Storage Issues in GENOS

저자 : 김기성, 이태휘, 김형주

요약

생물정보학에서 Gene ontology는 데이터 통합에 있어 중추적 역할을 한다. 이에 Gene ontology를 다루고 활용하기 위한 여러 어플리케이션이 활발히 개발되고 있다. 본 논문에서는 이러한 Gene ontology를 이용한 어플리케이션의 기반 시스템으로써 GENOS를 제안한다. GENOS는 OWL 형태로 표현된 Gene ontology를 저장하고 여러 질의를 효과적으로 처리하기 위한 시스템이다. 이를 위해 RDF 저장소인 Sesame를 사용하고 하부 RDBMS로 Oracle을 사용할 것을 제안한다. 또한 저장소의 관점에서 실제 구현 시에 발생하는 몇 가지 문제에 대해 원인과 해결책을 제안하고 질의 처리를 최적화할 수 있는 방안에 대해 논한다.

키워드: Gene ontology, OWL, RDF, SeRQL

1. 서론

온톨로지란 특정 도메인의 지식을 공유된 개념으로써 명시적으로 정의한 것을 말한다. 지식에 대한 구성원 공통의 명시적 개념화를 제공함으로써 구성원 간 지식의 공유는 물론 기계가 이해할 수 있는 지식의 표현까지 가능해 지고 있다. 현재 온톨로지는 지식기반 시스템에서 중추적인 역할을 담당하고 있다.

생물정보학 분야에서도 이런 온톨로지를 응용하고자 하는 움직임이 활발하다. 특히 Gene ontology[1]는 생물정보학에서의 공통된 용어를 제공함으로써 수많은 데이터가 존재하는 생물정보학 분야에 데이터의 통합의 길을 열어가고 있다.

GENOS는 OWL[2] 데이터로 표현된 Gene Ontology를 효과적으로 처리하기 위한 시스템이다. W3C에서 제안한 OWL은 웹 상의 온톨로지를 표현하기 위한 표준으로 자리매김을 하고 있다. GENOS에서는 OWL 저장을 위해 RDF 저장소를 이용하고, OWL 질의 처리를 효과적이고 손쉽게 하기 위해 래퍼를 구현하였다.

논문은 우선 GENOS의 전체적 구조를 설명하고, 시스템 구성에서 RDF 저장소와 RDBMS 저장소의 채택에 대해 논한다. 또한 GENOS 시스템을 구성하는 데에서 경험했던 문제점과 해결책에 대해 논하고 결론을 내린다.

2. GENOS의 구조

GENOS는 OWL로 표현된 Gene ontology를 다루기 위한 시스템이다. OWL 데이터를 저장하기 위한 저장소로 Sesame[3]를 사용하였으며 하부 RDBMS로는 Oracle을 사용하였다. 각 저장소 채택에 대해서는 다음 절에서 논하겠다. 전체적인 구조는 그림1과 같다.

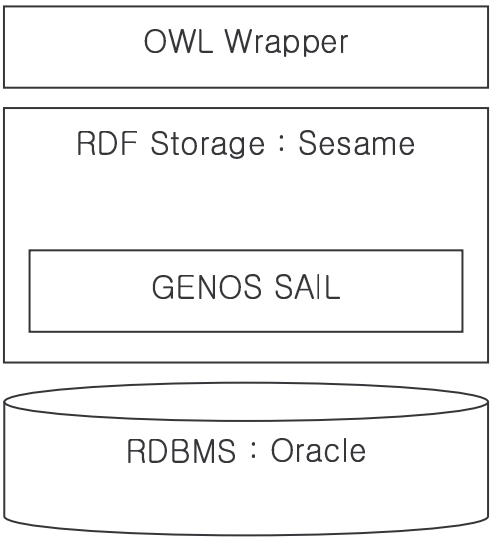

그림 1 GENOS 구조

Gene ontology OWL 데이터는 RDF 저장소를 이용하여 저장한다. OWL 표준 자체가 RDF[4]를 기반으로 하였기 때문에 RDF 저장소를 이용하여 OWL 데이터를 저장하는 데에는 별다른 문제는 없다. 하지만 질의 수행에 있어서 RDF 저장소가 제공하는 질의 처리 능력만으로는 OWL 시멘틱 질의 처리는 어렵기 때문에 RDF 저장소 위에 OWL 질의 처리를 위한 래퍼를 구현하였다. 이 래퍼는 GENOS를 사용하는 어플리케이션이 직접 RDF 질의문을 사용하지 않고도 간편하게 OWL 데이터에 대한 질의를 할 수 있도록 한다.

Sesame는 하부 저장소에 대한 처리를 담당하는 인터페이스로 SAIL을 제공한다. 이 SAIL을 통해 여러 하부 저장방식을 유연하게 적용할 수 있는 확장성을 제공한다. GENOS는 Sesame에서 기본적으로 제공하는 RDFSchema SAIL을 확장한 GENOS SAIL을 사용한다.

2.1 RDF 저장 시스템 선택

현재 여러 RDF 저장소가 개발 되었다. 이들 시스템은 각각 저장 방식, 추론 능력, 질의 언어 등에 대해 차별성을 갖고 있다. GENOS에서 RDF 저장소는 데이터의 저장 및 질의 처리를 담당하기 때문에 RDF 저장소의 선택은 매우 중요하다. GENOS에서는 Sesame를 RDF 저장소로 선택하였다. 초기 단계에서는 Jena[5]를 고려했지만 다음과 같은 이유로 Sesame를 사용하게 되었다.

우선 Sesame는 클래스 계층 구조에 대한 질의를 지원한다. Jena와 Sesame는 모두 RDF 데이터를 파싱하여 triple statement를 한 개의 table에 저장하는 구조를 취한다. 그러나 Sesame는 RDF schema의 정보를 저장할 table을 만들어 클래스 계층 구조 정보를 따로 저장한다. 따라서 Sesame에서는 클래스 계층 구조에 대한 질의는 이 table을 사용하여 빠르게 처리 할 수 있는 반면에, Jena에서는 statement table상에서 검색 해야 하는 단점이 있

다. Gene ontology의 각 term 들은 rdfs:class로 정의되어 있기 때문에 클래스 계층 구조에 대한 질의가 주를 이룬다. 따라서 스키마에 대한 질의 처리를 지원하는 Sesame의 질의 처리 능력이 Jena보다는 유용할 것으로 판단되었다.

또한 질의 표현에 있어서도 Jena는 RDQL[6]만을 지원하는 반면, Sesame는 RQL[7]과 SeRQL[8]을 지원한다. RDQL을 사용하면 "어떤 클래스의 모든 조상을 찾아라" 와 같은 질의는 여러 번의 질의 수행 후 모든 결과를 찾을 수 있다. 반면 SeRQL을 사용하면 class hierarchy에 대한 질의를 직접적으로 표현할 수 있기 때문에 사용이 편리하다.

마지막으로 대용량 OWL 저장소에 대한 벤치 마크 논문[9]에서 Jena는 질의 수행이 끝나지 않는 경우가 있어 벤치 마크에서 제외었다. 반면 sesame는 저장 속도에서 성능 저하를 보이긴 하였지만 비교적 우수한 성능을 보였다. 이 논문에서는 또 다른 OWL 저장소인 DLDB[10]가 우수한 성능을 보였지만 아직은 Sesame와 같은 체계적인 API를 제공하지 못하고 있기 때문에 고려 대상에서 제외하였다.

2.2 하부 RDBMS의 선택

Sesame는 하부 RDBMS로 Oracle, MySQL, PostgreSQL 등을 지원한다. GENOS에서는 Oracle을 하부 RDBMS로 선택하였다. Oracle을 선택한 이유는 대용량 데이터 처리에 적합하다는 판단에서이다. 또한 Oracle은 SQL에서 재귀질의를 지원하는 CONNECT BY 제공한다. 이 CONNECT BY를 이용하면 트리 구조의 데이터에 대한 탐색을 SQL을 통해 간단히 할 수 있다. CONNECT BY의 사용은 3절에서 좀더 자세히 설명하겠다.

3. Sesame 시스템과 Oracle 사용시 문제점 및 해결 방안

이번 장에서는 Sesame 시스템에 Oracle을 적용할 때 발생하는 문제점과 해결방안에 대해 논하겠다. 먼저 밝힐 것은 Sesame의 Oracle 모듈은 Sesame 개발사인 Aduna[11]에서 직접 개발한 것이 아닌 third-party에서 개발하였다. 따라서 Oracle 모듈 상의 문제는 개발사와는 직접적인 관계가 없다.

3.1 저장 시간

처음 Sesame를 사용하여 GO OWL 데이터를 저장하였을 때 가장 큰 문제는 저장 속도가 매우 느리다는 것이었다. 그림 2는 초기 Sesame의 시간에 따른 저장된 statement 개수 그래프와 개선된 저장속도를 나타내는 그래프이다. 초기 Sesame에 Oracle을 사용하였을 때는 MySQL을 사용했을 때에 비해 10배 이상 느린 속도로 저장되었다.

쿼리 로그 분석 결과 조인 연산의 질의 수행 계획에 문제가 있기 때문인 것으로 밝혀졌다. 저장 속도 그래프를 보면 후반기에 속도변화가 있는 것을 볼 수 있다. 이는 질의 수행 계획이 변했기 때문이다. 초기에는 merge 조인을 사용하였기 때문에 table을 full scan하였지만 후반기부터는 index nested loop 조인을 사용한다. 따라서 초기의 질의 수행 역시 index nested loop을 사용하도록 유도하여 주면 저장속도가 빨라지게 된다. 이를 위해 Sesame의

Oracle 모듈에 주기적으로 테이블의 통계정보를 계산하도록 하는 질의를 수행하도록 수정하였다. 이 질의는 다음과 같다.

analyze table <테이블명> estimate statistics;

이를 통해 데이터 저장 속도를 개선할 수 있었다.

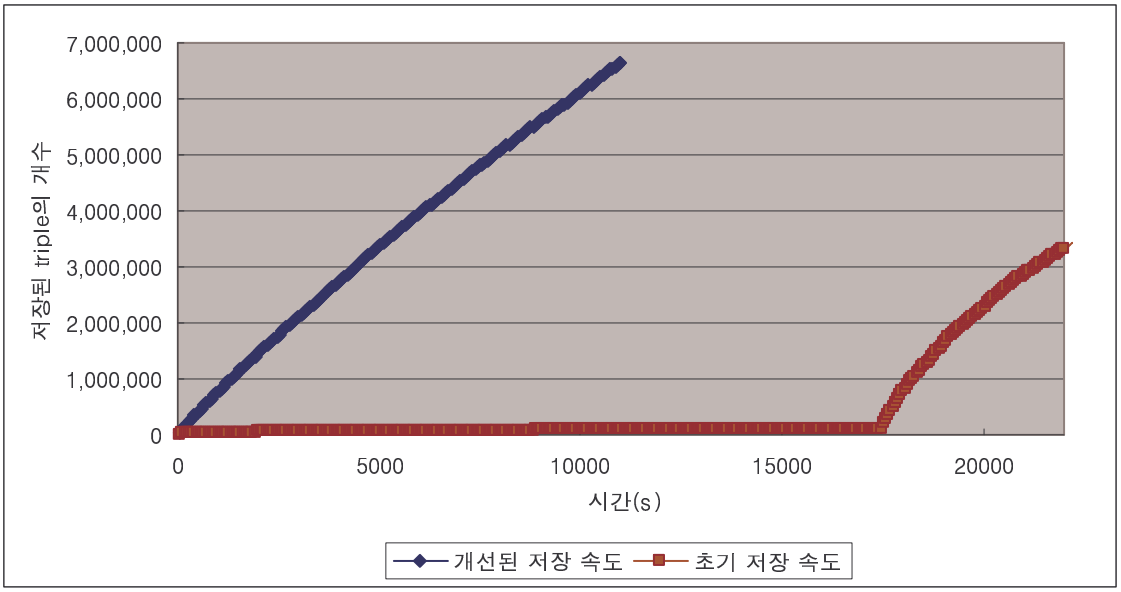

그림 2 Sesame-Oracle 저장 속도

3.2 NULL값 처리의 문제

Sesame와 Oracle을 사용하여 OWL데이터를 저장하였을 때 발생한 또 다른 문제점은 triple의 object로 literal을 취하는 triple statement가 생성되지 않는 것이었다. 이 문제점은 Oracle의 NULL 값 처리와 관련된 문제였다. Oracle에서는 NULL값과 NULL값은 같은 값으로 취급하지 않는다. 따라서 조인 조건식의 컬럼에 NULL값이 있는 튜플은 조인의 대상에서 제외된다. 결국 NULL값과 NULL값을 같다고 보는 다른 RDBMS(ex. MySQL)와 같은 테이블에 대해 조인의 결과가 달라진다. Sesame에서 literal이 object로 있는 statement를 생성할 때에 이와 같은 NULL값을 가진 튜플에 대한 조인이 필요하였고 따라서 올바른 결과가 생성되지 않았었다.

이를 해결하기 위해서 다음과 같은 방법을 생각할 수 있다. 만일

table1.column1 = table.column1

와 같은 조인 조건식이 있는 경우 이 컬럼에 NULL값이 있는 경우에는 다음과 같이 바꿔

준다.

table1.column1 = table.column1 or
(table1.column1 IS NULL and table2.column1 IS NULL)

즉, 두 컬럼이 모두 NULL값을 갖는 경우에도 조인이 되도록 바꾸는 것이다. 이와 동일한 효과를 다음과 같은 방법으로도 얻을 수 있다.

nval(table1.column1, '$') = nval(table2.column1,'$')

nval 연산식은 Oracle에서 제공하는 연산식으로 첫번째 인수가 NULL값이면 두 번째 인수로 지정된 값을 반환한다. 따라서 두 컬럼이 모두 NULL값인 경우에도 조인 결과로 나올 수 있다. 이때 두 번째 인수는 대상 컬럼에 없는 값을 지정하도록 주의하여야 한다.

두 가지 방법 중 nval을 사용하는 것이 좀더 효율적인 질의 수행 계획을 유도 할 수 있다. Oracle에서는 NULL 값은 인덱스를 만들 때에 제외가 된다. 따라서 첫 번째 방법을 사용할 경우 인덱스 사용의 효율이 낮다고 판단하여 비효율적인 질의 수행 계획을 따를 수 있다.

3.3 CONNECT BY 사용

Oracle은 SQL에서 재귀적 질의 처리를 표현하는 CONNECT BY 절을 제공한다. 이 절을 사용하면 계층적인 구조를 가지고 있는 데이터에 대해서 질의를 할 때, 효과적으로 질의 결과를 얻어낼 수 있다.

예를 들어, 트리 구조를 가지고 있는 데이터에 대해서 어떤 한 노드에서 다른 노드까지 연결되는 모든 경로를 구하고자 할 때, 기본적인 SQL을 이용하여 결과를 얻으려면 한 단계씩 탐색해 가며 질의를 반복적으로 수행해야 한다. 이러한 경우 Oracle의 CONNECT BY 절을 이용하면, 질의 결과를 한 SQL을 통하여 간단하고 효과적으로 얻어낼 수 있다. Oracle 8i에서는 CONNECT BY 절에 조인 연산이 포함될 수 없었으나, Oracle 9i부터는 이러한 제한이 없어져서 유용하게 사용될 수 있다.

CONNECT BY 절을 이용하기 위해서는 탐색 시작 지점과 만족시켜야 하는 탐색 조건을 지정해 주어야 한다. 탐색 시작 지점과 탐색 조건에 따라 top-down과 bottom-up 탐색을 결정할 수 있다. top-down 방향으로 탐색하기 위해서는 CONNECT BY PRIOR parent_column = child_column으로, bottom-up 방향으로 탐색하기 위해서는 CONNECT BY PRIOR child_column = parent_column으로 질의를 하면 된다. 또한, Oracle 9i 에서는 SYS_CONNECT_BY_PATH 라는 새로운 함수가 추가되어, 조건에 맞는 데이터뿐만 아니라, 시작 지점에서 해당 데이터까지의 경로들을 전부 결과로 얻을 수 있다.

GENOS에서는 Gene ontology의 term 계층도를 만드는 데에 CONNECT BY를 사용하였다. term 계층도는 선택한 term의 root term에서부터의 모든 경로를 나타내주는 것이다. 이

를 위해 다음과 같은 SQL을 사용하면 한번에 모든 경로를 찾을 수 있다.

```
SELECT DISTINCT sys_connect_by_path(super, '/') path_from_root
FROM
    (SELECT * FROM triples
    WHERE (predicate = 19 or predicate = 877 or object = 887) and explicit =1)
WHERE subject = [ROOT TERM ID]
START WITH subject = [TERM ID]
CONNECT BY NOCYCLE PRIOR subject = object;
```

위와 같은 경우 Sesame의 triples table에서 ROOT TERM ID로 지정한 term에서부터 TERM ID로 지정한 term까지의 연결된 경로를 모두 검색해준다. 이때 검색의 범위를 줄이기 위해 FROM절 부분을 SELECT를 사용하였다. 또한 ROOT TERM부터 검색할 경우 경로가 매우 많기 때문에 bottom-up 방식의 탐색을 하도록 하였다. 좀더 질의 처리를 빠르게 하기 위해서는 FROM 절의 SELECT를 실체화 뷰를 사용할 수 있다. 이때 이 뷰는 term간의 subclassOf 관계 정보를 갖는 뷰가 된다. NOCYCLE 옵션은 탐색 도중 사이클이 발생하였을 때 탐색이 끝나지 않는 경우를 위해 사용한다. Gene ontology 데이터는 term 계층 관계가 DAG 형태로 사이클이 존재 할 수 있다. 따라서 NOCYCLE 옵션을 사용하여야 한다.

그림 3은 CONNECY BY를 사용했을 때와 여러 SQL을 재귀적으로 호출하였을 때의 질의 속도를 나타낸다. 각 질의 속도는 연속적으로 20번 수행 후의 평균치를 구한 것이다. CONNECT BY를 사용했을 때 질의 부분을 간단히 처리할 수 있는 장점 외에 성능의 향상을 볼 수 있음을 알 수 있다.

Method	Time (ms)
Naïve approach	270.4
Connect By (with Select)	181.9
Connect By (with Materialized View)	167.9

그림 3 Term 계층도 계산 수행 시간

4. SeRQL 질의 처리 최적화

SeRQL은 RQL을 확장시킨 질의언어로 Sesame에서 지원하는 새로운 RDF 질의 언어이다. SeRQL은 label()이란 연산자를 제공한다. 이번 장에서는 label() 연산자에 대한 질의 수행을 최적화하는 방안에 대해 설명하겠다.

label() 연산자는 literal에 대해 label 정보만을 필요로 한다고 지정을 해주는 연산자이다. 다음은 Geneontology term, “nucleus“와 annotation 관계가 있는 모든 Gene Product의 이

름을 검색하는 SeRQL 질의문이다.

```
SELECT DISTINCT
        GENESYMBOL
FROM
        {<go:GO_0005634>} go:association {} go:gene_product {}
         go:name {GENESYMBOL}
```

이 질의에서 구하고자 하는 값은 그래프 패턴을 만족시키는 변수 GENESYMBOL이다. GENESYMBOL에 해당하는 값은 Sesame에서 literal로 저장 되어있다. 따라서 질의 결과는 literal 객체가 된다. 이 literal 객체는 literal의 label 뿐만 아니라 datatype, language 등의 정보를 담고 있다. 그러나 만일 literal의 label만 필요로 할 경우에는 label() 연산자를 사용하면 된다. 즉 다음과 같이 질의문을 작성한다.

```
SELECT DISTINCT
        label(GENESYMBOL)
FROM
        {<go:GO_0005634>} go:association {} go:gene_product {}
         go:name {GENESYMBOL}
```

그러나 이 두 가지 질의문에 대해 Sesame는 동일한 과정을 거쳐 질의 결과를 구한다. Sesame의 질의 처리 과정은 다음과 같다. 우선 FROM 절의 그래프 패턴을 만족시키는 모든 변수의 ID를 검색한다. 이 ID는 변수의 type에 따라 resource ID 일수도 있고 literal ID 일수도 있다. 그 후 각각의 resource ID 또는 literal ID 대해 resources, literals table의 내용을 가져오는 SQL질의를 수행하게 된다. 만일 1000개의 결과 행이 있다면 1000번의 select 질의를 수행하여 질의 결과에 대한 내용을 가져오게 되는 것이다. 따라서 결과가 매우 많은 질의를 수행할 경우 그래프 패턴에 해당하는 resource, literal의 ID를 찾는 시간보다 그 ID에 해당하는 내용을 가져오는 시간이 더욱 길어지게 된다. 위의 질의의 경우 “nucleus” term에 annotation된 gene product의 개수가 9,263개이며 질의 수행은 약 155초 정도 걸린다. 이처럼 모든 결과의 resource, literal에 대해 select문을 통해 내용을 추출하는 것은 매우 비효율적이다. 각각의 질의 수행은 적은 시간으로 수행되지만, 한 질의를 수행하기 위해 필요한 작업들이 불필요하게 중복 되기 때문이다.

그러나 label() 연산자로 지정된 variable에 대해서는 위의 과정을 거치지 않고도 빠르게 질의를 수행할 수 있다. 이는 질의 수행 시 Sesame 내부에서 생성하는 SQL문을 수정하는 과정을 약간 변경하면 된다. 그림 4는 Sesame에서 위의 SeRQL에 대해 생성하는 SQL과 수정된 SQL을 나타낸다. 수정된 SQL에서 초기의 SQL과 다른 부분에는 밑줄로 표시하였다.

Sesame의 SQL	수정된 Sesame의 SQL
SELECT DISTINCT t2.object FROM triples t0, triple2 t1, triples t2 WHERE t0.subject = 1435764 AND t0.predicate = 878 AND t1.subject = t0.object AND t1.predicate = 833 AND t2.subject = t1.object AND t2.predicate = 884	SELECT DISTINCT l0.label FROM triples t0, triple2 t1, triples t2, literals l0 WHERE t0.subject = 1435764 AND t0.predicate = 878 AND t1.subject = t0.object AND t1.predicate = 833 AND t2.subject = t1.object AND t2.predicate = 884 AND t2.object = l0.ID

그림 4 SeRQL에 대한 중간 SQL

초기의 SQL에서는 literal의 ID을 추출하지만 수정된 SQL은 바로 literal의 label을 추출하게 된다. 각각의 방법으로 SeRQL을 수행한 결과는 그림 5와 같다. 좀더 정확한 질의 수행 시간의 비교를 위해 20번을 연속으로 질의 수행을 하여 평균 시간을 계산하였다. 결과에서 볼 수 있듯이 질의문의 결과가 많을수록 위의 방법의 효과가 큼을 알 수 있다. 또한 대부분의 SeRQL 질의문의 경우 literal의 label만을 원하는 경우가 많기 때문에 전반적인 성능향상을 볼 수 있다.

초기 수행 시간(ms)	개선된 수행 시간(ms)
157935	3023

그림 5 SeRQL 수행 속도

5. 결론

GENOS 시스템은 Gene Ontology데이터를 다루기 위한 시스템으로 Gene ontology를 사용하는 어플리케이션을 만들기 위한 기반 시스템으로 활용할 수 있다. 이를 위해 Sesame RDF 저장소와 Oracle RDBMS를 사용하는 방안을 제시하였으며 실제 구현상에서 나타나는 문제점들에 대해 논하였다. 구현에 사용된 Sesame는 1.1 버전이었으며 현재 최신 버전은 1.1.2 이다. Sesame 1.1.2 버전에서는 Oracle 모듈에서 본 논문에서 언급한 저장속도 개선 방안이 반영되어 있다. 향후 계획으로는 OWL 래퍼의 기능을 강화하여 추론 룰을 처리하는 리즈너의 기능을 추가할 것이며 데이터 로딩 시간을 개선 등도 해결해야 할 문제이다.

참고문헌

[1] The Gene Ontology Consortium, “Creating the Gene Ontology Resource: Design and Implementation”, Genome Research, 11(8), pp1425-33, 2001

[2] OWL – Web Ontology Language, http://www.w3c.org/2004/OWL

[3] RDF – Resource Description Framework, http://www.w3c.org/RDF

[4] Jeen Broekstra, Arjohn Kampman, Frank van Harmelen, Sesame: A Generic Architecture for Storing and Querying RDF ad RDF schema, ISWC, 2002

[5] JK. Wilkinson, C. Sayers, H. Kuno, D. Reynolds, Efficient RDF Storage and Retrieval in Jena2, SWDB 2003

[6] RDQL – A Query Language for RDF, http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/

[7] G. Karvounarakis, S. Alexaki, V. Christophides, D. Plexousakis, M. Scholl, “RQL: A Declarative Query Language for RDF”, WWW2002.

[8] Jeen Broekstra, Arjohn Kampman, “SeRQL: An RDF Query and Transformation Language”, http://openrdf.org, 2004

[9] Y. Gui, Z. Pan, J. Heflin, An Evaluation of Knowledge Base Systems for Large OWL Datasets, ISWC, 2004

[10] Z. Pan., J. Heflin, DLDB: Extending Relational Databases to Support Semantic Web Queries, ISWC, 2003

[11] Aduna, http://aduna.biz