**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**Viện Công nghệ thông tin và Truyền thông**



**BÁO CÁO BÀI TẬP LỚN**

**Môn: LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG**

ĐỀ TÀI 1 : Sử dụng Virtuoso

Mã Lớp : 104403

Nhóm sinh viên thực hiện: nhóm 6

|  |  |  |  |
| --- | --- | --- | --- |
| STT | Họ và tên | MSSV | % đóng góp |
| 1 | Nguyễn Hoàng Thuyên | 20163968 | 40% |
| 2 | Lê Duy Anh |  | 30% |
| 3 |  |  | 10% |
| 4 |  |  | 10% |
| 5 |  |  | 10% |

Giảng viên hướng dẫn: TS. Trịnh Tuấn Đạt

Hà Nội

# ĐÓNG GÓP THÀNH VIÊN TRONG NHÓM

|  |  |  |
| --- | --- | --- |
| Thành viên | Phần trăm đóng góp | Nhiệm vụ |
| Nguyễn Hoàng Thuyên | 40% | Thiết kế phân chia gói, lớp, code phần sinh dữ liệu , add dữ liệu lên server (gói InsertData), gói entity và hàm main.  Code phần các câu truy vấn, tính thời gian truy vấn. (class Query)  Viết báo cáo phần phương pháp giải quyết, biểu đồ gói, biểu đồ lớp, đánh giá kết quả (ưu nhược điểm). |
| Lê Duy Anh | 30% | Tạo các dữ liệu dạng file text, tìm hiểu về công nghệ Virtuoso.  Viết báo cáo phần cách cài đặt trên Window, Ubuntu, báo cáo phần các kỹ thuật như kết nối với VirtuosoServer và cách truy vấn. |
|  | 10% | Tham gia lên ý tưởng và nghiên cứu tài liệu |
|  | 10% | Tham gia lên ý tưởng và nghiên cứu tài liệu |
|  | 10% | Tham gia lên ý tưởng và nghiên cứu tài liệu |

# CHƯƠNG I : TỔNG QUAN VỀ CÔNG NGHỆ

## I. Giới thiệu về công nghệ Virtuoso

Virtuoso cung cấp một “Dữ liệu Junction Box” mà các ổ đĩa doanh nghiệp và sự nhanh nhẹn cá nhân bằng cách phát sinh một Semantic Web của dữ liệu liên kết từ các hầm chứa dữ liệu hiện có.

Virtuoso là một giải pháp hiện đại cho việc truy cập dữ liệu, ảo hóa, hội nhập và đa mô hình quản lý cơ sở dữ liệu quan hệ (bảng SQL hoặc đồ thị RDF).

* **Nhanh nhẹn thông qua việc khai thác trí tuệ nhân tạo** : tận dụng lý luận tường thuật và khả năng suy luận đó cho thấy nhiều máy tính nhận thức như một phần của lõi RDBMS chức năng.
* **Web Application Deployment**: có hỗ trợ ảo hóa của các hành động khác nhau trên một loạt các API liên quan đến Web Services.
* **Privacy dữ liệu & An ninh**: qua sử dụng kiểm soát truy cập thuộc tính dựa trên hạt mịn để cung cấp truy cập dữ liệu vô song và an ninh thao tác thông qua sử dụng các tiêu chuẩn mở hiện có như TLS và HTTPS
* **Tối đa hóa hệ thống:** bảo vệ khả năng trộn và kết hợp “tốt nhất của lớp” giải pháp bằng cách đảm bảo tất cả các chức năng cốt lõi dựa trên các tiêu chuẩn mở hiện có (URI, URL, HTTP, SPARQL, SQL, ODBC, JDBC, ADO.NET, OLEDB, XMLA, XQuery, XPath, XSLT).

## II. Một số thuật ngữ

Chúng ta cần làm rõ một số thuật ngữ trước khi tiến hành.

* "RDF" là http://www.w3.org/RDF được xác định bởi http://www.w3.org (W3C). Nó cung cấp một phương tiện đơn giản để mô tả các đối tượng tài nguyên đa diện và để liên kết chúng thành các biểu đồ quan hệ phức tạp. AllegroGraph Server tạo, tìm kiếm và quản lý các đồ thị RDF như vậy.
* "URI" là [Uniform Resource Identifier](http://www.ietf.org/rfc/rfc2396.txt). Nó là nhãn được sử dụng để xác định duy nhất các loại thực thể khác nhau trong biểu đồ RDF. Một URI điển hình trông rất giống địa chỉ web: http://example.org/project/ class#number. Mặc dù giống nhau, URI có thể là địa chỉ web hoặc không tùy người dùng định nghĩa. Nó chỉ đơn giản là một nhãn. URI được dùng với bộ mã ASCII còn IRI (được coi là bản nâng cấp của URI được dùng với mã UNICODE.
* "Triple" là một câu lệnh dữ liệu, được lưu trữ ở định dạng RDF. Nó nói rằng một tài nguyên có một thuộc tính với một giá trị. Nó bao gồm 3 trường:
  + Subject: Trường đầu tiên chứa URI xác định duy nhất tài nguyên mà bộ ba này mô tả.
  + Predicate: Trường thứ hai chứa URI xác định một thuộc tính của tài nguyên này, chẳng hạn như màu sắc hoặc kích thước của nó hoặc mối quan hệ giữa tài nguyên này với tài nguyên khác, chẳng hạn như nguồn gốc hoặc quyền sở hữu.
  + Object: Trường thứ ba là giá trị. Nó có thể là một giá trị theo nghĩa đen, chẳng hạn như "màu đỏ" hoặc URI của tài nguyên được liên kết.
* "Resource description" được định nghĩa là một tập hợp các triples mà tất cả đều có cùng một URI trong trường chủ đề. Nói cách khác, triples đều mô tả các thuộc tính của cùng một thứ.
* Statements là một đối tượng Java phía máy khách mô tả triples (quad, quint).

## III. Hướng dẫn cài đặt Virtuoso (phiên bản 7.2)

### Trên Linux - Ubuntu

* Tạo thư mục virtuoso chứa các gói cài đặt với lệnh sau:
* *sudo mkdir virtuoso*
* Lên trang chủ tải Virtuoso tại đường dẫn: <https://shop.openlinksw.com/license_generator/virtuoso-download/>

(Chọn bản 7.2 giành cho Linux)

* Khi đó ta được file lovpz2zz.tar
* Copy file mới tải về vào thư mục “virtuoso” vừa tạo bên trên với câu lệnh:
* *sudo cp lovpz2zz.tar /virtuoso*
* Sau đó giải nén file mới copy vào thư mục “virtuoso” với câu lệnh sau:
* *sudo tar -xvf lovpz2zz.tar*
* Sau khi giải nén xong ta sẽ được 2 file đó là install.sh và universal-server.taz
* file install.sh là file để ta cài đăt Virtuoso, ta thay quyền cho nó bằng lệnh:
* *sudo chmod +x install.sh*
* Sau đó ta chạy file install.sh (nó sẽ tạo ra rất nhiều file) bằng lệnh sau:
* *./install.sh*
* Sau câu lệnh đó thì ta đã cài đặt hoàn thành, nó sẽ tự truy nhập trên Ứng dụng truy cập Web mặc định để vào thay đổi tài khoản và mật khẩu (ta tắt nó đi, coi như mặc định tên tài khoản là dba và mật khẩu cũng là dba)
* Để chạy server ta vào thư mục /bin chạy lệnh sau:
* *sudo ./virtuoso-start.sh (tương tự lệnh sudo ./virtuoso-stop.sh để dừng server)*
* Để kiểm tra Server có hoạt động hay không bạn hăy vào trang [http://localhost:8890](http://localhost:88905) để kiểm tra, nhập tên với mật khẩu dba đã mặc định ở trên tạo ở mục trên.
* Để gỡ Virtuoso, ta chỉ cần xóa thư mục “virtuoso” với lệnh sau:
* *sudo rm -rf virtuoso*

### Trên Windows

* *Bước 1*: Truy cập link <https://shop.openlinksw.com/license_generator/virtuoso-download/> .
* *Bước 2*: chọn phiên bản và hệ điều hành tương ứng, sau đó bấm download và tiến hành cài đặt.
* *Bước 3*: trên trình duyệt web gõ localhost:1111/ nếu hiển thị như bên dưới thì thành công.

## IV. Các bước chuẩn bị

### Các bước chuẩn bị các API

* Bước đầu tiên ta cần tạo 1 project maven:

Chọn File menu, chọn New -> New Project… . Sau đó chọn Maven Project. Chọn tùy chọn “Create a simple project” và nhấn Next.

Trong màn hình tiếp theo bạn phải xác định thêm chi tiết về Maven project như group id-, artifact id-, name- với các thông số dưới đây bạn có thể điền tùy ý:

group id (ví dụ: org.example)

artifact id (ví dụ: rdf4j-getting-started)

name (ví dụ: HelloRDF4J)

Sau đó nhấn Finish để hoàn tất.

* Để import các API vào project, bạn thực hiện các bước sau:
* Download file (virt\_rdf4j.jar và virtjdbc4\_2.jar) tại đường link sau:

http://vos.openlinksw.com/owiki/wiki/VOS/VOSDownload

* Chọn chuột phải vào project, chọn Build path -> Configure build path.
* Chọn Libraries và chọn Add JARs…
* Sau đó them 2 file đã tải ở trên.
* Sau khi thêm ta cần update lại project: Chuột phải vào project của bạn chọn Maven >Update project.

### Kết nối Virtuoso

* Để kết nối với máy chủ ta cần cung cấp IP address và port number như là 1 chuỗi.
* Ngoài ra cần cung cấp username và password đăng nhập vào Virtuoso:

Repository myRepository1 = new VirtuosoRepository("jdbc:virtuoso://localhost:1111", "dba", "dba");

RepositoryConnection = myRepository1.getConnection();

* Để đóng connection ta sử dụng phương thức close() của đối tượng connection.

### Cách truy vấn sử dụng công nghệ Sparql

* Virtuoso lưu trữ dữ liệu dưới dạng các “triples”, lưu trữ theo định dạng RDF với 3 trường subject-predicate-object. Để truy xuất các “RDF triples”, ta sử dụng truy vấn SparQL.
* Ví dụ về 1 câu lệnh SparQL: “SELECT ?s ?p ?o WHERE {?s ?p ?o .}”
* Mệnh đề SELECT trả về biến ?s, ?p, ?o. Các biến được liên kết với các giá trị subject, predicate, object của 1 “triple” thỏa mãn trong mệnh đề WHERE. Dấu chấm (.) biểu thị sự kết thúc cho ràng buộc.
* Ví dụ 1 câu lệnh sử dụng SPARQL filter match:

Florentia Muino đã đến thăm những nước nào?:

String query = "PREFIX label:<http://www.example.org/ontology/> \n";

+= "PREFIX re:<http://www.example.org/relationship/> \n";

+= "SELECT ?s ?p ?o \n";

+= "WHERE { \n";

+= " ?person label:label ?s .\n";

+= " ?person ?p ?country .\n";

+= " ?country label:label ?o .\n";

+= " FILTER regex(?s, \"Florentia Muino\") ";

+= " FILTER (?p =re:visit)";

+= "}";

* Phương thức: prepareTupleQuery(QueryLanguage.SPARQL, queryString) của đối tượng RepositoryConnection với tham số đầu vào là ngôn ngữ sử dụng (ở đây là SPARQL) và câu lệnh. Phương thức này trả về 1 đối tượng TupleQuery.
* Sau đó sử dụng phương thức evaluate() của TupleQuery để trả về 1 tập kết quả tương tự như ResultSet trong JDBC, nó trả về 1 TupleQueryResult mà ta có thể truy xuất kết quả với phương thức getValue().

VD:

TupleQuery tupleQuery = connection.prepareTupleQuery(QueryLanguage.SPARQL, query);

TupleQueryResult result = tupleQuery.evaluate();

while (result.hasNext()) {

BindingSet bind = result.next();  
 Value s = bind.getValue("s");  
 Value p = bind.getValue("p");

Value o = bind.getValue("o");

System.out.prindln(“%s %s %s”, s, p, o);

}

# CHƯƠNG II : PHƯƠNG PHÁP GIẢI QUYẾT

## I. Cấu trúc dữ liệu

* Ở đề tài này, nhóm cũng sẽ sử dụng cấu trúc lưu trữ dưới dạng các triples (s-p-o) để lưu trữ các thực thể cũng như quan hệ (lưu trữ trên Virtuoso).
* Ý tưởng ở đây là ta sẽ lưu các thực thể ở dạng id-att-value.  
   Ví dụ: s : Person1 p : label o : “Hoàng Thuyên”
* Từ đó suy ra mỗi thực thể sẽ chiếm tầm 5-6 triples tùy thuộc vào loại thực thể đó có bao nhiêu thuộc tính.
* Còn về quan hệ, thì mỗi quan hệ còn lưu 1 triples kết nối 2 thực thể ở dạng : idEntity1 – nameRelation – idEntity2.

Ví dụ: s:Person1 p:”visit” o:Country1

## II. Giải thuật sinh dữ liệu

* Để sinh ngẫu nhiên dữ liệu giả lập lên server, ta sẽ phải có 1 số file lưu danh sách các thuộc tính của các thực thể, quan hệ. Qua đó ta sẽ đọc file để lưu các thuộc tính đó vào các List<String>.
* Viết 1 hàm trả khởi tạo ngẫu nhiên các thực thể, sau đó tạo IRI kết hợp thành các statements (dạng triples) và add nó lên server.
* Mỗi lần tạo ra được 1 thực thể ngẫu nhiên, ta sẽ thêm nó vào List<IRI>
* Cứ tiếp tục như vậy cho đến khi đủ số lượng thực thể.
* Sau khi đã thêm xong thực thể, ta đã có 1 List<IRI>, từ đó để tạo quan hệ, ta lấy ngẫu nhiên 2 phần tử trong List<IRI> rồi ghép nó với 1 quan hệ nào đó trong List<String> quan hệ đã khởi tạo từ đầu. Sau đó tạo IRI kết hợp thành các statements (dạng triples) và add nó lên server.
* Để kết hợp chúng, ta cần có các namespace tương ứng với các thuộc tính. Để cụ thể hơn, ta có ví dụ sau:

Ví dụ:

* Câu lệnh dưới là câu lệnh tạo IRI thực thể person kèm theo Id của thực thể đó.

IRI iri = valueFactory.createIRI(nameSpace, person.getId());

* Câu lệnh dưới tạo biến Literal label của thực thể person trên.

Literal label = valueFactory.createLiteral(person.getLabel());

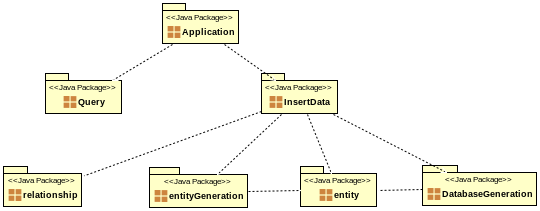
* Câu lệnh trên sẽ thêm vào model 1 statements dạng triples với LABEL là namespace của label, được tạo với câu lệnh

model.add(iri, LABEL, label);

* Tương tự với các thuộc tính khác của các thực thể cũng như quan hệ, ta cũng thêm nó vào 1 model rồi sau đó sẽ add lên server. Việc này sẽ giảm thời gian sinh và add thực thể, quan hệ lên server so với việc add thẳng thực thể lên.

# CHƯƠNG III : BIỂU ĐỒ GÓI VÀ BIỂU ĐỒ LỚP

## I. Biểu đồ gói



### Package entity

* Package entity có chức năng chứa các class model như:
* Entity: là class model, chứa các thuộc tính chung của các thực thể nên class entity sẽ được kế thừa từ các lớp khác (sẽ được thể hiện rõ ở biểu đồ lớp). Class này chứa các thuộc tính như id, label, description, extractedLink, extractedDate cùng với contructor và set, get tương ứng.
* Country: là class model của các thực thể Country, được kế thừa từ lớp Entity, trong class này chứa thuộc tính noCountry là biến static nó giúp ra xác định được id của thực thể Country. Ngoài ra còn có contructor gọi đến contructor class Entity để khởi tạo 1 đối tượng Country.
* Event: là class model của các thực thể Event, được kế thừa từ lớp Entity, trong class này chứa thuộc tính noEvent là biến static nó giúp ra xác định được id của thực thể Event. Ngoài ra còn có contructor gọi đến contructor class Entity để khởi tạo 1 đối tượng Event.
* Location: là class model của các thực thể Location, được kế thừa từ lớp Entity, trong class này chứa thuộc tính noLocation là biến static nó giúp ra xác định được id của thực thể Location. Ngoài ra còn có contructor gọi đến contructor class Entity để khởi tạo 1 đối tượng Location.
* Organization: là class model của các thực thể Organization, được kế thừa từ lớp Entity, trong class này chứa thuộc tính noOrganization là biến static nó giúp ra xác định được id của thực thể Organization và thuộc tính headquater là dung để lưu trữ Trụ Sở của Organization. Ngoài ra còn có contructor gọi đến contructor class Entity để khởi tạo 1 đối tượng Organization và phương thức getHeadquater() và setHeadquater().
* Person: là class model của các thực thể Person, được kế thừa từ lớp Entity, trong class này chứa thuộc tính noPerson là biến static nó giúp ra xác định được id của thực thể Person và thuộc tính age, job là dung để lưu trữ tuổi và công việc của Person. Ngoài ra còn có contructor gọi đến contructor class Entity để khởi tạo 1 đối tượng Person và phương thức getAge(), setAge(), getJob() và setJob().
* Time: là class model của các thực thể Time, được kế thừa từ lớp Entity, trong class này chứa thuộc tính noTime là biến static nó giúp ra xác định được id của thực thể Time. Ngoài ra còn có contructor gọi đến contructor class Entity để khởi tạo 1 đối tượng Time.

### Package entitygeneration

* Package entityGeneration chứa các class như EntityGen, PersonGen, TimeGen, EventGen, LocationGen, OrganizationGen các class này sẽ có các phương thức đọc file lưu vào các biến static và phương thức khởi tạo thực thể tương ứng.
* Ngoài ra còn class RandomEntity được kết tập từ các class trên, chứa các khởi tạo thực thể random các thực thể.

### Package relationship

* Package relationship chứa các lớp Relationship và RelationshipGen, có chức năng chính là chứa class model của relationship và sinh ra relationship.
* Relationship: chứa thuộc tính description có tác dụng lưu trữ mô tả của quan hệ. Ngoài ra, còn có contructor, getDescription(), setDescription().
* RelationshipGen: chứa phương thức đọc file và lưu nó vào biến ArrayList<String>, còn có phương thức lấy random quan hệ từ ArrayList<String> và phương thức khởi tạo relationship.

### Package DatabaseGeneration

* Package DatabaseGenerate chứa class DBGeneration có chức năng kết nối Database, tạo ra các Namespace; IRI; Literal và các phương thức add triples vào biến Model.

### Package InsertData

* Package InsertData chứa class insertData có các thuộc tính như các ArrayList<IRI> là các IRI của entity được add lên Database, đi theo nó là phương thức genRelationship lấy random 2 biến trong ArrayList<IRI> và lấy ngẫu nhiên 1 biến trong ArrayList<String> của relationship để tạo quan hệ rồi add nó lên Server.
* Ngoài ra, class insertData còn phương thức genEnity có chức năng random các đối tượng entity truyền nó vào phương thức InsertEntity để nó add vào model rồi add lên Server.

### Package Query

* Package Query có class Query chức năng chính là truy vấn dữ liệu từ Server, class Query chứ 20 phương thức ứng với 20 câu truy vấn từ cơ bản đến nâng cao. Ngoài ra còn có các phương thức in ra dữ liệu và phương thức làm sạch dữ liệu.

### Package Application

* Package Application chứa lớp App chứa hàm main của chương trình, gọi các phương thức cần thiết của các class khác.

## II. Biểu đồ lớp

### Biểu đồ lớp các thực thể



* Các class như : Person, Orgranization, Location, Country, Event, Time đều kế thừa từ class Entity bởi vì chúng sẽ dung chung các thuộc tính như id, label, extractedLink, extractedDate và các phương thức get(), set() ứng với các thuộc tính đó. Điều này giúp ta giảm thiểu được code và dễ dàng quản lý, chỉnh sửa chương trình hơn.

### Biểu đồ các lớp sinh thực thể

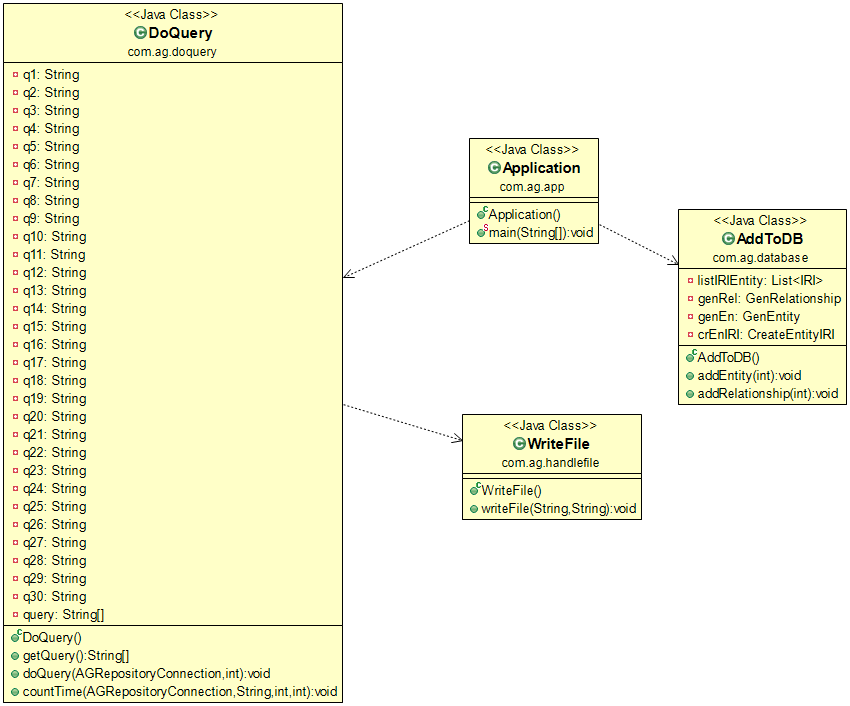


* Tương tự các class PersonGen, OrganizationGen, EventGen, TimeGen, LocationGen, CountryGen kết thừa từ class EntityGen.
* Class RandamEntityGen được kết tập từ các class trên.

### Biểu đồ lớp add thực thể lên Server



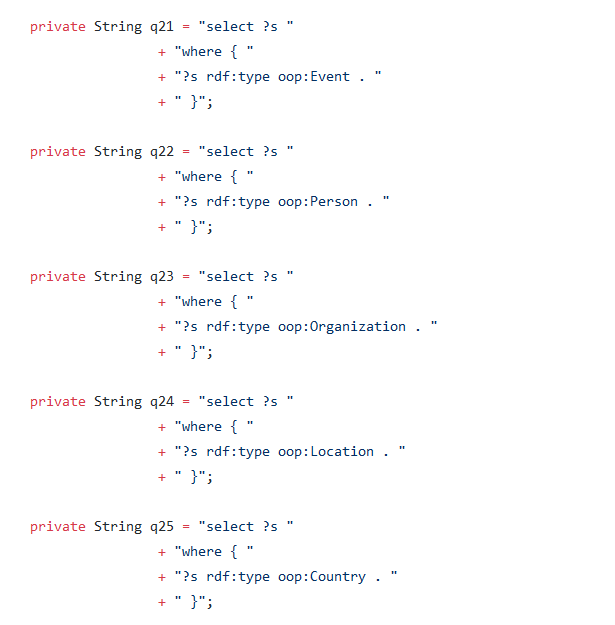
### Biểu đồ lớp chạy hàm main

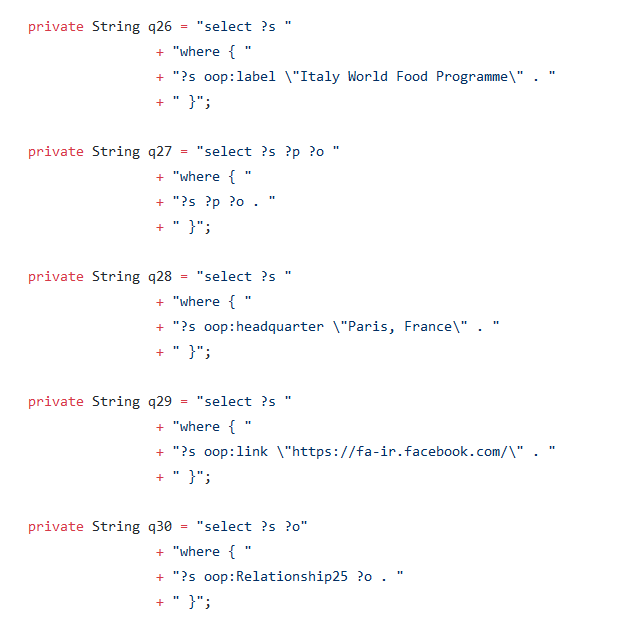


# CHƯƠNG IV : KẾT QUẢ KIỂM THỬ HIỆU NĂNG

## Các câu truy vấn

### Truy vấn cơ bản





### Truy vấn nâng cao

* Query 1:

"select ?description

where {

?s oop:description ?description .

?s oop:label ?name .

?s rdf:type oop:Person .

FILTER(regex(str(?name), "Cherryl Confer", "i"))

}"

* Lấy description của thực thể Person có tên là Cherryl Confer
* Query 2:

"select ?label

where {

?s oop:label ?label .

?s ?rel ?o .

?s rdf:type oop:Person .

?o oop:label "Food and Agriculture Organization" .

?o rdf:type oop:Organization .

?rel oop:name ?name .

FILTER(regex(str(?name), "manage", "i"))

}"

* Lấy nhãn của thực thể Person mà có quan hệ manage với thực thể Organization có nhãn Food and Agriculture Organization
* Query 3:

"select ?link

where {

?s oop:link ?link .

?s oop:label ?name .

?s rdf:type oop:Event .

FILTER(regex(str(?name), "Open Doors on Astronomy 2018", "i"))

}”

* Lấy link (nguồn trích rút) của thực thể Event có nhãn Open Doors on Astronomy 2018
* Query 4:

"select ?description

where {

?s oop:description ?description .

?s oop:label ?name .

?s rdf:type oop:Location .

FILTER(regex(str(?name), "European Fecal Standards and Measurements", "i"))

}"

* Lấy description của thực thể Location có nhãn European Fecal Standards and Measurements
* Query 5:

"select ?headquarter

where {

?s oop:headquarter ?headquarter .

?s oop:label ?name .

?s rdf:type oop:Organization .

FILTER(regex(str(?name), "International Organization for Migration (IOM)", "i"))

}"

* Lấy headquarter (trụ sở) của thực thể Organization có nhãn International Organization for Migration (IOM)
* Query 6:

"select ?label

where {

?o ?rel ?s .

?o rdf:type oop:Country .

?o oop:label ?label .

?s oop:label "Space Trip" .

?s rdf:type oop:Event .

?rel oop:name ?name .

FILTER(regex(str(?name), "held", "i"))

}"

* Lấy nhãn của thực thể Country có quan hệ held với thực thể Event nhãn Space Trip
* Query 7:

"select ?label

where {

?o ?rel ?s .

?o rdf:type oop:Country .

?s oop:label "Renae Ratledge" .

?s oop:position "Dentists" .

?s rdf:type oop:Person .

?o oop:label ?label .

?rel oop:name ?name .

FILTER(regex(str(?name), "have", "i"))

}"

* Lấy nhãn của thực thể Country có quan hệ have với thực thể Person nhãn Renae Ratledge và làm nghề (position) Dentists
* Query 8:

"select ?link

where {

?s oop:link ?link .

?s rdf:type oop:Location .

?s oop:time\_extracted "Nov. 10, 1990" .

}"

* Lấy link (nguồn trích rút) của thực thể Location có thời gian trích rút là Nov. 10, 1990
* Query 9:

"select ?headquarter

where {

?s rdf:type oop:Organization .

?s oop:label "Belgium–Luxembourg Economic Union" .

?s oop:headquarter ?headquarter .

}"

* Lấy trụ sở của thực thể Organization có nhãn Belgium–Luxembourg Economic Union
* Query 10:

"select distinct ?label

where {

?s ?rel ?o .

?rel oop:name ?name .

?s rdf:type oop:Country .

?s oop:label ?label .

?o rdf:type oop:Organization .

?o oop:label "World Trade Organization (WTO)" .

FILTER(regex(str(?name), "have", "i"))

}"

* Lấy nhãn của thực thể Country có quan hệ have với thực thể Organization có nhãn World Trade Organization (WTO)
* Query 11:

"select distinct ?label

where {

?s ?rel ?o .

?rel oop:name ?name .

?s oop:label ?label .

?s rdf:type oop:Event .

?o oop:label "South Park Church" .

?o rdf:type oop:Location .

FILTER(regex(str(?name), "held in", "i"))

}"

* Lấy nhãn của các thực thể Event có quan hệ held in với thực thể Location nhãn South Park Church
* Query 12:

"select distinct ?label

where {

?s ?rel ?o .

?o oop:label ?label .

?o rdf:type oop:Country .

?s rdf:type oop:Person .

?s oop:label "Genaro Kopacz" .

?rel oop:name ?name .

?rel oop:time ?time .

FILTER(regex(str(?time), "2000") || regex(str(?name), "visit", "i"))

}"

* Lấy nhãn của các thực thể Country mà thực thể Person nhãn Genaro Kopacz tới (quan hệ visit) vào năm 2000 (thời gian xảy ra quan hệ)
* Query 13:

"select distinct ?label

where {

?s ?rel ?o .

?s rdf:type oop:Organization .

?s oop:label "Council of Europe (CoE)" .

?o rdf:type oop:Event .

?o oop:label ?label .

?rel oop:name ?name .

?rel oop:time ?time .

FILTER(regex(str(?time), "2009") || regex(str(?name), "held", "i"))

}"

* Lấy nhãn những Event mà tổ chức Council of Europe (CoE) tổ chức (held) năm 2009
* Query 14:

"select distinct ?label

where {

?s ?rel ?o .

?s rdf:type oop:Event .

?s oop:label ?label .

?o rdf:type oop:Country .

?o oop:label "Finland" .

?rel oop:name ?name .

?rel oop:time ?time .

FILTER(regex(str(?time), "2010") || regex(str(?name), "held in", "i"))

}"

* Lấy nhãn của các Event có quan hệ held in (được tổ chức) ở Country Finland vào năm 2010
* Query 15:

"select distinct ?label

where {

?s ?rel ?o .

?s oop:label ?label .

?s rdf:type oop:Organization .

?o oop:label "Nordic Council" .

?o rdf:type oop:Organization .

?rel oop:name ?name .

FILTER(regex(str(?name), "in the same country", "i"))

}"

* Lấy các nhãn của Organization mà có quan hệ in the same country (cùng nước) với Organization nhãn Nordic Council
* Query 16:

"select distinct ?label

where {

?s ?rel ?o .

?s rdf:type oop:Person .

?s oop:label ?label .

?o rdf:type oop:Organization .

?o oop:label "European Science Foundation" .

?rel oop:name ?name .

FILTER(regex(str(?name), "work in", "i"))

}"

* Lấy nhãn của các Person work in Organization European Science Foundation
* Query 17:

"select distinct ?label

where {

?s ?rel ?o .

?s rdf:type oop:Person .

?s oop:label ?label .

?s oop:position "Fashion designers" .

?o rdf:type oop:Country .

?o oop:label "Latvia" .

?rel oop:name ?name .

FILTER(regex(str(?name), "live in", "i"))

}"

* Lấy nhãn của các Person làm Fashion designers live in Country Latvia
* Query 18:

"select distinct ?label

where {

?s ?rel ?o .

?s rdf:type oop:Location .

?s oop:label ?label .

?o oop:label "SlaterCarey Mental Hospital" .

?o rdf:type oop:Location .

?rel oop:name ?name .

FILTER(regex(str(?name), "in same country with", "i"))

}"

* Lấy nhãn của các Location mà in same country with với Location nhãn SlaterCarey Mental Hospital
* Query 19:

"select distinct ?headquarter

where {

?s ?rel ?o .

?s oop:headquarter ?headquarter .

?s rdf:type oop:Organization .

?o oop:label "Italy World Food Programme” .

?o rdf:type oop:Organization .

?rel oop:name ?name .

FILTER(regex(str(?name), "in the same country", "i"))

}"

* Lấy headquarter của các Organization in the same country với Organization Italy World Food Programme
* Query 20:

"select distinct ?label

where {

?s ?rel ?o .

?s rdf:type oop:Event .

?s oop:label ?label .

?o oop:label "Prociencia en la Semana Mundial del Espacio" .

?o rdf:type oop:Event .

?rel oop:name ?name .

?rel oop:time ?time .

FILTER(regex(str(?name), "is held by same organization", "i") || regex(str(?time), "2005"))"

}"

* Lấy nhãn các Event mà is held by same organization với Event nhãn Prociencia en la Semana Mundial del Espacio

## Thời gian truy vấn

Do Allegro Graph chỉ cho phép lưu trữ tối đa 5 triệu triples nên ở đây nhóm sẽ chỉ sinh dữ liệu với độ lớn tối đa là 400.000 thực thể và 500.000 quan hệ (do đã ước tính mỗi thực thể chiếm 5-6 triples, mỗi quan hệ chiếm 5 triples).

Vì thời gian được tính bằng code Java nên kết quả có thể không được chính xác hoàn toàn và lâu hơn thời gian chính xác của nó (phụ thuộc cả vào máy).

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (N,M) | CB1 | CB12 | CB3 | CB4 | CB5 | CB6 | CB7 | CB8 | CB9 | CB10 |
| (5k,6.25k) | 4361 | 1556 | 630 | 1124 | 1873 | 48 | 1767 | 1388 | 735 | 291 |
| (30k,3.75k) | 14210 | 6904 | 5159 | 4093 | 2591 | 44 | 1954 | 1884 | 8429 | 1030 |
| (100k, 125k) | 17968 | 26941 | 6540 | 11662 | 5081 | 779 | 3845 | 9095 | 16267 | 907 |
| (400k, 500k) | 61018 | 26146 | 5613 | 49026 | 1725 | 706 | 51879 | 93160 | 81894 | 10342 |

*Đơn vị: ms*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (N,M) | NC1 | NC2 | NC3 | NC4 | NC5 | NC6 | NC7 | NC8 | NC9 | NC10 |
| (5k, 6.25k) | 18478 | 235 | 4154 | 1576 | 3782 | 10173 | 34 | 1784 | 474 | 1240 |
| (30k,3.75k) | 16357 | 352 | 2280 | 1162 | 2030 | 21020 | 27 | 7601 | 1633 | 9136 |
| (100k, 125k) | 10369 | 63 | 35662 | 2059 | 2153 | 83817 | 800 | 18893 | 11450 | 45763 |
| (400k, 500k) | 112874 | 1196 | 95337 | 9983 | 1622 | 131557 | 3031 | 24443 | 68838 | 138346 |

*Đơn vị: ms*

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (N,M) | NC11 | NC12 | NC13 | NC14 | NC15 | NC16 | NC17 | NC18 | NC19 | NC20 |
| (5k,6.25k) | 2936 | 106 | 6028 | 2770 | 5190 | 2845 | 2183 | 1885 | 2871 | 1858 |
| (30k,3.75k) | 10807 | 74 | 6567 | 8357 | 10381 | 7927 | 2371 | 6277 | 7142 | 7556 |
| (100k, 125k) | 44029 | 173 | 27640 | 26236 | 41122 | 36544 | 7711 | 36661 | 39394 | 36240 |
| (400k, 500k) | 226587 | 3209 | 72776 | 149521 | 77688 | 155338 | 98437 | 151692 | 82912 | 121207 |

*Đơn vị: ms*

## Kết luận

### Ưu điểm

* Cách tương tác với API, cách cài đặt AGServer khá dễ dàng, dễ hiểu.
* Giao diện server (AGWebView: <http://localhost:10035>) trực quan, dễ sử dụng.
* AGWebView có tích hợp cho việc truy vấn trực tiếp trên đây. Thời gian truy vấn nhanh.
* Hỗ trợ nhiều ngôn ngữ như Java, Python, Scala, Ruby, Perl, C#...

### Nhược điểm

* Khi import file lên server thì tốc độ khá nhanh nhưng khi sử dụng java để add các triples lên server thì tốc độ lại chậm.
* Bản free license chỉ lưu trữ được tối đa 5 triệu triples.