**SPARQL**

**1. SPARQL란**

W3C는 RDF 그래프에 질문하고 거기에서 정보를 추출하는 SPARQL 언어를 제안한다. 대부분의 SPARQL 쿼리는 기본 차트 패턴 BGP[11]라는 트리플 패턴이 확인되고 있다. SPARQL 쿼리의 트리플 패턴은 RDF 패턴과 비슷하지만, 주어와 술어 및 개체가 변경될 수 있다는 점에서 차이가 있다. Figure 3는 SPARQL query의 예시이다.

|  |
| --- |
| SELECT ?age ?courses  WHERE {  ex:S1 foaf:age ?age.  ex:S1 ex:study ?courses  } |

[ 그림 1 : An example of SPARQL query ]

WHERE절에 있는 트리플 패턴(TP)은 BGP를 만들고, 이러한 패턴들은 변수들은 포함한다(‘?’은 변수를 위해 사용됨). RDF 그래프의 용어를 변수로 대체하여 BGP와 RDF 서브그래프를 매칭 시킬 수 있다. SPARQL query를 실행하려면 차트 패턴 매칭이 필요하다. Figure 2안에 RDF 그래프로 위에 query를 BGP에 매칭 시킨 결과는 다음과 같다.

?age = “25”and?courses = ex:C1.

트리플 패턴은 다음과 같이 형식화 할 수 있다.

- 조건은 만족하는 하나의 튜플을 프리플 패턴이라 불리고, V는 변수들의 한 집합이다.

SPARQL query들은 star, chain, complex 세 가지로 분류될 수 있다. star query는 subject-subject 조인을 포함하고, 조인 변수는 모든 트리플 패턴의 대상이다. chain query는 subject-object 조인을 포함되어 있고(조인은 트리플 패턴의 subject와 다른 트리플 object 사이에 있음), 복잡한 query는 star query와 chain query의 조합을 의미한다.

**2. Advantages**

SPARQL은 사용자들이 "키 값"데이터 또는 보다 구체적으로 말할 수 있는 데이터에 대해 쿼리 할 수 있다. 따라서 전체 데이터베이스는 "주체 예측-객체"트리플 집합의 집합이다. 이는 "document-doculus"라는 용어와 같은 "document-key-value"라는 용어의 사용과 유사하다.  
SQL관계 데이터베이스 조건에서, RDF데이터는 열 칼럼, 술어 기둥, 객체 열을 포함한 3개의 열이 있는 테이블로 간주될 수 있다. RDF의 주제는 특정 비즈니스 객체의 데이터 요소(또는 필드)에 있는 데이터 요소(또는 필드)의 데이터 요소(또는 필드)가 여러개 이상의 테이블에 분산되어 있으며, 고유한 키로 식별됩니다. RDF에서, 이러한 필드는 동일한 주제를 공유하는 별개의 예비 용어로, 동일한 주제를 공유하며, 동일한 키를 사용하여 동일한 키를 가지고 있다. 관계형 데이터베이스와는 달리 개체 열은 이질적인 유형입니다. 즉, 일반적으로 기가 바이트당 데이터 유형은 일반적으로 술어 값으로 암시됩니다. 또한 uniqueSQL은 여러개의 "Personal"항목을 포함할 수 있습니다. 예를 들어"개인"에 대한 "하위"항목을 여러개 가질 수 있으며"하위"와 같은 개체 모음을 반환할 수 있습니다.  
따라서 SPARQL은 별도 스키마 정의가 필요하지 않은 데이터의 본질적인 부분을 위해 스키마가 본질적으로 데이터의 일부인 JOIN, SORT, AGGRAP와 같은 일련의 분석 쿼리를 제공한다. 그러나, 스키마 정보(ontology)는 종종 외부 데이터 셋을 모호하게 할 수 있도록 외부적으로 제공된다. 또한 SPARQL은 그래프로 간주할 수 있는 데이터에 대한 특정 그래프 통과 구문을 제공한다.  
아래 예시는 "friend-off"ontology"friend"라고 불리는 ontology정의"foaf"를 활용하는 단순한 쿼리를 보여 준다.  
특히 다음 쿼리는 데이터 셋에 있는 모든 사람의 이름과 이메일을 반환합니다.

|  |
| --- |
| **PREFIX** **foaf**: <http://xmlns.com/foaf/0.1/>  **SELECT** ?name  ?email  **WHERE**  {  ?person **a** **foaf**:**Person** .  ?person **foaf**:**name** ?name .  ?person **foaf**:**mbox** ?email .  } |

이 쿼리는 일치하는 제목인 "a"와 일치하는 모든 트리플을 포함하고 있으며, 해당 유형은 개인(forton:person)과 사서함(foap:name)과 편지함(foap://)을 가집니다.  
이 쿼리의 저자는 읽기 쉬운 명확성을 위해 변수 이름"?per"를 사용하여 주제를 참조하도록 선택했다. 세번째 요소의 첫번째 요소는 항상 주제이기 때문에, 저자는 "?subj"또는"?x"와 같은 변수 이름을 쉽게 사용할 수 있었을 것이다. 어떤 이름을 선택하든지 간에 물음표는 쿼리 엔진이 동일한 제목을 가진 트리플 헤더에 가입할 것임을 나타내는 각 쿼리의 각 줄에 동일해야 합니다.  
조인의 결과는 일련의 행들인데요?사람, 이름, 이메일입니다. 이 쿼리는?이름과?메일을 반환합니다. 왜냐하면 사람은 대개 사람 친화적인 문자열을 대신하기보다는 복잡한 URI이기 때문입니다. 모든 편지함에는 여러개의 편지함이 있으므로 반환된 세트는 각 편지함마다 한번씩 여러번 나타날 수 있습니다.  
이 쿼리는 SPARQL쿼리 및 반환 결과(SPARQL쿼리 및 반환 결과), 계산된 결과를 취합하여 연합 쿼리라고 알려진 절차에 배포할 수 있다.  
통합된 방식 또는 현지에서 추가적인 세가지 정의를 통해, 쿼리는 예를 들어, 자동차와 같은 다른 종류의 주제에 동참할 수 있고, 예를 들어 연비가 높은 사람들을 위한 이름과 이메일을 반환합니다.

**3. SPARQL의 유형**

데이터베이스에 질의하는 쿼리는 흔히들 사용하는 SELECT문 이외에도 UPDATE, INSERT, DELETE 문들이 있습니다. 마찬가지로 SPARQL에도 쓰임새는 약간 다르지만 이러한 유형들이 존재합니다. 이 유형들은 SELECT, ASK, CONSTRUCT, DESCRIBE입니다. 이중에서 흔히 사용하는 유형은 SELECT입니다.

**3.1 SELECT**

SELECT는 사용자가 찾고자 하는 데이터를 찾는 용도로 사용합니다. 마치 데이터베이스에서 SELECT문으로 질의를 하는 것과 같습니다.

**3.2 ASK**

ASK는 사용자가 찾고자 하는 쿼리의 결과가 있는지 없는지를 boolean(true 또는 false) 값으로 알려줍니다. SELECT문은 쿼리를 결과를 사용자에게 가져다주는 반면에 ASK는 결과가 있으면 true를, 결과가 아무것도 존재하지 않으면 false를 가져다주는 점이 큰 차이점입니다.

**3.3 CONSTRUCT**

CONSTRUCT는 사용자가 찾고자 하는 쿼리를 하면서 동시에 자신이 원하는 템플릿을 입력해줌으로 쿼리의 결과가 사용자가 원하는 템플릿대로 나오도록 할 때 사용합니다. 이는 single RDF Graph라고 하는데 쉽게 말하면 내가 원하는 트리플 모양대로 결과를 반환해 달라는 것입니다.

**3.4 DESCRIBE**

DESCRIBE는 사용자가 찾고자 하는 정보에 대해서 그 정보와 연결된 모든 트리플을 반환받고자 할 때 사용합니다.

이상 4가지의 유형에 대해서 간단하게 설명을 하였습니다. 그러면 이제 제일 많이 사용하는 SELECT를 가지고 좀 더 자세히 살펴보도록 하겠습니다.

**4. SPARQL 만들기**

**4.1 SELECT 사용하기**

SPARQL의 SELECT는 어떻게 사용하는 것일까요? 우선 데이터베이스의 SELECT문을 예로 살펴보겠습니다.

|  |  |
| --- | --- |
| <DBQuery> | SELECT id, name  FROM author  WHERE birthYear=1933  ORDER BY id DESC  LIMIT 3  OFFSET 4 |

위의 문장에서와 같이 질의를 하는 대상, 조건, 정렬조건, 결과건수제한 그리고 찾는 값에 대한 정보들이 있습니다.

SPARQL에서도 마찬가지로 질의하는 대상, 조건, 정렬조건, 결과건수제한 그리고 찾는 값에 대한 정보들을 SPARQL에 표현을 하게 됩니다.

먼저 자세히 알아보기 전에 기억해야 할 것은 SPARQL 쿼리는 기본적으로 트리플 패턴으로 사용자가 찾고자 하는 결과를 찾아옵니다. 트리플, 즉 주어부(subject), 술어부(predicate), 목적부(object)를 구성하는 패턴에 따라 결과를 매칭하게 됩니다.

SPARQL에서 쿼리의 유형은 제일 앞에 나오게 됩니다. 여기서는 당연히 SELECT에 대하여 설명을 하고 있으니 SELECT가 먼저 나오겠죠.

그 다음 찾는 값은 데이터베이스에서 컬럼명이 나오는데 SPARQL에서는 변수라고 지칭하는 문자를 사용합니다. 이 변수들은 특수문자인 ‘?’ 혹은 ‘$’를 사용하여 표현하며 대부분은 ‘?’을 사용하여 표현합니다. 따라서 기본적인 모습은 ‘SELECT ?변수’ 로 만들어집니다.  
**4.2 WHERE 사용하기**

질의하는 대상은 데이터베이스에서는 FROM 절을 사용하여 표현하지만 SPARQL에서는 데이터베이스의 조건에 해당하는 WHERE를 사용하는 점이 다릅니다. 이는 위에서 설명한 트리플 패턴이라는 것으로 결과를 찾는다고 설명하였는데 여기에서 사용이 됩니다. 데이터베이스에서는 author 이라는 테이블에서 조건에 맞는 컬럼값을 가져오라는 질의이지만, SPARQL에서는 그 author 라는 정보가 트리플로 표현되어 있기 때문에 그 패턴을 WHERE절에 작성해 주는 것입니다.  
그럼 여기까지 SPARQL로 표현해보면 아래와 같습니다.

|  |  |
| --- | --- |
| <Query 1> | SELECT ?id  WHERE {  ?id rdf:type <<http://lod.nl.go.kr/ontology/Author>> .  } |

각각에 대한 설명은 아래와 같습니다.

|  |  |
| --- | --- |
| 구분 | 설명 |
| SELECT | 질의 유형 |
| ?id | 찾는 변수(특수문자를 앞에 꼭 붙여야 함) |
| WHERE | 찾는 조건 입력 부분 |
| ?id | 트리플이 주어부에 해당하는 것으로 SELECT 문에서도 사용하고 있는 변수 |
| rdf:type | 트리플의 술어부에 해당하는 것으로 클래스 타입이 무엇인지를 지칭함) |
| <<http://lod.nl.go.kr/ontology/Author>> | 트리플의 목적부에 해당하는 것으로 Author를 가리키는 URI |

이때 WHERE에서는 { }를 사용하여 그 안에 트리플 패턴을 정의하고 있다는 것을 기억해야 합니다. 그리고 하나의 트리플 패턴이 끝날 때에는 마침표(.)를 통해 트리플이 끝났다는 것을 명시해 주어야 합니다.

**4.3 패턴 추가하기**

이번에는 <DBQuery>에서처럼 name을 찾는 SPARQL을 만들어보겠습니다.

데이터베이스에서는 간단하게 컬럼명을 적어주면 되지만, SPARQL에서는 트리플 패턴이라고 말씀드렸듯이 WHERE절에 그 패턴을 지정해 주어야 합니다.

name을 찾기 위해서는 찾는 변수 부분에 ?name을 추가하고 WHERE 절에 패턴을 추가해야 합니다.

국립중앙도서관의 저자정보에는 이름을 <<http://xmlns.com/foaf/0.1/name>>을 사용하여 트리플을 생성하였기에 이 정보를 가지고 패턴을 만들어보면 아래와 같습니다.

|  |  |
| --- | --- |
| <Query 2-1> | SELECT ?id ?name  WHERE {  ?id rdf:type <<http://lod.nl.go.kr/ontology/Author>> .  ?id <<http://xmlns.com/foaf/0.1/name>> ?name .  } |

변수는 ?name 으로 SELECT에 추가하였는데 변수들 구분하기 위해서는 공백만 있으면 됩니다. (데이터베이스처럼 콤마(,)를 통해 구분하지 않습니다)

<Query 2-1>은 저자 중에서 id와 name을 찾는 쿼리입니다. 이를 좀 더 간단하게 표현할 수도 있습니다. 공통된 subject를 가지는 트리플 패턴에서는 공통인 subject를 하나만 사용하고 predicate와 object를 각각 표현하며 그 구분은 세미콜론(;) 으로 합니다.

|  |  |
| --- | --- |
| <Query 2-2> | SELECT ?id ?name  WHERE {  ?id rdf:type <<http://lod.nl.go.kr/ontology/Author>> ;  <<http://xmlns.com/foaf/0.1/name>> ?name .  } |

<Query 2-2>는 세미콜론으로 공통된 subject를 가지는 트리플 패턴을 생성한 것이며 항상 마칠 때에는 콤마를 사용해야 합니다.

4.4 **PREFIX 사용하기**

이제는 <Query 2-2>가 그 전보다 좀 간결해 졌지만 여기서 더 간결하게 생성할 수도 있습니다. <Query 2-2>를 살펴보면 <>를 사용하여 표현하는 부분이 있습니다. 이는 URI를 표현할 때 ‘<’ 와 ‘>’ 문자를 사용하여 그 안에 자원(Resource)이라고 하는 URI를 입력하기 때문입니다. 그런데 또 살펴보면 rdf:type 라는 것이 있습니다. 자원의 URI를 표현하기 위해서는 <> 문자를 사용하여 표현한다고 했는데 이것은 형태가 다른 부분이라고 생각할 수 있습니다만, 이는 PREFIX를 사용하여 표현한 동일하지만 다른 형태의 표현입니다.

rdf:type를 위와 같이 표현하면 <[http://www.w3.org/1999/02/22-rdf-syntax-ns#t ype](http://www.w3.org/1999/02/22-rdf-syntax-ns#type)> 이렇게 표현이 가능합니다. PREFIX를 사용하면 URI를 길게 쓰지 않고 짧게 사용할 수 있기 때문에 간결한 표현이 가능합니다. 그럼 PREFIX를 사용하여 <Query 2-2>를 다시 표현해면 <Query 2-3>과 같습니다.

|  |  |
| --- | --- |
| <Query 2-3> | PREFIX nlon: <<http://lod.nl.go.kr/ontology/>>  PREFIX foaf: <<http://xmlns.com/foaf/0.1/>>  SELECT ?id ?name  WHERE {  ?id rdf:type nlon:Author ;  foaf:name ?name .  } |

이제 WHERE에서 URI를 길게 쓰지 않고 PREFIX를 사용하여 간결하게 입력할 수 있게 된 것입니다. 다른 한 가지 의문은 rdf:type는 PREFIX 선언하는 부분에 rdf가 없는데 어떻게 된 것인지 의문을 가질 수 있습니다. 이는 SPARQL를 처리하는 엔진에서 디폴트로 선언을 해주고 있기 때문에 국립중앙도서관의 SPARQL Endpoint에서는 rdf를 굳이 선언하지 않아도 내부적으로 선언을 하고 있기 때문에 별도의 선언 없이 사용할 수 있습니다.

**4.5 FILTER 사용하기**

데이터베이스에서와 마찬가지로 SPARQL에서도 특정 조건에 맞는 정보를 찾아 볼 수가 있습니다. 이때 사용하는 것이 FILTER입니다. 데이터베이스 질의에서는 WHERE 절에서 비교, 수식 등과 같은 조건을 통해 자신이 원하는 정보를 찾아오듯이 SPARQL에서는 WHERE 절 안에 FILTER를 만들어서 조건을 입력하도록 되어 있습니다. FILTER에는 다양한 연산을 사용하여 조건에 대한 제한을 사용할 수 있으며 여기에서는 심플한 것을 소개하도록 하겠습니다. 보다 다양한 연산은 3장을 참고하시기 바랍니다.

<DBQuery>에서는 조건이 birthYear가 1933으로 제한하고 있습니다. 이를 SPARQL에서 표현하기 위해서는 우선 birthYear라는 변수가 WHERE절의 패턴에서 정의가 되어야 합니다. 왜냐하면 WHERE절에서 트리플의 패턴이 어떤 것인지 정의되어 있지 않으면 FILTER에서도 적용이 되지 않기 때문입니다. 이를 적용하면 아래와 같습니다.

|  |  |
| --- | --- |
| <Query 3> | PREFIX nlon: <<http://lod.nl.go.kr/ontology/>>  PREFIX foaf: <<http://xmlns.com/foaf/0.1/>>  SELECT ?id ?name  WHERE {  ?id rdf:type nlon:Author ;  foaf:name ?name ;  nlon:birthYear ?birth .  } |

<Query 3>의 의미를 풀어보면 ‘?id는 Author라는 집합에 속해 있고, ?id의 foaf:name은 ?name이며, ?id의 nlon:birthYear는 ?birth이다’ 라는 세 가지 패턴을 모두 만족하고 있는 집합에서 ?id와 ?name을 가져오라는 것입니다.

이 쿼리에 birthYear가 1933년생인 ?id를 가져오라는 조건을 추가해보겠습니다. 이 조건은 FILTER를 사용하여 그 안에 조건을 입력하게 됩니다.

|  |  |
| --- | --- |
| <Query 4> | PREFIX nlon: <<http://lod.nl.go.kr/ontology/>>  PREFIX foaf: <<http://xmlns.com/foaf/0.1/>>  SELECT ?id ?name  WHERE {  ?id rdf:type nlon:Author ;  foaf:name ?name ;  nlon:birthYear ?birth .  FILTER(?birth=1933)  } |

FILTER 안에는 ‘(’와 ‘)’를 사용하여 조건을 입력해야 합니다. 그리고 조건을 제한하고자 하는 변수 ?birth와 그 조건을 명시함으로서 FILTER를 완성하게 됩니다.

**4.6 ORDER BY 사용하기**

검색한 결과에 대해서 때로는 정렬이 필요할 때가 있습니다. 이를 위해서 사용하는 것이 ORDER BY입니다. 이는 데이터베이스의 질의와 유사하게 사용됩니다.

먼저 사용되는 곳부터 살펴보면 SPARQL의 WHERE을 닫은 뒤에 ORDER BY를 사용합니다.

<DBQuery> 에서와 같이 id로 정렬을 하도록 만든 SPARQL는 아래와 같습니다.

|  |  |
| --- | --- |
| <Query 5> | PREFIX nlon: <<http://lod.nl.go.kr/ontology/>>  PREFIX foaf: <<http://xmlns.com/foaf/0.1/>>  SELECT ?id ?name  WHERE {  ?id rdf:type nlon:Author ;  foaf:name ?name ;  nlon:birthYear ?birth .  FILTER(?birth=1933)  }  ORDER BY ?id |

정렬 순서는 디폴트로 ASC(오름차순)가 적용되어 있기 때문에 별도로 입력하지 않아도 적용이 됩니다. 하지만 역순으로 정렬을 하고자 할 경우에는 약간의 변화가 필요합니다. ORDER BY 이후에 DESC(내림차순)을 쓰고 변수를 사용해야 합니다. 이때 변수 앞뒤로 괄호(())를 사용하여 변수를 그 안에 넣어야 합니다. 결론적으로 ASC나 DESC 모두 ORDER BY ASC(?변수) 혹은 ORDER BY DESC(?변수) 형태의 SPARQL 입니다

|  |  |
| --- | --- |
| <Query 6> | PREFIX nlon: <<http://lod.nl.go.kr/ontology/>>  PREFIX foaf: <<http://xmlns.com/foaf/0.1/>>  SELECT ?id ?name  WHERE {  ?id rdf:type nlon:Author ;  foaf:name ?name ;  nlon:birthYear ?birth .  FILTER(?birth=1933)  }  ORDER BY DESC (?id) |

정렬은 하나에 대해서만 가능한 게 아니라 다중 정렬도 가능합니다. 다중 정렬을 사용하는 방법은 ORDER BY 이후에 정렬 순서에 따라 변수를 열거하여 질의를 할 수 있습니다. 이때 변수를 열거하는 경우에는 공백을 사용하여 열거하며 정렬순서를 어떤 방식으로 할 것인지에 따라 ASC나 DESC를 사용하여 변수를 열거하게 됩니다.

아래의 <Query 7>은 다중 정렬을 적용한 예시로서 첫 번째 정렬 방식은 name 오름차순으로 하고 두 번째는 id 내림차순으로 정렬을 하겠다는 의미로 사용하였습니다.

|  |  |
| --- | --- |
| <Query 7> | PREFIX nlon: <<http://lod.nl.go.kr/ontology/>>  PREFIX foaf: <<http://xmlns.com/foaf/0.1/>>  SELECT ?id ?name  WHERE {  ?id rdf:type nlon:Author ;  foaf:name ?name ;  nlon:birthYear ?birth .  FILTER(?birth=1933)  }  ORDER BY ?name DESC (?id) |

**4.7 LIMIT 사용하기**

SELECT를 사용하여 정보를 검색하다보면 예기치 않게 많은 수의 결과가 반환될 때도 있습니다. 이럴 경우에 결과의 수를 제한함으로서 원하는 결과를 찾을 수 있도록 설정할 수 있습니다. 결과의 개수를 제한할 때는 LIMIT를 사용하여 매칭이 된 결과 중 사용자가 원하는 개수만큼만 가져오도록 설정을 할 수 있습니다. LIMIT가 사용되는 위치는 ORDER BY 가 끝나고 뒤에 사용하든지 ORDER BY 가 사용되지 않으면 WHERE가 끝나는 뒤에 사용하면 됩니다. <Query 8> 은 <Query 6>의 결과에 개수 제한을 걸어 10개의 결과만 가져오도록 작성한 것입니다.

|  |  |
| --- | --- |
| <Query 8> | PREFIX nlon: <<http://lod.nl.go.kr/ontology/>>  PREFIX foaf: <<http://xmlns.com/foaf/0.1/>>  SELECT ?id ?name  WHERE {  ?id rdf:type nlon:Author ;  foaf:name ?name ;  nlon:birthYear ?birth .  FILTER(?birth=1933)  }  ORDER BY DESC (?id)  LIMIT 10 |

4.8 OFFSET 사용하기

지금까지 조건을 만들고 정렬도 하고 개수 제한도 해보았습니다. 마지막으로는 검색한 결과를 가져올 때 몇 번째 정보부터 가져올 것인지를 정해보도록 하겠습니다. 검색한 결과가 아무런 제한이 없다면 1,000건이 있다고 가정할 때 500번째부터 10개만 가져와서 보고 싶을 경우에 사용이 가능합니다. 이때 사용하는 것이 OFFSET입니다. OFFSET을 사용하면 전체 1,000건의 정보 중에서 500번째 것부터 가져오도록 제한을 둘 수가 있습니다.

아래의 <Query 9>는 120번째부터 10개의 정보를 가져오도록 제한을 두었습니다.

|  |  |
| --- | --- |
| <Query 9> | PREFIX nlon: <<http://lod.nl.go.kr/ontology/>>  PREFIX foaf: <<http://xmlns.com/foaf/0.1/>>  SELECT ?id ?name  WHERE {  ?id rdf:type nlon:Author ;  foaf:name ?name ;  nlon:birthYear ?birth .  FILTER(?birth=1933)  }  ORDER BY DESC (?id)  LIMIT 10  OFFSET 120 |

 4.9 **Query 형태**

지금까지 SPARQL에서 사용하는 기본적은 것들을 살펴보았습니다. 2장에서 설명한 내용은 아주 기본적인 것들이며 대략적인 형태는 아래와 같습니다. 이 중에서 FROM(NAMED)는 약간의 혼동이 생길 수 있음으로 이 설명에서는 제외하였으며 나머지 생소한 용어는 [3장 알아두면 유용한 SPARQL]에서 자세히 다루고 있습니다.

|  |
| --- |
| PREFIX  SELECT / CONSTRUCT / ASK / DESCRIBE  (DISTINCT / REDUCED)  FROM (NAMED)  WHERE  Graph Pattern / OPTIONAL / FILTER / UNION / GRAPH  ORDER BY  LIMIT  OFFSET |