancia con patrones de grafo básicos.

12.1.3 Variables de consulta

> **Definición:** Variable de consulta
>
> Una **variable de consulta** es un miembro del conjunto V, donde V es infinito y disjunto con respecto a RDF-T.

12.1.4 Patrones de tripleta

> **Definición:** Patrón de tripleta
>
> Un **patrón de tripleta** es miembro del conjunto:
> (RDF-T unión V) x (I unión V) x (RDF-T unión V)

Esta definición de patrón de tripleta incluye sujetos literales. Esto ya ha sido señalado en los fundamentos de RDF

```
"[El grupo de trabajo sobre el núcleo de RDF] señala que no hay c
  ser sujetos y un futuro grupo de trabajo con un carácter menos
  extender la sintaxis para permitir literales como subjetos de l
```

Debido a que los grafos RDF pueden no contener literales como subjetos, cualquier patrón de tripleta SPARQL con un literal como sujeto fallará en la concordancia sobre cualquier grafo RDF.

12.1.5 Patrones de grafo básicos

> **Definición:** Patron de grafo básico
>
> Un **patrón de grafo básico** es un conjunto de patrones de tripleta.

El patrón de grafo vacío es un patrón de grafo básico que consiste en el conjunto vacío.

12.1.6 Correspodencia de soluciones

Una correspondencia de solución es un correspondencia a partir de un conjunto de variables con un conjunto de términos RDF. Se utiliza la expresión "solución" allí donde está claro.

> **Definición: Correspondencia de solución**
>
> Una **correspondencia de solución**, μ, es una función parcial μ : V -> T.
>
> El dominio de μ, dom(μ), es el subconjunto de V donde μ esté definida.

> **Definición: Secuencia de solución**
>
> Una **secuencia de solución** es una lista de soluciones ordenadas posibles.

12.1.7 Modificadores de secuencia de solución

Esta sección define el proceso de conversión de patrones de grafo y modificadores de solución en una cadena para una consulta SPARQL mediante una expresión del álgebra de SPARQL.

Despues de analizar una cadena de una consulta SPARQL y aplicar las abreviaturas para IRIs y patrones de tripleta mostradas en la sección 4, hay un árbol de sintaxis abstracta compuesta de:

Patrones	Modificadores	Formas de consulta
Términos RDF	DISTINCT	SELECT
Patrones de tripleta	REDUCED	CONSTRUCT
Patrones de grafo básicos	PROJECT	DESCRIBE
Grupos	ORDER BY	ASK
OPTIONAL	LIMIT	
UNION	OFFSET	
GRAPH		
FILTER		

El resultado de la conversión como un arbol abstracto de sintaxis abstracta es una consulta SPARQL que usa los siguientes símbolos del álgebra SPARQL:

Patrón de grafo	Modificadores de solución
BGP	ToList
Join	OrderBy
LeftJoin	Project
Filter	Distinct
Union	Reduced
Graph	Slice

Slice (porción) es la combinación de OFFSET y LIMIT. *mod* es cualquier de los modificadores de solución.

ToList es utilizado donde la conversión de los resultados del patrón de grafo concuerdan con las secuencias.

Definición: Consulta SPARQL

Una **consulta abstracta SPARQL** es una tupla (E, DS, R) donde:

- E es una expresión del álgebra de SPARQL
- DS es un conjunto de datos RDF
- R es una forma de consulta

12.2.1 Conversión de patrones de grafo

Esta sección describe el proceso de traducir un patrón de grafo SPARQL en una expresión del álgebra SPARQL. Después de traducir las abreviaturas sintácticas para IRIs y patrones de tripleta, se procesarán recursivamente las formas sintácticas en expresiones algebráicas:

El grupo de trabajo observa que aplicar este punto en el paso de simplificación conduce a una transformación ambígua de las consultas The working group notes that the point at the simplification step is applied leads to ambiguous transformation of queries conllevando la doble necesidad de un filtro y un patrón en una cláusula`OPTIONAL`:

```
OPTIONAL { { ... FILTER ( ... ?x ... ) } }..
```

Esto es ilustrado por dos casos de prueba no normativos:

- Simplificación aplicada después del proceso de transformación
- Simplificación aplicada durante la transformación.

Primero, abreviaturas expandidas para IRIs y patrones de tripleta de la seccion 4.

WhereClause consiste en un patrón de grafo de grupo, GroupGraphPattern, que está compuesto por las siguientes formas de sintaxis:

- TriplesBlock
- Filter
- OptionalGraphPattern
- GroupOrUnionGraphPattern
- GraphGraphPattern

Cada forma está traducida por el siguiente procedimiento

Transform(forma de sintaxis)

Si la forma es TriplesBlock

```
El resultado es BGP(lista de patrones de tripleta)
```

Si la forma es GroupOrUnionGraphPattern

```
Sea A := indefinido

Para cada elemento G en GroupOrUnionGraphPattern
    Si A es indefinido
        A := Transform(G)
    Si no Entonces
        A := Union(A, Transform(G))

El resultado es A
```

Si la forma es GraphGraphPattern

```
Si la forma es GRAPH IRI GroupGraphPattern
    El resultado es Graph(IRI, Transform(GroupGraphPattern))
Si la forma es GRAPH Var GroupGraphPattern
    El resultado es Graph(Var, Transform(GroupGraphPattern))
```

Si la forma es GroupGraphPattern

Se introducen los siguientes símbolos:

- Join(Pattern, Pattern)
- LeftJoin(Pattern, Pattern, expression)
- Filter(expression, Pattern)

```
Sea FS := el conjunto vacío
Sea G := el patrón vacío, Z, un patrón de grafo básico que es el con

Para cada elemento E en GroupGraphPattern
   Si E es de la forma FILTER(expr)
       FS := FS set-union {expr}
   Si E es de la forma OPTIONAL{P}
   Entonces
       Sea A := Transform(P)
       Si A es de la forma Filter(F, A2)
           G := LeftJoin(G, A2, F)
       Si no Entonces
           G := LeftJoin(G, A, true)
   Si E es de cualquier otra forma:
      Sea A := Transform(E)
      G := Join(G, A)


Si FS no está vacío:
  Sea X := Conjunción de las expresiones en FS
  G := Filter(X, G)

El resultado es G.
```

Paso de simplificación:

Grupos de un patrón de grafo (no un filtro) se convierten en join(Z, A) y puede sustituirse por A. El patrón de grafo vacío Z es la identidad para la unión:

```
Sustituir join(Z, A) por A
Sustituir join(A, Z) por A
```

12.2.2 Ejemplos de correspondencias entre patrones de grafo

La segunda forma de un ejemplo de reescritura es como el primero, pero con los grupos vacíos unidos eliminados a través del paso de simplificación.

Ejemplo: grupo con un patrón de grafo básico que consiste en un único patrón de tripleta:

```
{ ?s ?p ?o }

Join(Z, BGP(?s ?p ?o) )

BGP(?s ?p ?o)
```

Ejemplo: grupo con un patrón de grafo básico que consiste en dos patrones de tripleta:

```
{ ?s :p1 ?v1 ; :p2 ?v2 }

BGP( ?s :p1 ?v1 .?s :p2 ?v2 )
```

Ejemplo: grupo formado por la unión de dos patrones de grafo básicos:

```
{ { ?s :p1 ?v1 } UNION {?s :p2 ?v2 } }

Union(Join(Z, BGP(?s :p1 ?v1)),
      Join(Z, BGP(?s :p2 ?v2)) )

Union( BGP(?s :p1 ?v1) , BGP(?s :p2 ?v2) )
```

Ejemplo: grupo formado por la unión de una unión y un patrón de grafo básico:

```
{ { ?s :p1 ?v1 } UNION {?s :p2 ?v2 } UNION {?s :p3 ?
v3 } }

Union(
    Union( Join(Z, BGP(?s :p1 ?v1)),
           Join(Z, BGP(?s :p2 ?v2))) ,
    Join(Z, BGP(?s :p3 ?v3)) )

Union(
    Union( BGP(?s :p1 ?v1) ,
           BGP(?s :p2 ?v2),
    BGP(?s :p3 ?v3))
```

Ejemplo: grupo formado por un patrón de grafo básico y un patrón de grafo opcional:

```
{ ?s :p1 ?v1 OPTIONAL {?s :p2 ?v2 } }

LeftJoin(
    Join(Z, BGP(?s :p1 ?v1)),
    Join(Z, BGP(?s :p2 ?v2)) ),
    true)

LeftJoin(BGP(?s :p1 ?v1), BGP(?s :p2 ?v2), true)
```

Ejemplo: grupo que consiste en un patrón de grafo básico y dos patrones de grafo opcionales:

```
{ ?s :p1 ?v1 OPTIONAL {?s :p2 ?v2 } OPTIONAL { ?s :p3 ?
v3 } }

LeftJoin(
    LeftJoin(
        BGP(?s :p1 ?v1),
        BGP(?s :p2 ?v2),
        true) ,
    BGP(?s :p3 ?v3),
    true)
```

Ejemplo: grupo formado por un patrón de grafo básico y un patrón de grafo opcional con un filtro:

```
{ ?s :p1 ?v1 OPTIONAL {?s :p2 ?v2 FILTER(?v1<3) } }

LeftJoin(
     Join(Z, BGP(?s :p1 ?v1)),
     Join(Z, BGP(?s :p2 ?v2)),
     (?v1<3) )

LeftJoin(
    BGP(?s :p1 ?v1) ,
    BGP(?s :p2 ?v2) ,
   (?v1<3) )
```

Ejemplo: grupo formado por la unión de un patrón de grafo y un patrón de grafo opcional:

```
{ {?s :p1 ?v1} UNION {?s :p2 ?v2} OPTIONAL {?s :p3 ?
v3} }

LeftJoin(
  Union(BGP(?s :p1 ?v1),
        BGP(?s :p2 ?v2)) ,
  BGP(?s :p3 ?v3) ,
  true )
```

Ejemplo: grupo formado por un patrón de grafo básico, un filtro y un patrón de grafo opcional:

```
{ ?s :p1 ?v1 FILTER (?v1 < 3 ) OPTIONAL {?s :p2 ?
v2} } }

Filter( ?v1 < 3 ,
  LeftJoin( BGP(?s :p1 ?v1), BGP(?s :p2 ?v2), true) ,
  )
```

12.2.3 Conversión de modificadores de solución

Paso 1 : ToList

ToList transforma un multiconjunto en una secuencia con los mismos elementos y cardinalidad. Esto no implica la ordenación de la secuencia; los duplicados no tienen por qué ser adyacentes.

Sea M := ToList(Patrón)

Paso 2 : ORDER BY

Si la cadena de consulta tiene una cláusula ORDER BY

M := OrderBy(M, lista de comparadores de orden)

Paso 3 : Proyección

M := Project(M, vars)

donde vars es el conjunto de variables mencionadas en la cláusula SELECT o todas las variables mencionadas en la consulta se se utiliza SELECT * .

Paso 4 : DISTINCT

Si la consulta contiene la cláusula DISTINCT,

M := Distinct(M)

Paso 5 : REDUCED

Si la consulta contiene la cláusula REDUCED,

M := Reduced(M)

Paso 6 : OFFSET y LIMIT

Si la consulta contiene "OFFSET start" o "LIMIT length"

M := Slice(M, start, length)

El valor por defecto de "start" es 0

El valor por defecto de "length" es (size(M)-start).

La consulta abstracta final es M.

12.3 Patrones de grafo básicos

Cuando concuerdan patrones de grafo, las posibles soluciones forman un *multiconjunto* [multiset], también conocido como *bag* (saco). Un multiconjunto es una colección no ordenada de elementos en la que cada elemento puede aparecer más de una vez. Está descrito por un conjunto de elementos y una fución de cardinalidad que indica el número de ocurrencias de cada elemento en el multiconjunto.

Se utiliza μ para representar las correspondencias de solución y

Se utiliza μ_0 para representar aquella correspondencia para la que dom(μ_0) es el conjunto vacío.

Se utiliza Ω_0 para representar el multiconjunto formado exactamente por la correspondencia vacía μ_0, con cardinalidad 1. Esta es la identidad de la unión.

Se utiliza μ(?x->t) para representar la variable de correspondencia de la solución con el término RDF t : { (x, t) }

Se utiliza Ω(?x->t) para representar el multiconjunto formado exactamente por μ(?x->t), es decir, { { (x, t) } } con cardinalidad 1.

> **Definicion: Correspondencias compatibles**
>
> Dos correspodencias de solución μ_1 y μ_2 son compatibles si, para cada variable v en dom(μ_1) y en dom(μ_2), $\mu_1(v) = \mu_2(v)$.

Si μ_1 y μ_2 son compatibles entonces la unión de conjuntos entre μ_1 *y* μ_2 es también una correspondencia. Se utiliza merge(μ_1, μ_2) para representar la unión de conjuntos entre μ_1*y* μ_2.

Se utiliza card[Ω](μ) para la cardinalidad de la correspondencia de solución μ en un multiconjunto de correspondencias Ω.

12.3.1 Concordancia de patrones de grafo básicos SPARQL

Los patrones de grafo básicos forman la base de la concordancia de patrones SPARQL. Un patrón de grafo básico concuerda con un grafo activo para esa parte de la consulta. Los patrones de grafo básicos pueden ser instanciados sustituyendo tanto variables como nodos en blanco por términos, ofreciendo dos nociones de instancia. Los nodos en blanco son sustituidos usando una instancia RDF de correspondencia, σ, transformandolos en términos RDF; las variables son sustituidas por una correspondencia de solución, transformándolas en términos RDF.

> **Definición: Correspondencia de instancia de patrón**
>
> Una **correspondencia de instancia de patrón**, P, es la combinación de una correspodencia de instancia RDF, σ, y una correspondencia de solución, μ. P(x) = μ(σ(x))

Cualquier correspondencia de instancia de patrón define una única correspondencia de solución y una única correspondencia de instancia RDF, obtenida mediante la restricción de las variables de consulta y de los nodos en blanco respectivamente.

> **Definición: Correspondencia de patrón de grafo básico**
>
> Sea BGP un patrón de grabo básico y sea G un grafo RDF.
>
> μ es una **solución** para BGP a partir de G cuando existe una correspondencia de instancia de patrón P, de tal manera que P(BGP) es un subgrafo de G y μ es la restricción de P para las variables de consulta en BGP.
>
> card[Ω](μ) = card[Ω](número de correspondencias de instancias RDF distintas, σ, de tal manera que P = μ(σ) es una correspondencia de instancia de patrón y P(BGP) es un subgrafo de G).

Si un patrón de grafo básico es el conjunto vacío, entonces la solución es Ω_0.

12.3.2 Tratamiento de nodos en blanco

Esta definición permite a la correspondencia de solución vincular una variable en un patrón de grafo básico, BGP, con un nodo en blanco en G. Debido a que SPARQL trata los identificadores de nodos en blanco tal y como se indica en el documento Formato XML de los resultados de consultas SPARQL, no pueden entenderse como nodos identificados en el grafo activo del conjunto de datos. Si DS es el conjunto de datos de una consulta, se entiende que las soluciones de patrón no provienen del grafo activo de DS en sí, sino de un grafo RDF denominado grafo de ámbito (*scoping graph*), que es un grafo equivalente al grafo activo de DS pero que no comparte ningún nodo en blanco con DS o con BGP. El mismo grafo de ámbito es utilizado para todas las soluciones de una misma consulta. El grafo de ámbito es puramente una construcción teórica; en la práctica, el efecto es obtenido simplemente por las convenciones de ámbito de documento para identificadores de nodos en blanco.

Puesto que los nodos en blanco RDF permiten soluciones redundantes de infinitud de muchos patrones, pueden existir infinitas soluciones de patrones (obtenidas mediante la sustitución de nodos en blanco por otros diferentes). Es necesario, por tanto, delimitar de alguna manera las soluciones para un patrón de grafo básico. SPARQL utiliza el criterio de concordancia de subgrafo para determinar las soluciones de un patrón de grafo básico. Existe una solución para cada correspondencia distinta de instancia de patrón de grafo básico con el subconjunto del grafo activo.

Esto está optimizado para facilitar el cálculo en vez de eliminar la redundancia. Permite que los resultados de un consulta tenga redundancia, incluso cuando el grafo activo del conjunto de datos es reducido, permitiendo que se obtenjan diferentes resultados de un consulta a partir de conjuntos de datos lógicamente equivalentes.

12.4 Álgebra SPARQL

Para cada símbolo de una consulta abstracta SPARQL, se define un operador de evaluación. Los operadores algebraicos de SPARQL del mismo nombre se utilizan para evaluar consultas abstractas SPARQL del modo en el que se describe en "Semántica de evaluación SPARQL".

Definición: Filter

Sea Ω un conjunto múltiple de soluciones de correspondencia y sea expr una expresión. Se define:

Filter(expr, Ω) = { μ | μ en Ω y expr(μ) como una expresión que tiene un valor efectivo booleano de verdadero }

card[Filter(expr, Ω)](μ) = card[Ω](μ)

Definición: Join

Sean Ω_1 y Ω_2 conjuntos múltiples de soluciones de correspondencia. Se define:

Join(Ω_1, Ω_2) = { merge(μ_1, μ_2) | μ_1 en Ω_1 y μ_2 en Ω_2, y μ_1 y μ_2 son compatibles}

card[Join(Ω_1, Ω_2)](μ) =
  para cada merge(μ_1, μ_2), μ_1 en Ω_1 y μ_2 en Ω_2 tal que μ = merge(μ_1, μ_2),
    sea la suma de (μ_1, μ_2), card[Ω_1](μ_1)*card[Ω_2](μ_2)

Es posible que una solución de correspondencia μ en un Join pueda plantearse en diferentes soluciones de correspondencia, μ_1 y μ_2 en los conjuntos múltiples objeto de la unión. La cardinalidad de μ es la suma de las cardinalidades de todas las posibilidades.

Definición: Diff

Sean Ω_1 y Ω_2 conjuntos múltiples de soluciones de correspondencia. Se define:

Diff(Ω_1, Ω_2, expr) = { μ | μ en Ω_1 tal que para todo μ' en Ω_2, o bien μ y μ' no son compatibles o μ y μ' son compatibles y expr(merge(μ, μ')) tiene un valor efectivo booleano de falso }

card[Diff(Ω_1, Ω_2, expr)](μ) = card[Ω_1](μ)

Diff se emplea internamente para la definición de LeftJoin.

Definición: LeftJoin

Sean Ω_1 y Ω_2 multiconjuntos de soluciones de correspondencia y sea expr una expresión. Se define:

LeftJoin(Ω_1, Ω_2, expr) = Filter(expr, Join(Ω_1, Ω_2)) *unión-conjunto* Diff (Ω_1, Ω_2, expr)

card[LeftJoin(Ω_1, Ω_2, expr)](μ) = card[Filter(expr, Join(Ω_1, Ω_2))](μ) + card[Diff(Ω_1, Ω_2, expr)](μ)

Es decir, escrito en su forma completa:

LeftJoin(Ω_1, Ω_2, expr) =
  { merge(μ_1, μ_2) | μ_1 en Ω_1 y μ_2 en Ω_2, y μ_1 y μ_2 son compatibles y expr(merge (μ_1, μ_2)) es verdadero }
unión-conjunto
  { μ_1 | μ_1 en Ω_1 y μ_2 en Ω_2, y μ_1 y μ_2 no son compatibles }

unión-conjunto
{ μ_1 | μ_1 en Ω_1 y μ_2 en Ω_2, y μ_1 y μ_2 son compatibles y expr(merge(μ_1, μ_2)) es falso }

Puesto que son distintos, la cardinalidad de LeftJoin es la cardinalidad de los componentes individuales de la definición.

Definición: Union

Sean Ω_1 y Ω_2 multiconjuntos de soluciones de correspondencia. Se define:

Union(Ω_1, Ω_2) = { μ | μ en Ω_1 o μ en Ω_2 }

card[Union(Ω1, Ω2)](μ) = card[Ω1](μ) + card[Ω2](μ)

Se escribe [x | C] para denotar una secuencia de elementos donde C(x) es verdadero.

Se escribe card[L](x) para denotar la cardinalidad de x en L.

Definición: ToList

Sea Ω un multiconjunto de soluciones de correspondencia. Se define:

ToList(Ω) = secuencia de correspondencias μ en Ω en cualquier orden, con card[Ω](μ) que indica las ocurrencias de μ

card[ToList(Ω)](μ) = card[Ω](μ)

Definición: OrderBy

Sea Ψ una secuencia de soluciones de correspondencia. Se define:

OrderBy(Ψ, condición) = [μ | μ en Ψ y la secuencia satisface la condición requerida]

card[OrderBy(Ψ, condición)](μ) = card[Ψ](μ)

Definición: Project

Sea Ψ una secuencia de soluciones de correspondenciay PV un conjunto de variables.

Para la correspondencia μ, se escribe Proj(μ, PV) para denotar la restricción de μ a las variables en PV.

Project(Ψ, PV) = [Proj(Ψ[μ], PV) | μ en Ψ]

card[Project(Ψ, PV)](μ) = card[Ψ](μ)

El orden Project(Ψ, PV) debe preservar cualquier oden indicado por OrderBy.

Definición: Distinct

Sea Ψ una secuencia de soluciones de correspondencia. Se define:

Distinct(Ψ) = [μ | μ en Ψ]

card[Distinct(Ψ)](μ) = 1

El orden de Distinct(Ψ) debe preservar cualquier oden indicado por OrderBy.

Definición: Reduced

Sea Ψ una secuencia de soluciones de correspondencia. Se define:

Reduced(Ψ) = [μ | μ en Ψ]

card[Reduced(Ψ)](μ) esté comprendido entre 1 y card[Ψ](μ)

El orden de Reduced(Ψ) debe preservar cualquier orden indicado por OrderBy.

Un solución a la que se aplica el modificador de secuencia Reduced no garantiza una cardinalidad definida.

Definition: Slice

Sea Ψ una secuencia de soluciones de correspondencia. Se define:

Slice(Ψ, start, length)[i] = Ψ[start+i] for i = 0 to (length-1)

12.5 Semántica de evaluación SPARQL

Se define eval(D(G), patrón de grafo) como la evaluación de un patrón de grafo con respecto a un conjunto de datos D que incluye el grafo activo G. El grafo activo es inicialmente el grafo por defecto.

```
D : un conjunto de datos
D(G) : D un conjunto de datos con un grafo activo G (aquel cuyos pa
D[i] : El grafo con IRI i en el conjunto de datos D
D[DFT] : el grafo por defecto de D
P, P1, P2 : patrones de grafo
L : una secuencia de solución
```

Definición: Evaluación de Filter(F, P)

```
eval(D(G), Filter(F, P)) = Filter(F, eval(D(G),P))
```

Definición: Evaluación de Join(P1, P2)

```
eval(D(G), Join(P1, P2)) = Join(eval(D(G), P1), eval(D(G
```

Definición: Evaluación de LeftJoin(P1, P2, F)

```
eval(D(G), LeftJoin(P1, P2, F)) = LeftJoin(eval(D(G), P1
```

Definición: Evaluación de un Patrón de Grafo Básico

```
eval(D(G), BGP) = multiconjunto de soluciones de corresp
```

Véase sección 12.3 Patrones de Grafo Básico

Definición: Evaluación de una Unión de Patrones

```
eval(D(G), Union(P1,P2)) = Union(eval(D(G), P1), eval(D(
```

Definición: Evaluación de un Patrón de Grafo

```
Si IRI es un nombre de grafo en D
eval(D(G), Graph(IRI,P)) = eval(D(D[IRI]), P)

si IRI no es un nombre de grafo en D
eval(D(G), Graph(IRI,P)) = el multiconjunto vacío

eval(D(G), Graph(var,P)) =
     Sea R el multiconjunto vacío
     Para cada IRI i en D
        R := Union(R, Join( eval(D(D[i]), P) , Ω(?var->i
     el resultado es R
```

La evaluación del grafo utiliza el operador de unión del álgebra de SPARQL. La cardinalidad de una solución de correspondencia es la suma de las cardinalidades de las soluciones de correspondencia de cada operación join.

Definición: Evaluación de ToList

```
eval(D, ToList(P)) = ToList(eval(D(D[DFT]), P))
```

Definición: Evaluación de Distinct

```
eval(D, Distict(L)) = Distinct(eval(D, L))
```

Definición: Evaluación de Reduced

```
eval(D, Reduced(L)) = Reduced(eval(D, L))
```

Definición: Evaluación de Project

```
eval(D, Project(L, vars)) = Project(eval(D, L), vars)
```

Definición: Evaluación de OrderBy

```
eval(D, OrderBy(L, condition)) = OrderBy(eval(D, L), con
```

Definición: Evaluación de Slice

```
eval(D, Slice(L, start, length)) = Slice(eval(D, L), sta
```

12.6 Ampliación de concordancias de grafos básicos SPARQL

El diseño general de SPARQL se puede utilizar para las consultas que conllevan una forma de inferencia más elaborada que una inferencia simple, reescribiendo las condiciones de comparaciones de los patrones de grafo básico. Puesto que se trata de un problema de investigación abierto al establecimiento de dicha condiciones en una única forma general aplicables a todas las formas de inferencia y elimina de manera óptima las redundancias innecesarias o inapropiadas, este documento sólo ofrece las condiciones necesarias que debe satisfacer cualquier solución. Éstas tendrán que ampliarse a las definiciones completas para cada caso en particular..

Los patrones de grafo básico tienen la misma relación con los patrones de tripletas que a su vez tienen los grafos RDF con las tripletas RDF, compartiendo gran parte de la terminología. En particular dos patrones de grafo básico se dicen que son equivalentes si existe una biyección M entre los términos de un patrón de tripletas que define las correspondencias entre nodos en blanco o entre variables, literales e IRIs, tal y como una tripleta (s, p, o) está en el primer patrón, si y solo si la tripleta (M(s), M(p), M(o)) está en la segunda. Esta definición amplía esto para inferencias de grafos RDF con patrones de grafo básico, preservando los nombres de variable entre patrones equivalentes.

Un *régimen de inferencia* especifica

1. un subconjunto de grafos RDF denominado *bien-formado* para el régimen
2. una relación de *inferencia* entre subconjuntos de grafos bien formados y grafos bien formados.

Ejemplos de regímenes de inferencia incluyen la inferencia simple [RDF-MT], la inferencia RDF [RDF-MT], la inferencia RDFS [RDF-MT], D-inferencia [RDF-MT] y la inferencia OWL-DL [OWL-Semantics]. De éstos, solo la inferencia OWL-DL restringe el conjunto de grafos bien-formados. Si E es un régimen de inferencia entonces será posible referirse a E-implicación, E-consistencia, etc, siguiendo esta convención para los nombres.

Algunos regímenes de inferencia pueden categorizar algunos grafos RDF como inconsistentes. Por ejemplo, el siguiente grafo RDF:

```
_:x rdf:type xsd:string .
_:x rdf:type xsd:decimal .
```

es D-inconsistente cuando D contiene los tipos de datos XSD. El efecto de una consulta sobre un grafo inconsistente no está cubierto por esta especificación, pero debe ser especificada por una amplicación SPARQL particular.

Una ampliación de SPARQL para E-inferencia debe satisfacer las siguientes condiciones.

1 -- El grafo de ámbito, SG, correspondiente con cualquier grafo activo consistente AG es especificado de manera única y es E-equivalente a AG.

2 -- Para cualquier patron de grafo básico BGP y solución de correspondencia de patrón P, P(BGP) está bien-formado para E

3 -- Para cualquier grafo de ámbito SG y conjunto de respuestas $\{P_1 \ldots P_n\}$, para un patrón de grafo básico BGP, donde $\{BGP_1 \ldots BGP_n\}$ es un conjunto de patrones de grafo básico, todos equivalentes a BGP, ninguno de los cuales comparten ningún nodo en blanco entre sí o con SG, se afirma que:

$$SG \text{ E-inferencia } (SG \text{ union } P_1(BGP_1) \text{ union } \ldots \text{ union } P_n(BGP_n))$$

Estas condiciones no determinan totalmente el conjunto de posibles respuestas, puesto que RDF permite redundancias de forma ilimitada. Además, por tanto, debe cumplirse lo siguiente.

4 -- Cada ampliación SPARQL debe proporcionar condiciones sobre los conjuntos de respuestas que garanticen que cada BGP y AG tengan un conjunto finito de respuestas, el cual es único hasta la equivalencia del grafo RDF.

Notas

(a) SG será a menudo el gráfico equivalente a AG, pero restringiendo esto a E-equivalencia permite algunas formas de normalización, por ejemplo la eliminación de redundancias semánticas, que se aplicará a los documentos fuente originales antes de la consulta.

(b) La construcción de la condición 3 asegura que cualquier nodo en blanco introducido por la correspondencia de solución sea usado de forma que sea internamente consistente con el modo en el que aparecen los nodos en blanco en SG. Esto asegura que los identificadores de nodos en blanco aparezcan en

una respuesta de un conjunto de respuestas solo cuando los nodos en blanco asi identificados sea efectivamente idénticos en SG. Si la ampliación no permite vincular respuestas con nodos en blanco, entonces esta condición puede simplicarse con esta otra:

SG E-inferencia P(BGP) para cada patrón de solución P.

(c) Estas condiciones no imponen el requisito SPARQL de que SG no comparta nodos en blanco con AG o BGP. En particular, permite que SG sea realmente AG. Esto permite protocolos de consulta en los que los identificadores de nodos en blanco mantienen su significado entre la consulta y el documento fuente original, o entre múltiples consultas. Sin embargo, dichos protocolos no son soportados por la especificación actual del protocolo SPARQL.

(d) Puesto que las condiciones 1 a 3 son únicamente necesarias en las respuestas, la condición 4 permite casos donde el conjunto de respuestas correctas pueda restringirse de varias formas. Por ejemplo, el estado actual de las consultas OWL-DL se centran en los casos donde los vínculos entre respuestas y nodos en blanco están prohibidos. Nótese que estas condiciones permiten incluso casos de 'silencio' patológicos donde cada consulta tiene un conjunto de respuestas vacías.

(e) Ninguna de estas condiciones se refieren explícitamente a instancias de correspondencias sobre nodos en blanco en BGP. Para algunos regímenes de inferencia, la interpretación de la existencia de nodos en blanco no puede ser totalmente capturada a través de la existencia de una única instancia de correspondencia. Estas condiciones permiten que dichos regímenes den a los nodos en blanco en los patrones de consulta una 'existencia plena' de lectura.

Es fácil demostrar que SPARQL satisface estas condiciones para el caso en el que E es una inferencia simple, dado que la condición SPARQL en SG indica que se trata de un grafo equivalente a AG con la salvedad de que no comparte nodos en blanco con AG o BGP (que satisfagan la primera condición). La única condición no trivial es (3).

Cada respuesta P_i es la restricción de correspondencia de solución de una instancia SPARQL M_i tal que $M_i(BGP_i)$ es un subgrafo de SG. Puesto que BGP_i y SG no tienen nodos en blanco en común, el rango de M_i no contienen nodos en blanco de BGP_i; por tanto, la correspondencia de solución P_i y la instancia RDF objeto de la correspondencia I_i que componen M_i son conmutativas, así pues $M_i(BGP_i) = I_i(P_i(BGP_i))$. Por lo tanto

$M_1(BGP_1)$ union ... union $M_n(BGP_n)$
= $I_1(P_1(BGP_1))$ union ... union $I_n(P_n(BGP_n))$
= $[I_1 + ... + I_n](P_1(BGP_1)$ union ... union $P_n(BGP_n))$

puesto que los dominios de todas las instancias I_i objeto de la correspondencia son mútuamente excluyentes. Teniendo en cuenta que también son excluyentes con respecto a SG,

SG union $[I_1 + ... + I_n](P_1(BGP_1)$ union ... union $P_n(BGP_n))$
= $[I_1 + ... + I_n](SG$ union $P_1(BGP_1)$ union ... union $P_n(BGP_n))$

por ejemplo

SG union $P_1(BGP_1)$ union ... union $P_n(BGP_n)$

tiene una instancia que es un subgrafo de SG, en consencuencia es una inferencia simple a partir de SG debido al Lema de interopación RDF [RDF-MT].

A. Gramática SPARQL

A.1 Cadena de consulta SPARQL

Una cadena de consulta SPARQL es una cadena de caracteres Unicode (ver sección 6.1 Conceptos de cadena del documento [CHARMOD]) en el lenguaje definido por la siguiente gramática que comienza con la producción de una Consulta. Para asegurar la compatibilidad con futuras versiones de Unicode, los caracteres en esta cadena puede incluir puntos de código Unicode que están sin asignar en la fecha de publicar la presente recomendación (ver Sintaxis de Patrones e Identificadores en la sección 4 Sintaxis de Patrones del documento [UNIID]). Para producciones con clases de caracteres excluidos (por ejemplo `[^<>'{}|^`]`), los caracteres excluidos están en el rango `#x0 - #x10FFFF`.

A.2 Puntos de código para secuencias de escape

Una cadena de consulta SPARQL es procesada para buscar puntos de código de secuencias de escape antes de su análisis (parsing) a partir de la gramática definida a continuación en notación EBNF. Los puntos de código de las secuencias de escapepara una cadena de consulta SPARQL son:

Escape	Unicode code point
'\u' HEX HEXHEX HEX	Un punto de código Unicode en el rango que va de U+0 a U+FFFF inclusive, correspondiente al valor hexadecimal codificado.
'\U' HEX HEXHEX HEX HEXHEX HEX HEX	Un punto de código Unicode en el rango que va de U+0 a U+10FFFF inclusive, correspondiente al valor hexadecimal codificado.

donde HEX es un carácter hexadecimal

```
HEX ::= [0-9] | [A-F] | [a-f]
```

Ejemplos:

```
<ab\u00E9xy>        # Codepoint 00E9 is Latin small e with acute - é
\u03B1:a            # Codepoint x03B1 is Greek small alpha - α
a\u003Ab            # a:b -- codepoint x3A is colon
```

Los puntos de código de secuencias de escape pueden aparecer en cualquier lugar de la cadena de consulta. Pueden ser procesados antes de su análisis

(parsing) basándose en las reglas de la gramática y por tanto sustituidos por puntos de código con significado en dicha gramática, tales como ":" para marcar un nombre de prefijo.

Estas secuencias de escape no están incluidas en la siguiente gramática. Únicamente se ofrecen las secuencias de escape para caracteres que podrían ser legales en ese punto de la gramática. Por ejemplo, la variable "`?x\u0020y`" no es legal (`\u0020` es un espacio y no se permite en el nombre de una variable).

A.3 Espacio en blanco

El Espacio en blanco (en producción `WS`) se utiliza para separar dos terminales que de otra forma podrían ser reconocidos erroneamente como un único terminal. Las reglas de nombre especificadas más adelante en mayúsculas indican cuando un espacio en blanco es significativo; éstos constituyen una posible opción para los terminales en la construcción de un parser SPARQL. Los espacios en blanco son significativos en las cadenas.

Por ejemplo:

```
?a<?b&&?c>?d
```

es la secuencia de tokens para la variable '`?a`', una IRI '`<?b&&?c>`', y una variable '`?d`'; no se trata de una expresión que incluya al operador '`&&`' que conecte dos expresiones usando '`<`' (menor que) y '`>`' (mayor que).

A.4 Comments

Los comentarios en las consultas SPARQL toman la forma de '`#`', fuera de una referencia IRI o una cadena, y continuan hasta el fin de la línea (marcado por los caracteres `0x0D` ó `0x0A`) o hasta el final del fichero si no hay un fin de línea después de la marca de comentario. Los comentarios se tratan como si fueran espacios en blanco.

A.5 Referencias IRI

El texto coincidente con la producción `IRI_REF` y la producción *`PrefixedName`* (tras la expansión del prefijo), después del procesamiento de los escapes, deben ser conformes con la sintaxis genérica de referencias IRI descrita en la sección 2.2 del documento RFC 3987 "ABNF para referencias IRI e IRIs" [RFC3987]. Por ejemplo, la `IRI_REF` `<abc#def>` puede darse en una cadena de consulta SPARQL, pero la `IRI_REF` `<abc##def>` no debe darse.

Las IRIs base declaradas con la palabra clave BASE deben ser IRIs absolutas. Un prefijo declarado con la palabra clave PREFIX no puede ser re-declarado en la misma consulta. Véase la sección 4.1.1, Sintaxis para IRI, para una descripción de BASE y PREFIX.

A.6 Etiquetas de Nodos en Blanco

No puede utilizarse en una misma consulta sencilla la misma etiqueta de nodos en blanco en dos patrones de gráfico básicos diferentes.

A.7 Secuencias de escape en cadenas

Además de los puntos de código de secuencias de escape pueden aplicarse los siguientes secuencias de escape en cualquier `cadena` de production (por ejemplo, `STRING LITERAL1`, `STRING LITERAL2`, `STRING LITERAL LONG1`, `STRING LITERAL LONG2`):

Escape	Unicode code point
'\t'	U+0009 (tabulación)
'\n'	U+000A (salto de línea)
'\r'	U+000D (retorno de carro)
'\b'	U+0008 (retroceso)
'\f'	U+000C (salto de formulario)
'\"'	U+0022 (comillas, comillas dobles)
"\'"	U+0027 (apostrofe-comillas, comillas simples)
'\\'	U+005C (barra diagonal inversa)

Ejemplos:

```
"abc\n"
"xy\rz"
'xy\tz'
```

A.8 Gramática

La notación EBNF usada en la gramática está definida mediante lo indicado en la sección 6 (Notación) del documento de la recomendación de XML 1.1 [XML11].

La comprobación de las palabras clave se realiza sin diferenciar mayúsculas y minúsculas con las excepción de la palabra clave '`a`' la cual, en consonancia con Turtle y N3, se utiliza en lugar del IRI de `rdf:type` (`http://www.w3.org/1999/02/22-rdf-syntax-ns#type` en su forma completa).

Palabras clave:

BASE	SELECT	ORDER BY	FROM	GRAPH	STR	isURI
PREFIX	CONSTRUCT	LIMIT	FROM NAMED	OPTIONAL	LANG	isIRI
	DESCRIBE	OFFSET	WHERE	UNION	LANGMATCHES	isLITERAL
	ASK	DISTINCT		FILTER	DATATYPE	REGEX
		REDUCED		a	BOUND	true
					sameTERM	false

Las secuencias de escape son sensibles a las mayúsculas.

Al escoger una regla para su comparación, se elige la coincidencia más larga.

[1]	*Query*	::=	Prologue (SelectQuery | ConstructQuery | DescribeQuery | AskQuery)
[2]	*Prologue*	::=	BaseDecl? PrefixDecl*
[3]	*BaseDecl*	::=	'BASE' IRI_REF
[4]	*PrefixDecl*	::=	'PREFIX' PNAME_NS IRI_REF
[5]	*SelectQuery*	::=	'SELECT' ('DISTINCT' | 'REDUCED')? (Var+ | '*') DatasetClause* WhereClauseSolutionModifier
[6]	*ConstructQuery*	::=	'CONSTRUCT' ConstructTemplate DatasetClause* WhereClause SolutionModifier
[7]	*DescribeQuery*	::=	'DESCRIBE' (VarOrIRIref+ | '*') DatasetClause* WhereClause? SolutionModifier
[8]	*AskQuery*	::=	'ASK' DatasetClause* WhereClause
[9]	*DatasetClause*	::=	'FROM' (DefaultGraphClause | NamedGraphClause)
[10]	*DefaultGraphClause*	::=	SourceSelector
[11]	*NamedGraphClause*	::=	'NAMED' SourceSelector
[12]	*SourceSelector*	::=	IRIref
[13]	*WhereClause*	::=	'WHERE'? GroupGraphPattern
[14]	*SolutionModifier*	::=	OrderClause? LimitOffsetClauses?
[15]	*LimitOffsetClauses*	::=	(LimitClauseOffsetClause? | OffsetClauseLimitClause?)
[16]	*OrderClause*	::=	'ORDER' 'BY' OrderCondition+
[17]	*OrderCondition*	::=	(('ASC' | 'DESC') BrackettedExpression) | (Constraint | Var)
[18]	*LimitClause*	::=	'LIMIT' INTEGER
[19]	*OffsetClause*	::=	'OFFSET' INTEGER
[20]	*GroupGraphPattern*	::=	'{' TriplesBlock? ((GraphPatternNotTriples | Filter) '.'? TriplesBlock?)* '}'
[21]	*TriplesBlock*	::=	TriplesSameSubject ('.' TriplesBlock?)?
[22]	*GraphPatternNotTriples*	::=	OptionalGraphPattern | GroupOrUnionGraphPattern | GraphGraphPattern
[23]	*OptionalGraphPattern*	::=	'OPTIONAL' GroupGraphPattern
[24]	*GraphGraphPattern*	::=	'GRAPH' VarOrIRIrefGroupGraphPattern
		::=	

[25]	*GroupOrUnionGraphPattern*		GroupGraphPattern ('UNION' GroupGraphPattern)*
[26]	*Filter*	::=	'FILTER' Constraint
[27]	*Constraint*	::=	BrackettedExpression | BuiltInCall | FunctionCall
[28]	*FunctionCall*	::=	IRIref ArgList
[29]	*ArgList*	::=	(NIL | '('Expression (','Expression)* ')')
[30]	*ConstructTemplate*	::=	'{' ConstructTriples? '}'
[31]	*ConstructTriples*	::=	TriplesSameSubject ('.' ConstructTriples?)?
[32]	*TriplesSameSubject*	::=	VarOrTerm PropertyListNotEmpty | TriplesNodePropertyList
[33]	*PropertyListNotEmpty*	::=	Verb ObjectList (';' (Verb ObjectList)?)*
[34]	*PropertyList*	::=	PropertyListNotEmpty?
[35]	*ObjectList*	::=	Object (','Object)*
[36]	*Object*	::=	GraphNode
[37]	*Verb*	::=	VarOrIRIref | 'a'
[38]	*TriplesNode*	::=	Collection | BlankNodePropertyList
[39]	*BlankNodePropertyList*	::=	'[' PropertyListNotEmpty']'
[40]	*Collection*	::=	'(' GraphNode+ ')'
[41]	*GraphNode*	::=	VarOrTerm | TriplesNode
[42]	*VarOrTerm*	::=	Var | GraphTerm
[43]	*VarOrIRIref*	::=	Var | IRIref
[44]	*Var*	::=	VAR1 | VAR2
[45]	*GraphTerm*	::=	IRIref | RDFLiteral | NumericLiteral | BooleanLiteral | BlankNode | NIL
[46]	*Expression*	::=	ConditionalOrExpression
[47]	*ConditionalOrExpression*	::=	ConditionalAndExpression ('||' ConditionalAndExpression)*
[48]	*ConditionalAndExpression*	::=	ValueLogical ('&&' ValueLogical)*
[49]	*ValueLogical*	::=	RelationalExpression
[50]	*RelationalExpression*	::=	NumericExpression ('=' NumericExpression | '!=' NumericExpression | '<' NumericExpression | '>' NumericExpression | '<=' NumericExpression | '>=' NumericExpression)?
[51]	*NumericExpression*	::=	AdditiveExpression
[52]	*AdditiveExpression*	::=	MultiplicativeExpression ('+' MultiplicativeExpression | '-' MultiplicativeExpression |

			NumericLiteralPositive | NumericLiteralNegative)*
[53]	*MultiplicativeExpression*	::=	UnaryExpression ('*' UnaryExpression | '/' UnaryExpression)*
[54]	*UnaryExpression*	::=	'!' PrimaryExpression | '+' PrimaryExpression | '-' PrimaryExpression | PrimaryExpression
[55]	*PrimaryExpression*	::=	BrackettedExpression | BuiltInCall | IRIrefOrFunction | RDFLiteral | NumericLiteral | BooleanLiteral | Var
[56]	*BrackettedExpression*	::=	'(' Expression')'
[57]	*BuiltInCall*	::=	'STR' '(' Expression')' | 'LANG' '('Expression ')' | 'LANGMATCHES' '('Expression ','Expression ')' | 'DATATYPE' '('Expression ')' | 'BOUND' '('Var ')' | 'sameTerm' '('Expression ','Expression ')' | 'isIRI' '('Expression ')' | 'isURI' '('Expression ')' | 'isBLANK' '('Expression ')' | 'isLITERAL' '('Expression ')' | RegexExpression
[58]	*RegexExpression*	::=	'REGEX' '(' Expression',' Expression (',' Expression)? ')'
[59]	*IRIrefOrFunction*	::=	IRIref ArgList?
[60]	*RDFLiteral*	::=	String (LANGTAG | ('^^' IRIref))?
[61]	*NumericLiteral*	::=	NumericLiteralUnsigned | NumericLiteralPositive | NumericLiteralNegative
[62]	*NumericLiteralUnsigned*	::=	INTEGER | DECIMAL | DOUBLE
[63]	*NumericLiteralPositive*	::=	INTEGER_POSITIVE | DECIMAL_POSITIVE | DOUBLE_POSITIVE
[64]	*NumericLiteralNegative*	::=	INTEGER_NEGATIVE | DECIMAL_NEGATIVE | DOUBLE_NEGATIVE
[65]	*BooleanLiteral*	::=	'true' | 'false'
[66]	*String*	::=	STRING_LITERAL1 | STRING_LITERAL2 | STRING_LITERAL_LONG1 | STRING_LITERAL_LONG2
[67]	*IRIref*	::=	IRI_REF | PrefixedName
[68]	*PrefixedName*	::=	PNAME_LN | PNAME_NS
[69]	*BlankNode*	::=	BLANK_NODE_LABEL | ANON

Producciones para terminales:

[70]	*IRI_REF*	::=	'<' ([^<>"{}|^`\]-[#x00-#x20])* '>'
[71]	*PNAME_NS*	::=	PN_PREFIX? ':'
[72]	*PNAME_LN*	::=	PNAME_NS PN_LOCAL
[73]	*BLANK_NODE_LABEL*	::=	'_:' PN_LOCAL
[74]	*VAR1*	::=	'?' VARNAME
[75]	*VAR2*	::=	'$' VARNAME
[76]	*LANGTAG*	::=	'@' [a-zA-Z]+ ('-' [a-zA-Z0-9]+)*
[77]	*INTEGER*	::=	[0-9]+
[78]	*DECIMAL*	::=	[0-9]+ '.' [0-9]* | '.' [0-9]+
[79]	*DOUBLE*	::=	[0-9]+ '.' [0-9]* EXPONENT | '.' ([0-9])+ EXPONENT | ([0-9])+ EXPONENT
[80]	*INTEGER_POSITIVE*	::=	'+' INTEGER
[81]	*DECIMAL_POSITIVE*	::=	'+' DECIMAL
[82]	*DOUBLE_POSITIVE*	::=	'+' DOUBLE
[83]	*INTEGER_NEGATIVE*	::=	'-' INTEGER
[84]	*DECIMAL_NEGATIVE*	::=	'-' DECIMAL
[85]	*DOUBLE_NEGATIVE*	::=	'-' DOUBLE
[86]	*EXPONENT*	::=	[eE] [+-]? [0-9]+
[87]	*STRING_LITERAL1*	::=	"'" (([^#x27#x5C#xA#xD]) | ECHAR)* "'"
[88]	*STRING_LITERAL2*	::=	'"' (([^#x22#x5C#xA#xD]) | ECHAR)* '"'
[89]	*STRING_LITERAL_LONG1*	::=	"'''" (("'" | "''")? ([^'\] | ECHAR))* "'''"
[90]	*STRING_LITERAL_LONG2*	::=	'"""' (('"' | '""')? ([^"\] | ECHAR))* '"""'
[91]	*ECHAR*	::=	'\' [tbnrf\"']
[92]	*NIL*	::=	'(' WS* ')'
[93]	*WS*	::=	#x20 | #x9 | #xD | #xA
[94]	*ANON*	::=	'[' WS* ']'
[95]	*PN_CHARS_BASE*	::=	[A-Z] | [a-z] | [#x00C0-#x00D6] | [#x00D8-#x00F6] | [#x00F8-#x02FF] | [#x0370-#x037D] | [#x037F-#x1FFF] | [#x200C-#x200D] | [#x2070-#x218F] | [#x2C00-#x2FEF] | [#x3001-#xD7FF] | [#xF900-#xFDCF] | [#xFDF0-#xFFFD] | [#x10000-#xEFFFF]
[96]	*PN_CHARS_U*	::=	PN_CHARS_BASE | '_'
[97]	*VARNAME*	::=	(PN_CHARS_U | [0-9]) (PN_CHARS_U | [0-9] | #x00B7 | [#x0300-#x036F] | [#x203F-#x2040])*
[98]	*PN_CHARS*	::=	PN_CHARS_U | '-' | [0-9] | #x00B7 | [#x0300-#x036F] | [#x203F-#x2040]

[99]	*PN_PREFIX*	::=	PN_CHARS_BASE ((PN_CHARS|'.')* PN_CHARS)?
[100]	*PN_LOCAL*	::=	(PN_CHARS_U | [0-9]) ((PN_CHARS|'.')* PN_CHARS)? Note that SPARQL local names allow leading digits while XML local names do not.

Notas:

1. La gramática de SPARQL es LL(1) cuand las reglas con nombres en mayúsculas se utilizan como terminales.
2. En números con signo, no se permiten espacios en blanco entre el signo y el número. La regla gramatical *AdditiveExpression* permite esto al cubrir ambos casos en una expresión seguida de un número con signo. Estos producen una adición o sustracción del número sin signo según proceda.

Algunos ficheros con la gramática de algunas herramientas utilizadas comúnmente están disponibles aquí.

B. Conformidad

Véanse el apéndice A Gramática SPARQL al respecto de la conformidad de las cadenas de consulta SPARQL, and la sección 10 Formas de consulta para la conformidad de los resultados de la consulta. Véase el apéndice D. Tipos de medios en Internet para la conformidad con el tipo de medio application/sparql-query.

Esta especificación está desarrollada para su uso conjunto con el Protocolo SPARQL [SPROT] y el Formato XML de resultados de consultas SPARQL [RESULTS]. Véanse dichas especificaciones para ampliar información acerca de sus criterios de conformidad.

Nótese que el protocolo SPARQL describe tanto una interfaz abstracta como un protocolo de red, y la interfaz abstracta muy aplicarse tanto a APIs com a interfaces de red.

C. Consideraciones de seguridad (informativo)

Las consultas SPARQL que usen FROM, FROM NAMED o GRAPH pueden causar que las URIs especificadas sean dereferenciadas. Esto puede ocasionar un uso adicional de los recursos de red, disco o CPU junto con efectos secundarios asociados como la denegación de servicio. Deben tenerse en cuenta los aspectos de seguridad tratados en la sección 7 del documento Identificador Unificado de Recurso (URI): Sintaxis Genérica [RFC3986]. Además, en algunos casos los contenidos de las URIs `file:` pueden ser accedidos, procesados y devueltos como resultados, proporcionando no deseado a recursos locales.

El lenguaje SPARQL permite amplicaciones, las cuales pueden tener sus propias implicaciones de seguridad.

Múltiples IRIs pueden tener la misma apariencia. Los caracteres de diferentes escrituras pueden tener un aspecto similar (una "o" cirílica puede parecer similar a la letra latina "o"). Un carácter seguido de una combinación de caracteres pueden tener la misma representación visual que otro carácter (LA LETRA LATINA E MINÚSCULA, seguida de la COMBINACIÓN DE ACENTO AGUDO puede tener la misma representación visual que LA LETRA LATINA E MINÚSCULA CON ACENTO). Los usuarios de SPARQL deben tener en cuenta que las IRIs de las consultas deben coincidir con las IRIs de los datos. Puede encontrarse más información sobre la coincidencia de caracteres similares en Consideraciones de Seguridad de Unicode [UNISEC] y en la sección 8 de Identificadores Internacionalizados de Recursos (IRIs) [RFC3987].

D. Tipos de medios de Internet, extensiones de fichero y tipos de ficheros Macintosh

contact:
Eric Prud'hommeaux
Véase también:
Como registrar un tipo de medio para una especificación del W3C
Registro de tipos de medios de Internet, consistencia de uso
TAG hallado el 3 de Junio de 2002 (Revisado el 4 de septiembre de 2002)

El tipo de medio de Internet / Tipo MIME para el lenguaje de consulta SPARQL es "application/sparql-query".

Se recomienda que los ficheros de consultas SPARQL tengan la extensión ".rq" (en minúsculas) en todas las plataformas.

Se recomienda que los ficheros de consultas SPARQL almacenados en sistemas de fichero HFS de Macintosh se proporcione como un tipo de fichero "TEXT".

Nombre de tipo:
application
Nombre de subtipo:
sparql-query
Parámetros requeridos:
Ninguno
Parámetros opcionales:
Niguno
Consideraciones de codificación:
La sintaxis del lenguaje de consulta SPARQL está expresado mediante puntos de código Unicode [UNICODE]. La codificación es siempre UTF-8 [RFC3629].
Los puntos de código Unicode también pueden expresarse utilizando la sintaxis \uXXXX (U+0 a U+FFFF) o \UXXXXXXXX (de U+10000 en adelante) donde X es un dígito hexadecimal [0-9A-F]
Consideraciones de seguridad:
Véase el apéndice C, Consideraciones de seguridad así como la sección 7 sobre consideraciones de seguridad de RFC 3629 [RFC3629].

Consideraciones de interoperabilidad:
No hay cuestiones de interoperabilidad conocidas.
Especificación publicada:
Esta especificación.
Aplicaciones que usan este tipo de medio:
No hay aplicaciones conocidas que utilizan actualmente este tipo de medio.
Información adicional:
Número(s) mágico(s):
Una consulta SPARQL puede tener la cadena 'PREFIX' (con independencia del caso) cerca del inicio del documento.
Extensión o extensiones de fichero:
".rq"
URI Base:
La expresión SPARQL 'BASE <IRIref>' puede cambiar la base URI actual para referencias IRI relativas del lenguaje la consulta para que sean utilizadas secuencialmente más tarde en el documento.
Código(s) de tipo de fichero Macintosh:
"TEXT"
Persona y correo electrónico de contacto para información adicional:
public-rdf-dawg-comments@w3.org
Intención de uso:
Común
Restricciones de uso:
Ninguna
Control de autoría y cambios:
La especificación SPARQL es un trabajo producto del Grupo de trabajo de acceso a datos RDF del World Wide Web Consortium. El W3C tiene el control de los cambios sobre estas especificaciones.

E. Referencias

Referencias normativas

[CHARMOD]
Character Model for the World Wide Web 1.0: Fundamentals, R. Ishida, F. Yergeau, M. J. Düst, M. Wolf, T. Texin, Editors, W3C Recommendation, 15 February 2005, http://www.w3.org/TR/2005/REC-charmod-20050215/ . Última versión disponible en http://www.w3.org/TR/charmod/ .
[CONCEPTS]
Resource Description Framework (RDF): Conceps y Abstract Syntax, G. Klyne, J. J. Carroll, Editors, Recomendación del W3C, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/ . Última versión disponible en at http://www.w3.org/TR/rdf-concepts/ .
[FUNCOP]
XQuery 1.0 y XPath 2.0 Functions and Operators, J. Melton, A. Malhotra, N. Walsh, Editors, W3C Recommendation, 23 January 2007, http://www.w3.org/TR/2007/REC-xpath-functions-20070123/ . Última versión disponible en http://www.w3.org/TR/xpath-functions/ .
[RDF-MT]

RDF Semantics, P. Hayes, Editor, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-mt-20040210/ . Última versión disponible en http://www.w3.org/TR/rdf-mt/ .

[RFC3629]
RFC 3629 *UTF-8, a transformation format of ISO 10646*, F. Yergeau November 2003

[RFC4647]
RFC 4647 *Matching of Language Tags*, A. Phillips, M. Davis September 2006

[RFC3986]
RFC 3986 *Uniform Resource Identifier (URI): Generic Syntax*, T. Berners-Lee, R. Fielding, L. Masinter January 2005

[RFC3987]
RFC 3987, "Internationalized Resource Identifiers (IRIs)", M. Dürst , M. Suignard

[UNICODE]
The Unicode Standard, Version 4. ISBN 0-321-18578-1, en su versión actualizada de vez en cuando por la publicación de nuevas versiones. La última versión de Unicode e información adicional sobre versiones del estándar y sobre la base de datos de caracteres Unicode está disponible en http://www.unicode.org/unicode/standard/versions/.

[XML11]
Extensible Markup Language (XML) 1.1, J. Cowan, J. Paoli, E. Maler, C. M. Sperberg-McQueen, F. Yergeau, T. Bray, Editors, W3C Recommendation, 4 February 2004, http://www.w3.org/TR/2004/REC-xml11-20040204/ . Última versión disponible en http://www.w3.org/TR/xml11/ .

[XPATH20]
XML Path Language (XPath) 2.0, A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, J. Siméon, Editors, W3C Recommendation, 23 January 2007, http://www.w3.org/TR/2007/REC-xpath20-20070123/ . Última versión disponible en http://www.w3.org/TR/xpath20/ .

[XQUERY]
XQuery 1.0: An XML Query Language, S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, J. Siméon, Editors, W3C Recommendation, 23 January 2007, http://www.w3.org/TR/2007/REC-xquery-20070123/. Última versión disponible en http://www.w3.org/TR/xquery/ .

[XSDT]
XML Schema Part 2: Datatypes Second Edition, P. V. Biron, A. Malhotra, Editors, W3C Recommendation, 28 October 2004, http://www.w3.org/TR/2004/REC-xmlschema-2-20041028/ . Última versión disponible en http://www.w3.org/TR/xmlschema-2/ .

[BCP47]
Best Common Practice 47, P. V. Biron, A. Malhotra, Editors, W3C Recommendation, 28 October 2004, http://www.rfc-editor.org/rfc/bcp/bcp47.txt .

Referencias informativas

[CBD]

CBD - Concise Bounded Description, Patrick Stickler, Nokia, W3C Member Submission, 3 June 2005.

[DC]

Expressing Simple Dublin Core in RDF/XML Dublin Core Dublin Core Metadata Initiative Recommendation 2002-07-31.

[Multiset]

Multiset, Wikipedia, The Free Encyclopedia. Article as given on October 25, 2007 at http://en.wikipedia.org/w/index.php?title=Multiset&oldid=163605900. La última versión de este artículo se encuentra en http://en.wikipedia.org/wiki/Multiset.

[OWL-Semantics]

OWL Web Ontology Language Semantics and Abstract Syntax, Peter F. Patel-Schneider, Patrick Hayes, Ian Horrocks, Editors, W3C Recommendation http://www.w3.org/TR/2004/REC-owl-semantics-20040210/. Última versión en http://www.w3.org/TR/owl-semantics/.

[RDFS]

RDF Vocabulary Description Language 1.0: RDF Schema, Dan Brickley, R.V. Guha, Editors, W3C Recommendation, 10 February 2004, http://www.w3.org/TR/2004/REC-rdf-schema-20040210/ . Última versión en http://www.w3.org/TR/rdf-schema/ .

[RESULTS]

SPARQL Query Results XML Format, D. Beckett, Editor, W3C Recommendation, 15 January 2008, http://www.w3.org/TR/2008/REC-rdf-sparql-XMLres-20080115/ . Última versión disponible en http://www.w3.org/TR/rdf-sparql-XMLres/ .

[SPROT]

SPARQL Protocol for RDF, K. Clark, Editor, W3C Recommendation, 15 January 2008, http://www.w3.org/TR/2008/REC-rdf-sparql-protocol-20080115/ . Última versión disponible en http://www.w3.org/TR/rdf-sparql-protocol/ .

[TURTLE]

Turtle - Terse RDF Triple Language, Dave Beckett.

[UCNR]

RDF Data Access Use Cases and Requirements, K. Clark, Editor, W3C Working Draft, 25 March 2005, http://www.w3.org/TR/2005/WD-rdf-dawg-uc-20050325/ . Última versión disponible en at http://www.w3.org/TR/rdf-dawg-uc/ .

[UNISEC]

Unicode Security Considerations, Mark Davis, Michel Suignard

[VCARD]

Representing vCard Objects in RDF/XML, Renato Iannella, W3C Note, 22 February 2001, http://www.w3.org/TR/2001/NOTE-vcard-rdf-20010222/ . Última versión disponible en http://www.w3.org/TR/vcard-rdf .

[WEBARCH]

Architecture of the World Wide Web, Volume One, I. Jacobs, N. Walsh, Editors, W3C Recommendation, 15 December 2004, http://www.w3.org/TR/2004/REC-webarch-20041215/ . Última versión disponible en http://www.w3.org/TR/webarch/ .

[UNIID]

Identifier and Pattern Syntax 4.1.0, Mark Davis, Unicode Standard Annex #31, 25 March 2005, http://www.unicode.org/reports/tr31/tr31-5.html . Última versión disponible en http://www.unicode.org/reports/tr31/ .

[SPARQL-sem-05]

A relational algebra for SPARQL, Richard Cyganiak, 2005

[SPARQL-sem-06]
Semantics of SPARQL, Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez, 2006

F. Agradecimientos (Informativo)

SPARQL Lenguaje de Consulta para RDF es un producto de todo el Grupo de Trabajo para el Acceso a Datos RDF del W3C, y agradecemos las discusiones, comentarios y revisiones a todos los miembros presentes y pasados.

Además, se han tenido comentarios y discusiones con muchas personas a través de los comentarios recibidos en la lista del grupo de trabajo. Todos los comentarios han ayudado a hacer un documento mejor. Andy agradece particularmente a Jorge Pérez, Geoff Chappell, Bob MacGregor, Yosi Scharf and Richard Newman por explorar aspectos concretos relacionados con SPARQL. Eric agradece la ayuda incalculable de Björn Höhrmann.

Registro de cambios

Este es un resumen de alto nivel de los cambios realizados en este documento desde su publicación el 14 de junio de 2007 como Recomendación Candidata:

- In §9 Modificadores de secuencias de solución, el término `solution set` se cambio a `solution sequence`.
- El tipo de medio `application/sparql-query` fue aprobado, por lo que el texto sobre el estado de dicha petición ha sido eliminado.