**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG**

──────── \* ───────

**BÁO CÁO OOP**

**ĐỀ TÀI*: Tìm hiểu về công nghệ AllegroGraph***

**Giáo viên hướng dẫn: TS. TRỊNH TUẤN ĐẠT**

***Lớp 104403***

***Nhóm 5***

***Các thành viên:***

|  |  |  |  |
| --- | --- | --- | --- |
| **STT** | **MSSV** | **Họ tên** | **Đóng góp** |
| **1** | **20162569** | **Nguyễn Tử Toàn Lợi** | **41%** |
| **2** | **20162679** | **Vũ Duy Mạnh** | **27%** |
| **3** | **20163703** | **Lê Thị Thành** | **12%** |
| **4** | **20162974** | **Vũ Bích Ngọc** | **11%** |
| **5** | **20162859** | **Trần Văn Nhật Nam** | **9%** |

***Hà Nội, ngày 12 tháng 10, năm 2018***

|  |  |  |
| --- | --- | --- |
| Họ và tên | Đóng góp | Nhiệm vụ |
| **Nguyễn Tử Toàn Lợi** | 41% | Thiết kế package và các class, code sinh thực thể, quan hệ, tạo IRI, code đẩy dữ liệu vào CSDL qua API, báo cáo chương 3, thiết kế giải thuật sinh dữ liệu. |
| Vũ Duy Mạnh | 27% | Tìm hiểu công nghệ, tạo các file dữ liệu, thiết kế truy vấn dữ liệu, đo thời gian truy vấn, code giao diện cho chương trình, tổng hợp báo cáo. |
| Lê Thị Thành | 12% | Tìm hiểu công nghệ, báo cáo tìm hiểu công nghệ , refractor code, rút ra ưu nhược điểm của công nghệ. |
| Vũ Bích Ngọc | 11% | Tìm hiểu công nghệ, báo cáo giải thuật sinh dữ liệu, refractor code |
| Trần Văn Nhật Nam | 9% | Tìm hiểu công nghệ, báo cáo kết quả truy vấn, refractor code |

Mục Lục

[Chương 1: Tìm hiểu về công nghệ 5](#_Toc533531485)

[1. Giới thiệu 5](#_Toc533531486)

[2. Các thuật ngữ 5](#_Toc533531487)

[3. Hướng dẫn cài đặt chi tiết 6](#_Toc533531488)

[3.1. Cài đặt Server AllegroGraph 6](#_Toc533531489)

[3.2. Cài đặt Java Client 7](#_Toc533531490)

[4. Các kỹ thuật sử dụng 7](#_Toc533531491)

[4.1. Đăng nhập vào server 8](#_Toc533531492)

[4.2. Tạo repository, truy cập repository 8](#_Toc533531493)

[4.3. Add, delete, import triple 8](#_Toc533531497)

[4.4. Export file 9](#_Toc533531498)

[4.5. View triple 9](#_Toc533531499)

[5. Cách truy vấn 9](#_Toc533531500)

[5.1. Ngôn ngữ truy vấn 9](#_Toc533531501)

[5.2. Cách truy vấn trực tiếp trên Server 11](#_Toc533531502)

[6. Mô hình dữ liệu 14](#_Toc533531503)

[7. Cách giao tiếp bằng API 14](#_Toc533531504)

[7.1. Cách giao tiếp với API 14](#_Toc533531505)

[7.2. Cụ thể sử dụng API trong chương trình với các mục đích: 15](#_Toc533531506)

[Chương 2. Trình bày về mô hình dữ liệu 18](#_Toc533531507)

[1. Lưu trữ dữ liệu trên server. 18](#_Toc533531508)

[2. Lưu trữ dữ liệu trong project 18](#_Toc533531509)

[3. Giải thuật sinh dữ liệu 19](#_Toc533531510)

[a. Sinh thực thể 19](#_Toc533531511)

[b. Sinh quan hệ 22](#_Toc533531512)

[Chương 3: Trình bày về chương trình của nhóm 23](#_Toc533531513)

[1. Package và các lớp 23](#_Toc533531514)

[1.1. Package entity 23](#_Toc533531515)

[1.2. Package filereader 25](#_Toc533531516)

[1.3. Package query 26](#_Toc533531517)

[1.4. Package createdata 28](#_Toc533531518)

[1.5. Package time 32](#_Toc533531519)

[1.6. Package connection 32](#_Toc533531520)

[1.7. Package setting 34](#_Toc533531521)

[1.8. Package myinterface 34](#_Toc533531522)

[2. Sơ đồ biểu thị sự phụ thuộc giữa các package 36](#_Toc533531523)

[3. Sơ đồ lớp 36](#_Toc533531524)

[Chương 4 : Kiểm thử hiệu năng 39](#_Toc533531525)

[1. Các truy vấn cơ bản 39](#_Toc533531526)

[1.1. Truy vấn 1 39](#_Toc533531527)

[1.2. Truy vấn 2 39](#_Toc533531528)

[1.3. Truy vấn 3 39](#_Toc533531529)

[1.4. Truy vấn 4 40](#_Toc533531530)

[1.5. Truy vấn 5 40](#_Toc533531531)

[1.6. Truy vấn 6 41](#_Toc533531532)

[1.7. Truy vấn 7 41](#_Toc533531533)

[1.8. Truy vấn 8 41](#_Toc533531534)

[1.9. Truy vấn 9 42](#_Toc533531535)

[1.10. Truy vấn 10 42](#_Toc533531536)

[2. Truy vấn nâng cao 43](#_Toc533531537)

[2.1. Truy vấn 1 43](#_Toc533531540)

[2.2. Truy vấn 2 43](#_Toc533531543)

[2.3. Truy vấn 3 44](#_Toc533531544)

[2.4. Truy vấn 4 44](#_Toc533531545)

[2.5. Truy vấn 5 45](#_Toc533531546)

[2.6. Truy vấn 6 45](#_Toc533531547)

[2.7. Truy vấn 7 46](#_Toc533531548)

[2.8. Truy vấn 8 47](#_Toc533531549)

[2.9. Truy vấn 9 47](#_Toc533531550)

[2.10. Truy vấn 10 48](#_Toc533531551)

[3. Kiểm thử hiệu năng 49](#_Toc533531552)

[3.1. Truy vấn cơ bản 49](#_Toc533531553)

[3.2. Truy vấn nâng cao 49](#_Toc533531554)

[4. Đánh giá công nghệ 49](#_Toc533531555)

[4.1. Ưu điểm 49](#_Toc533531556)

[4.2. Nhược điểm 49](#_Toc533531557)

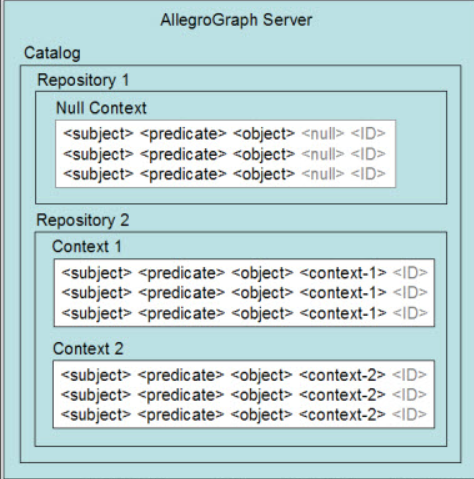
# Chương 1: Tìm hiểu về công nghệ

## Giới thiệu

* AllegroGraph là 1 Cơ sở dữ liệu, và là một app framework cho việc xây dựng Semantic Web Application.
* Nó có thể lưu trữ data và meta-data dưới dạng triple, và truy vấn những triple này thông qua các API truy vấn khác nhau như SPARQL (ngôn ngữ truy vấn chuẩn của W3C) và Prolog.
* Java Sesame API cung cấp truy cập thuận tiện và hiệu quả tới một máy chủ AllegroGraph từ một ứng dụng dựa trên Java.
* API này cung cấp các phương thức để tạo, truy vấn và duy trì dữ liệu RDF và cho việc quản lí các triple được lưu trữ.
* Java Sesame API mô phỏng API RDF4J của Eclipse để giúp việc migrate từ RDF4J sang AllegroGraph dễ dàng hơn.

## Các thuật ngữ

* RDF : (Resource Description Framework) là khung mô tả tài nguyên được tạo ra bởi 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 RDF graph như vậy.
* URI : (Uniform Resource Identifier) là một label được sử dụng để xác định duy nhất các loại thực thể khác nhau trong RDF graph. Một URI điển hình trông rất giống một địa chỉ Web : [http://example.org/project/class#number](http://example.org/project/class). Mặc dù giống nhau nhưng URI không phải là một địa chỉ Web . Nó đơn giản chỉ là một nhãn duy nhất.
* Triple : là một định dạng dữ liệu, một thực tế, được lưu trữ trong định dạng RDF. Nó cho biết một tài nguyên có một thuộc tính với một giá trị. Triple gồm có 3 trường:
  + *Subject*: trường đầu tiên chứa URI xác định duy nhất tài nguyên mà triple mô tả.
  + *Predicate*: trường thứ hai chứa URI xác định một thuộc tính của tài nguyên này, chẳng hạn như màu, hoặc kích thước hoặc mối quan hệ giữa tài nguyên này và tài nguyên khác.
  + *Object*: trường thứ ba là giá trị của thuộc tính. Nó có thể là một giá trị theo nghĩa đen, chẳng hạn như “red” hoặc URI của tài nguyên được liên kết.
* Quad: là một triple và được thêm một trường “context”, được sử dụng để chia repository thành các subgraphs. Context hoặc subgraph này chỉ là một nhãn URI xuất hiện trong trường thứ 4 của các triple liên quan.
* Quint: là một quad và được thêm một trường thứ 5: “triple ID”. AllegroGraph implement tất cả các triple như là một quint. Trường thứ 4 và thứ 5 thường bị ignore.
* Resource description: được định nghĩa như một tập hợp của các triple mà có cùng URI ở trường subject. Nói cách khác, các triples này đều mô tả thuộc tính của cùng một thứ .
* Statement: là một đối tượng Java phía client dùng để mô tả triple (quad, quint) .
* Trong 1 context của AllegroGraph Server



* Catalog: là danh sách các repositories thuộc sở hữu của máy chủ AllegroGraph.
* Repository: là một tập hợp các triple trong catalog, được lưu trữ và chỉ mục trên đĩa cứng.
* Nếu context không được sử dụng, triple được lưu trữ trong backgroud graph (mặc định)

## Hướng dẫn cài đặt chi tiết

### Cài đặt Server AllegroGraph

* Có 2 phương pháp cài đặt cho máy chủ AllegroGraph, một cho quản trị viên và một cho những người không phải quản trị viên. Quản trị viên có thể cài đặt RPM file và những người không phải quản trị viên có thể giải nén file TAR.GZ và cài đặt ở vị trí họ chọn.
* Cài đặt bằng file TAR.GZ (cho tất cả các hệ thống)
  + Không cần phải chạy dưới quyền root
  + Download agraph-6.4.5-linuxamd64.tar.gz từ trang <https://franz.com/agraph/downloads/> . Đặt nó trong thư mục rồi giải nén file cài đặt
    - tar zxf agraph-6.4.5-linuxamd64.64.tar.gz
  + Lúc này thư mục con agraph-6.4.5 sẽ được tạo, thư mục này chứa tập lệnh cài đặt ở bên trong.
  + Start AllegroGraph server :
    - /home/<**user>**/tmp/ag6.4.5/bin/agraph-control –config /home/<**user>**/tmp/ag6.4.5/lib/agraph.cfg start
  + Stop AllegroGraph server :
    - /home/<**user>**/tmp/ag6.4.5/bin/agraph-control –config /home/<**user>**/tmp/ag6.4.5/lib/agraph.cfg stop
  + Để kiểm tra xem AllegroGraph có đang chạy hay không, mở trình duyệt lên và gõ:
    - <http://localhost:10035>
  + Allegro chạy mặc định ở cổng 10035. Cần phải username và password để đăng nhập. Sau khi đăng nhập sẽ mở ra giao diện WebView , giao diện này cho phép quản lý người dùng và tương tác, kiểm tra dữ liệu trong kho lưu trữ dữ liệu (repositories).
  + Sử dụng bản free chỉ cho phép tối đa 5 triệu triple .

### Cài đặt Java Client

* Điều hướng đến [https://bit.ly/2V8cgQY](https://bit.ly/2V8cgQY%20) để xem hướng dẫn cài đặt chi tiết.
* Tải các bản phân phối Tar của AllegroGraph tại trang <https://bit.ly/2AcSydS>
* File Client AllegoGraph nằm trong thư mục agraph-java-client-6.4.5.tar.gz.
* Để sử dụng AllegroGraph Client, cài đặt nó trong kho lưu trữ của Maven bằng cách thực hiện lệnh sau:
  + Mvn install:install-file -Dfile=lib/agraph-java-client-\*.jar -DpomFile=pom.xml
* Chỉnh sửa tệp cấu hình agraph.cfg ở trong thư mục lib/ cho bản cài đặt TAR. Sửa lại số cổng là 10035

## Các kỹ thuật sử dụng

Sau khi đăng nhập vào server, ta có thể sử dụng các dịch vụ mà server cung cấp như : tạo mới repository, truy cập repository. Đối với mỗi repository, ta có thể import triple, export triple, add triple, delete triple, view triple và các nhiều tùy chọn nâng cao khác.

### Đăng nhập vào server

- Vào giao diện của server bằng cách điều hướng đến localhost của server mà ta đã nhập trong phần cài đặt. Mặc định là cổng 10035. Sau khi nhập user và password, ta vào giao diện chính của server như sau.



- Ta thấy trong màn hình chính của server có 2 mục là repository và catalogs, tuy nhiên trong bài tập lớn, ta chỉ quan tâm đến phần repository.

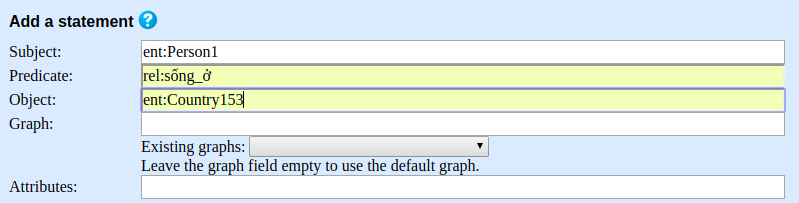
### Tạo repository, truy cập repository

- Sau khi đăng nhập vào, ta sẽ thấy mục create new repository. Chỉ cần nhập tên repository muốn tạo và nhấn crate. Khi đó, 1 repository mới sẽ được tạo trong list repository có sẵn, chỉ cần click vào repository đó là sẽ truy cập được vào giao diện của repository.



### Add, delete, import triple

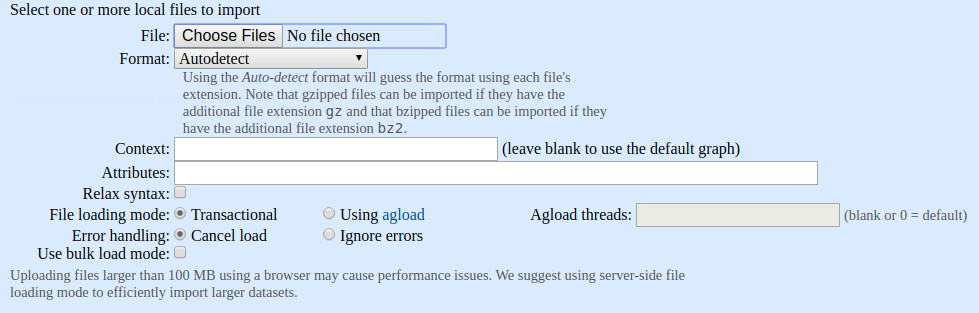
- Ở trong giao diện của từng repository, trong mục **Load and Delete Data**, ta chọn add a statements để thêm 1 triple



- Thêm các thông tin của triple vào (bao gồm cả prefix : ent và rel trong hình là các namespace đã được đặt sẵn, phần graph là đồ thị con, trong bài tập lớn ta không sử dụng đến nên sẽ không điền). Sau đó bấm vào button "OK". Server sẽ hiện lên thông báo trong cả 2 trường hợp add triple thành công và không thành công.

- Chọn delete statements để xóa 1 triple. Tương tự như add triple, ta sẽ nhập các thông tin của triple cần xóa (bao gồm cả prefix) và bấm vào button "OK". Server sẽ hiện lên thông báo trong cả 2 trường hợp delete triple thành công và không thành công.

- Trong phần Import RDF có 3 tùy chọn để import triple. Tuy nhiên cách thường sử dụng là import an uploaded file. Nhấn vào dòng import an uploaded file, server sẽ hiện lên bảng để import file.



- Bấm vào Choose Files để chọn file lưu triple có sẵn. Phần Format để chọn loại file được thêm, ví dụ như XML hoặc TriG. Ta sẽ để là tùy chọn Autodetect để server tự nhận dạng file import. Bên dưới là các tùy chọn nâng cao tùy vào mục đích của sử dụng mà ta sẽ quyết định chọn hay không chọn. Sau đó bấm botton "OK" để import file.

### Export file

- Ta có thể sử dụng chức năng export file để lấy file chứa các triple hiện có trong repository. Ở phần **Repository Control**, ta chọn export file, đồng thời chọn định dạng file muốn export. Ta sẽ được 1 file chứa các triple đã có trong repository. File này có thể dùng để import vào các server khác.

### View triple

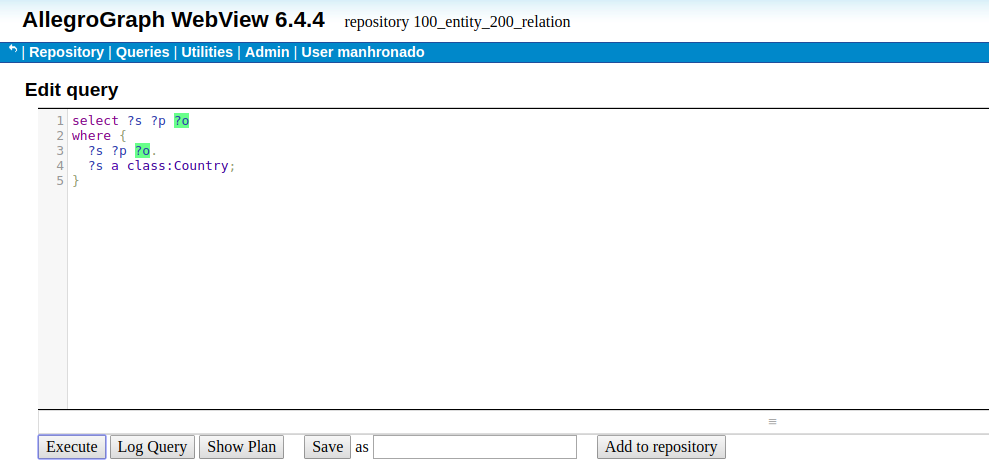
- Trong phần **Explore the Repository**, ta chọn mục view triples, ta có thể thấy được tất cả các triple hiện có trong repository.

Ngoài ra còn nhiều cách khác để thao tác với repository, tuy nhiên trong bài tập lớn không sử dụng nên bọn em xin phép không đi vào chi tiết cụ thể.

## Cách truy vấn

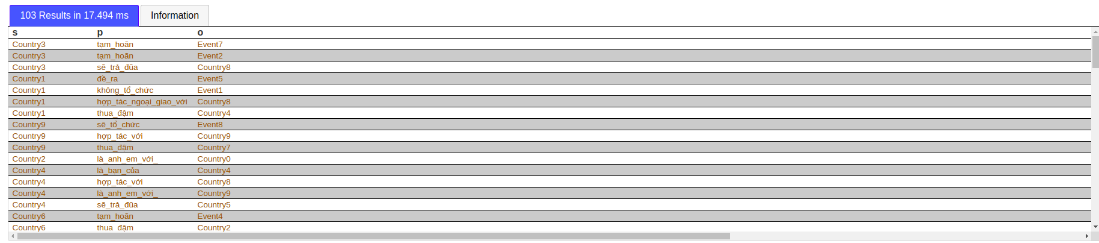
### Ngôn ngữ truy vấn

* + SPARQL
  + SPARQL là ngôn ngữ thực hiện các truy vấn phức tạp trên dữ liệu RDF phân tán
  + Ví dụ 1: ta có truy vấn sau:

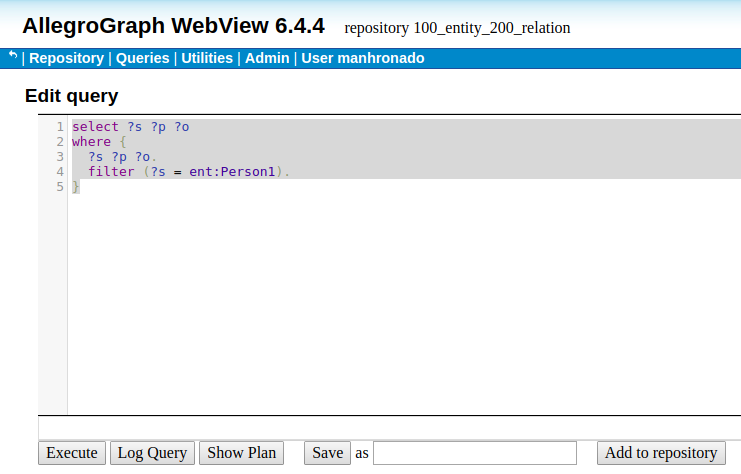


Trong đó ?s ?p ?o là biểu thị chọn tất cả các truy vấn dạng "s p o". (?s a class:Country) là viết tắt của (?s rdf:type class:Country), có ý nghĩa là chọn tất cả các biến ?s thuộc class Country. do đó câu truy vấn có ý nghĩa là lấy thông tin của tất cả những triple trong đó ?s là một thực thể country.

Kết quả:

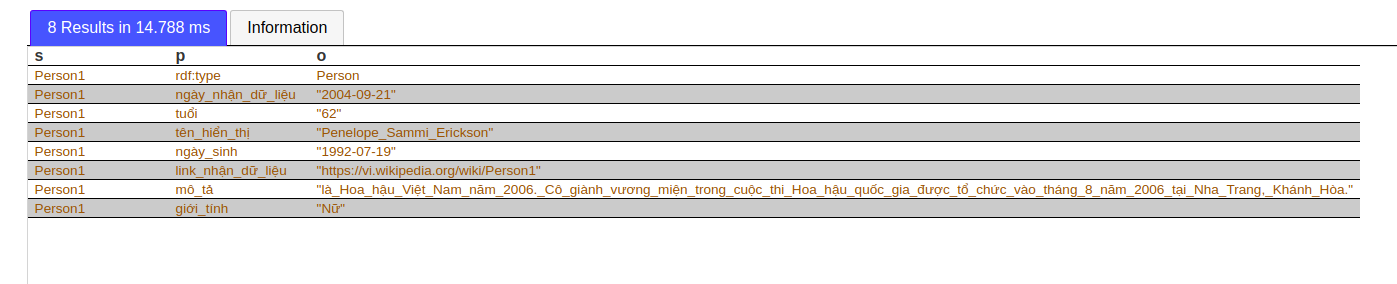


* + Ví dụ 2:



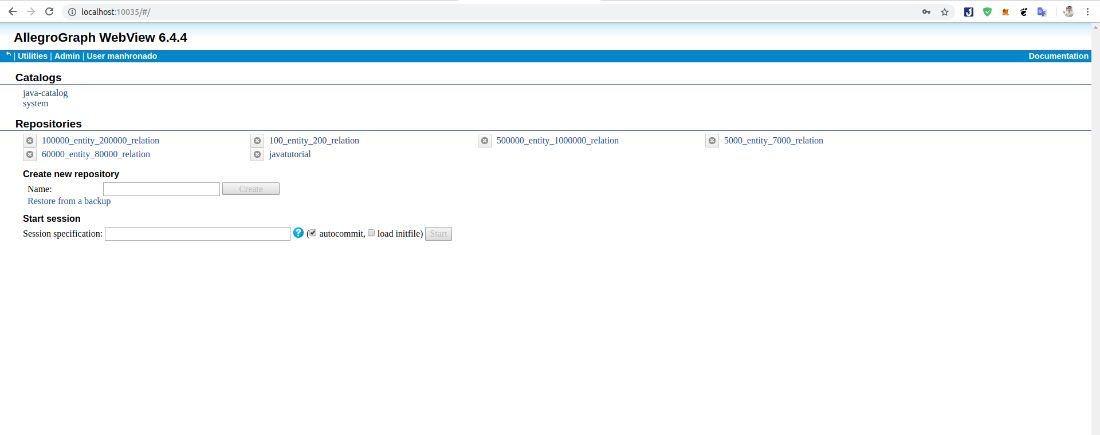
Câu truy vấn này lấy tất cả thông tin của thực thể Person1. filter ở đây là lọc những triple thỏa mãn phần bên trong ngoặc của filter .

Kết quả:

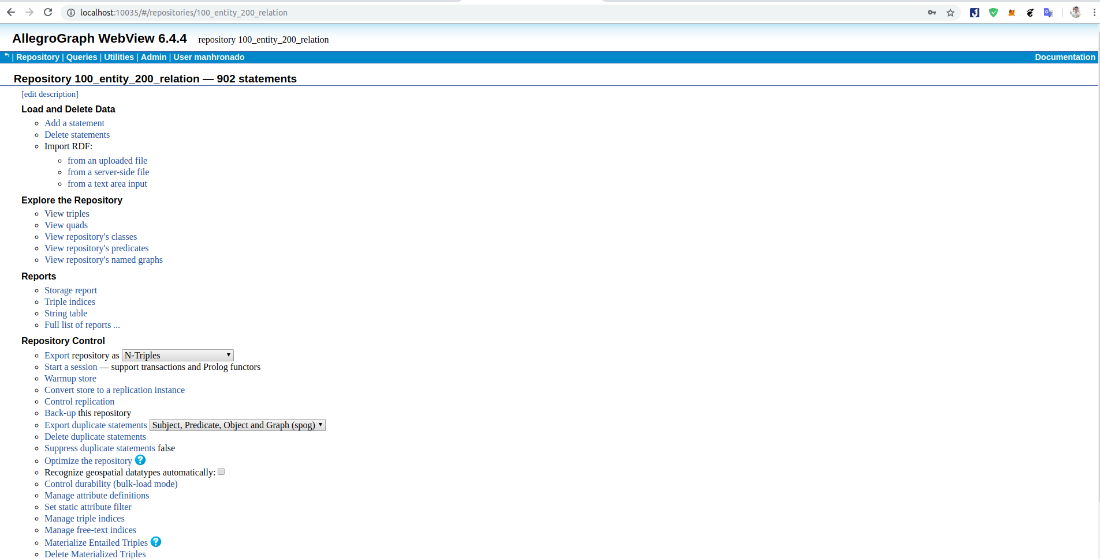


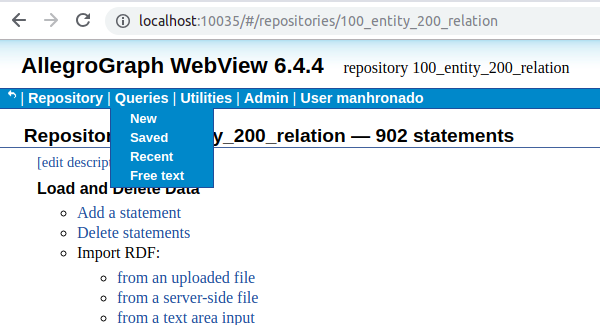
### Cách truy vấn trực tiếp trên Server

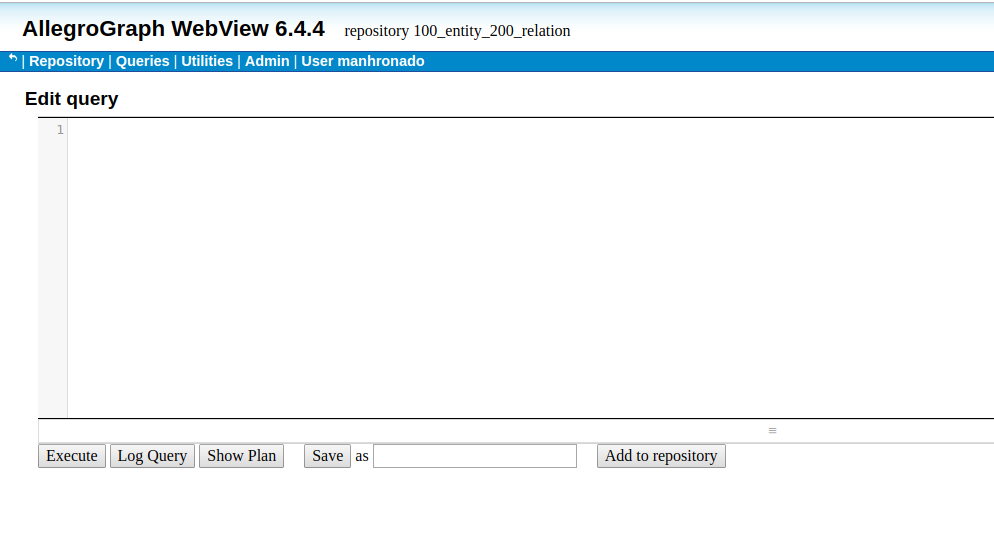
B1: Truy cập vào đường dẫn localhost:10035. Tai đây, sẽ xuất hiện các repositories như trên hình



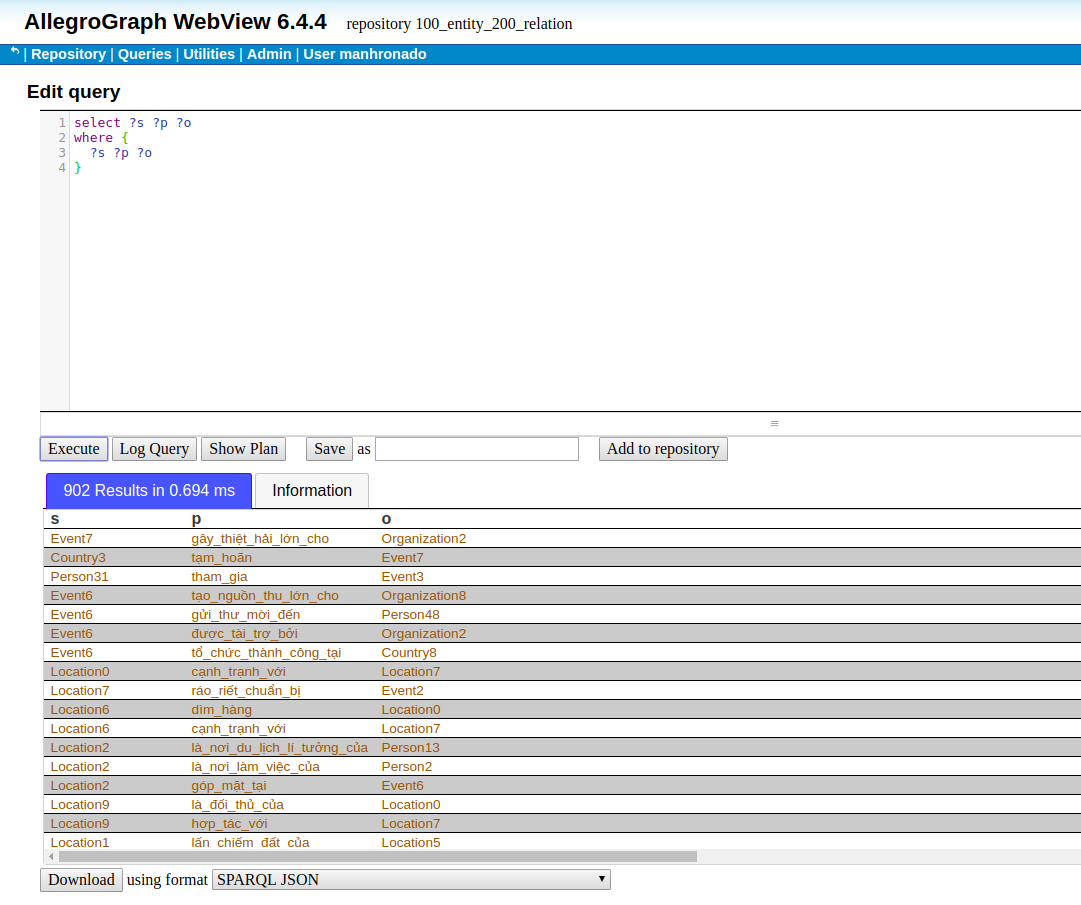
B2: Chọn repository mà bạn muốn truy vấn. Sau đó trên màn hình sẽ hiển thị ra như sau:



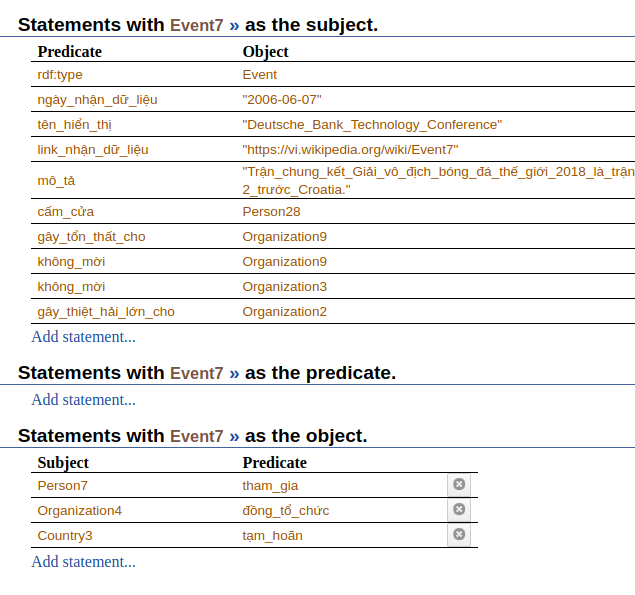
B3: Chọn Queries/New để mở ra khung truy vấn. 

Trên màn hình sẽ hiển thị ra khung truy vấn như sau: 

B4: Tiến hành nhập câu lệnh truy vấn. Ở đây ví dụ nhập câu lệnh hiển thị tất cả các triple có trong repository. Sau đó chọn excute để xem kết quả. Ta sẽ được kết quả như sau:



Ta có thể click vào các mục kết quả để xem chi tiết các mục đó. Ví dụ ta bấm vào phần event 7 thì. sẽ hiển thị tất cả thông tin của event 7 như hình sau:



## Mô hình dữ liệu

- Allegro Graph lưu trữ dữ liệu dưới dạng các triple như đã trình bày ở phần giới thiệu. Các triple có dạng **s p o**, tượng trưng cho subject predicate object. Các triple bao gồm các URI hoặc các giá trị Literal. Ngoài ra còn 2 giá trị nữa là **g i**, tượng trưng cho graph và triple-id. Trong đó g lưu tên của sub-graph, nếu giá trị g không được tạo thì triple thuộc đồ thị mặc định. i lưu id của triple. 2 giá trị g và i được thêm vào nhằm làm tăng tốc độ truy vấn khi làm việc với các bộ dữ liệu lớn.

- Mỗi URI có các prefix đứng trước nó. Allegro Graph có thể hiểu được một số prefix thông dụng như foaf: , err: , rdf: . Ngoài ra ta có thể tùy ý thêm các namespace do ta tự định nghĩa.

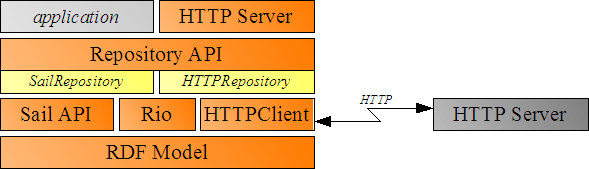
- Các giá trị Literal có thể là các giá trị String, int, float, boolean, datetime ... Các Literal thường là giá trị mang ý nghĩa dùng để hiển thị, lưu trữ các giá trị thuộc tính của một thực thể nào đó.

- Allegro Graph có thể lưu trữ dữ liệu dưới dạng đồ thị con, dùng để tăng tốc độ truy vấn khi tìm kiếm với dữ liệu lớn. Tuy nhiên trong bài tập lớn không sử dụng đến đồ thị con nên chúng em xin phép không đề cập đến trong bài báo cáo.

## Cách giao tiếp bằng API

### Cách giao tiếp với API

* AllegroGraph sử dụng RDF4J Framework - một framework dùng cho việc lưu trữ, truy vấn dữ liệu RDF
* RDF4J có thể được sử dụng theo 2 cách sau:
  + Như một server độc lập
  + Được nhúng vào trong ứng dụng như là một thư viện Java
* RDF4J hỗ trợ ngôn ngữ truy vấn SPARQL
* Cấu trúc của RDF4J



* Ứng dụng sẽ giao tiếp với RDF4J thông qua Repository API

### Cụ thể sử dụng API trong chương trình với các mục đích:

* Kết nối tới cơ sở dữ liệu

+ Các thư viện cần có:

com.franz.agraph.repository.AGCatalog

com.franz.agraph.repository.AGRepository

com.franz.agraph.repository.AGRepositoryConnection com.franz.agraph.repository.AGServer

+ Các bước kết nối:

1. Kết nối với server theo server, repository, user, password cho trước.Nếu chưa tồn tại, repository sẽ tự động tạo repository mới
2. Tạo 1 kết nối với repository mới tương ứng với repository vừa nhập vào ( cùng ID ). Nếu repository chưa tồn tại, tự động tạo ra 1 repository có tên tương ứng và trả về kết nối
3. Đặt mới lại repository, dùng để tạo nhiều repository tương ứng với nhiều bộ truy vấn. Nếu repository đã tồn tại, xóa đi và ghi đè.
4. Thoát khỏi phiên làm việc, đóng repository

* Đưa dữ liệu lên cơ sở dữ liệu

+ Các thư viện cần có:

org.eclipse.rdf4j.model.Model

org.eclipse.rdf4j.model.impl.TreeModel

org.eclipse.rdf4j.model.ValueFactory

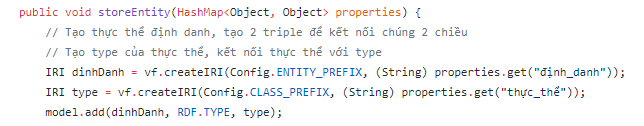
org.eclipse.rdf4j.model.IRI

org.eclipse.rdf4j.model.Literal

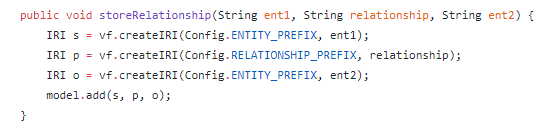
+ Các bước đưa dữ liệu lên cơ sở dữ liệu(CSDL):

1. Lưu trữ các Triple trước khi tải lên CSDL kiểu model cây (TreeModel)
2. Tạo nên 1 triple từ 3 thành phần Subject, Predicate và Object. Sử dụng ValueFactory **vf** để tạo nên các s, p, o và cho vào model:

* Tạo thực thể định danh



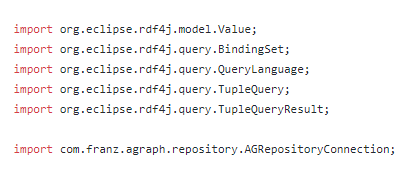
* Tạo s, p, o và cho vào model



1. Tải model lên CSDL

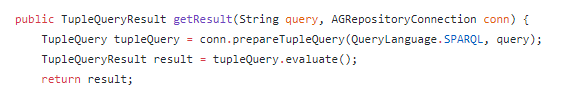
* Truy vấn dữ liệu

+ Các thư viện cần có:

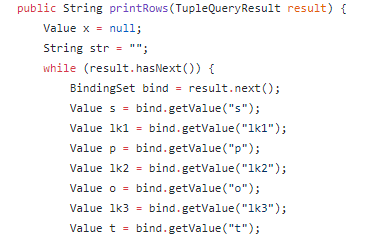


+ Các bước truy vấn:

* Dùng TupleQuery để lưu lại kết quả trả về từ cơ sở dữ liệu qua method prepareTupleQuery của connection



* Sau khi có được TupleQuery, chuyển mỗi một dòng của bảng trả về thành một BindingSet, sau đó xử lý BindingSet thành kết quả



# Chương 2. Trình bày về mô hình dữ liệu

## Lưu trữ dữ liệu trên server.

- Tạo ra 4 namespace là : ent <http://btl/entity>; prs <http://btl/properties/>; rel <http://btl/relationship>; class <http://btl/class/> để sử dụng khi truy vấn cũng như dễ dàng trong việc lưu trữ dữ liệu.

- Chương trình tạo ra 6 class tương ứng với 6 loại thực thể là Person, Location, Organization, Country, Event, Time bằng cách sử dụng rdf:type trong khi add triple lên server. Mỗi thực thể sẽ được add vào các class tương ứng. Các class được thể hiện bằng prefix class (<http://btl/class>).

- Về cách lưu trữ thực thể, mỗi thực thể sẽ được thể hiện bởi một id và có prefix dạng ent (<http://btl/entity>) . Id của thực thể sẽ liên kết với các thuộc tính của thực thể qua predicate có prefix là prs (<http://btl/properties/>), sau nó là 1 literal lưu giá trị thuộc tính của thực thể.

VD : ent:Person1 prs:tên\_hiển\_thị "Harry"

ent:Person1 prs:giới\_tính "Nam"

Ví dụ trên dùng để lưu trữ 1 người có Id là Person1, tên hiển thị là Harry và có giới tính là nam.

- Về cách lưu trữ các quan hệ, mỗi quan hệ sẽ thể hiện bởi 2 Id của 2 thực thể liên quan và relationship giữa chúng. 2 thực thể sẽ có prefix là ent (<http://btl/entity/>), quan hệ giữa chúng có prefix là rel (<http://btl/relationship/>).

VD: ent:Person1 rel:sinh\_ra ent:Person2

Ví dụ trên thể hiện rằng Person1 đã sinh ra Person2.

- Các giá trị literal được lưu trữ chủ yếu dạng String và int để dễ xử lý.

## Lưu trữ dữ liệu trong project

- Project lưu trữ dữ liệu trong 3 thư mực : RawData/Entity, RawData/Description và RawData/Relationship.

- Thư mực Entity gồm 6 file txt, mỗi file lưu tên hiển thị của một loại thực thể. Các tên hiển thị được lấy trên mạng qua các trang sinh ngẫu nhiên dữ liệu. Số lượng tên hiển thị của mỗi loại thực thể nằm trong khoản từ 220 đến 3000.

- Thư mục Description gồm 6 file txt, mỗi file mô tả của các thực thể tương ứng. Các mô tả này được lấy ngẫu nhiên trên Wiki và sẽ được ghép ngẫu nhiên với các thực thể (trình bày chi tiết trong phần kế tiếp). Mỗi file lưu 20 miêu tả.

- Thư mục Relationship gồm 19 file txt, có dạng <Entity1><Entity2> thể hiện sự liên kết giữa loại thực thể 1 và lọai thực thể 2. Ví dụ file có tên PersonEvent miêu tả liên kết giữa Person và Event. Những liên kết này là 1 chiều. Do một số cặp thực thể có ít liên kết (TimeTime) nên chúng em chỉ lấy 19 file liên kết để sinh dữ liệu và bỏ qua các cặp thực thể ít liên kết. Các cặp thực thể này gồm: CountryCountry, EventPerson, LocationPerson, OrganizationPerson, PersonOrganization, CountryEvent, EventTime, OrganizationEvent, PersonCountry, PersonPerson, EventCountry, LocationEvent, OrganizationLocation, PersonEvent, PersonTime, EventOrganization, LocationLocation, OrganizationOrganization, PersonLocation.

## Giải thuật sinh dữ liệu

### Sinh thực thể

- Đọc tên thực thể, miêu tả từ file. Lưu chúng vào mảng với tên là listEntity cho list tên thực thể và listDescription cho list miêu tả. Do có 6 loại thực thể nên listEntity và listDescription là các mảng có 6 phần tử. Mỗi phần tử là 1 ArrayList chứa tên hoặc miêu tả của các thực thể tương ứng.

Ví dụ cách lưu trữ mảng listEntity:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | listEntity[0] | listEntity[1] | listEntity[2] | listEntity[3] | listEntity[4] | listEntity[5] |
| 1 | 2018-06-11 | 3M | Brasília, Brazil | WWDC 2018 | Aruba | Shanti Wise |
| 2 | 2016-08-02 | AT&T | Huambo, Angola | Telefilm | Angola | Shri Pham |
| 3 | 2002-01-01 | Zara | Palmas, Brazil | VIETNAM EXPO | Afghanistan | Sidi Warren |
| ... | ... | ... | ... | ... | ... | ... |

- Đối với các bộ dữ liệu, ta có 1 mảng lưu số lượng các loại thực thể cần sinh. Ví dụ với bộ 100 thực thể, ta có 1 mảng là {10, 10, 10, 10, 10, 10, 50}, với thứ tự lần lượt là {Time,Organization, Location, Event, Country, Person}. Nghĩa là với bộ 100 thực thể, ta có 10 Time, 10 Organization, 10 Location, 10 Event, 10 Country, 50 Person. Lý do số lượng Person nhiều hơn là do số lượng các Person lớn hơn số lượng các thực thể còn lại.

- Số lượng các thực thể tương ứng với các bộ dữ liệu được nêu trong bảng sau :

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| (M,N) | Time | Organization | Location | Event | Country | Person |
| 100, 200 | 10 | 10 | 10 | 10 | 10 | 50 |
| 5000, 7000 | 100 | 100 | 100 | 100 | 100 | 4500 |
| 60000, 80000 | 220 | 220 | 220 | 220 | 220 | 58900 |
| 100000, 200000 | 220 | 220 | 220 | 220 | 220 | 98900 |
| 500000, 1000000 | 220 | 220 | 220 | 220 | 220 | 498900 |

- Sau khi đọc dữ liệu, ta có 2 mảng là listEntity và listDescription chứa tập tên và miêu tả của thực thể. Trước tiên ta sinh 3 chỉ số cho thực thể là Id, tên hiển thị và miêu tả (lý do các thuộc tính này được sinh riêng bởi ta phải đọc dữ liệu trong mảng ArrayList). Sau đó ta sinh 1 thực thể có đầy đủ thông tin bằng Id, tên hiển thị mà miêu tả đã có. Lặp lại các bước đó, ta sẽ có đủ số thực thể cần sinh. Chi tiết:

- Sử dụng 1 vòng for từ 1 đến số lượng thực thể cần sinh. Trong mỗi vòng lặp :

* + Lấy loại thực thể ghép lại biến chạy của vòng lặp để làm Id cho thực thể (VD:Person1) -> ID
  + Random 1 số nguyên, được x, ta lấy giá trị x của listEntity làm tên gán cho thực thể đang xét. -> Tên hiển thị
  + Random 1 số nguyên, được y, ta lấy giá trị y của listDescription làm miêu tả của thực thể đang xét. -> Miêu tả
  + Sau đó gọi phương thức createEntity của class SingleEntityCreator để tạo thực thể khi đã có 3 thuộc tính là Id, name, description.

- Hàm createEntity của class SingleEntityCreator nhận đầu vào là Id, tên hiển thị và miêu tả. Phụ thuộc vào Id nhận vào mà chương trình sẽ tạo các thực thể tương ứng (Person1 -> tạo thực thể Person). Sau khi tạo thực thể, chương trình sẽ gọi các hàm tạo thuộc tính với các thực thể tương ứng :

public AEntity createEntity() {

AEntity ent;

if (id.startsWith("Person")) {

ent = createPerson();

} else if (id.startsWith("Country")) {

ent = createCountry();

} else if (id.startsWith("Location")) {

ent = createLocation();

} else if (id.startsWith("Organization")) {

ent = createOrganization();

} else if (id.startsWith("Time")) {

ent = createTime();

} else

ent = createEvent();

return ent;

}

- Trong class có 6 hàm tạo thực thể tương ứng với 6 loại thực thể và 1 hàm tạo link, ngày nhận dữ liệu dùng chung.

\* Tạo ngẫu nhiên ngày tháng và link nhận dữ liệu của thực thể..

private AEntity create(AEntity p) {..

\* @param p : Thực thể Person sau khi được tạo ngẫu nhiên..

private Person createPerson() {..

\* @return p : Thực thể Organization sau khi được tạo ngẫu nhiên ..

private Organization createOrganization() {..

\* @return p : Thực thể Location sau khi được tạo ngẫu nhiên ..

private Location createLocation() {..

\* @return p : Thực thể Country sau khi được tạo ngẫu nhiên..

private Country createCountry() {..

\* @return p : Thực thể Event sau khi được tạo ngẫu nhiên ..

private Event createEvent() {..

\* @return p : Thực thể Time sau khi được tạo ngẫu nhiên

private Time createTime() {..

- Trong các hàm tạo thực thể, ta có 1 đối tượng của class RandomProperties để sinh ngẫu nhiên giá trị random của thuộc tính các thực thể . Ví dụ hàm tạo Person :

\* @param p : Thực thể Person sau khi được tạo ngẫu nhiên..

private Person createPerson() {

Person p = new Person (id, name, description);

p = (Person) create(p);

p.setNgaySinh(r.getRandomDate());

p.setTuoi(r.getRandomInt(100));

p.setGioiTinh(r.getRandomGioiTinh());

return p;

}

- Ta có r là đối tượng của lớp RandomProperties, r sẽ gọi các phương thức random như getRandomDate để tạo ngẫu nhiên ngày tháng, getRamdonInt(100) để tạo ngẫu nhiên các số <100, getRamdonGioiTinh để tạo random giới tính là nam hay nữ.

- Sau bước này, ta đã có 1 thực thể được tạo các thuộc tính hoàn chỉnh. Return của hàm CreateData là 1 AEntity, tuy nhiên lúc tạo thực thể, ta tạo các đối tượng của các lớp Person, Event... Nên khi return, dù các thực thể được gán nhãn là AEntity nhưng thực chất nó là các đối tượng của các lớp Person, Event ... Đây là 1 hình thức Upcasting để tránh lặp code.

- Trên đây là chi tiết phần tạo thực thể. Mỗi khi tạo xong 1 thực thể, ta sẽ gọi 1 phương thức để lấy các thuộc tính của thực thể đó, và add các triple của thực thể vào model, kiểm tra số triple trong model đủ lớn chưa (nếu triple trong model quá lớn sẽ dẫn đến lỗi, trong chương trình mặc định là > 250000 triple sẽ là đủ lớn), khi đủ lớn sẽ add lên repository. Do tốc độ triple vào model nhanh hơn nhiều với tốc độ add triple trực tiếp vào repository.

### Sinh quan hệ

- Ta đọc các quan hệ từ file, lưu trong listRelationship, listRelationship là 1 mảng 2 chiều, trong đó listRelationship[i][j] lưu 1 ArrayList chứa các quan hệ của Entity i và Entity j. Ví dụ listRelationship[Person][Event] (Person và Event sẽ được ánh xạ thành các số để dễ xử lý) lưu ArrayList chứa quan hệ giữa Person và Event.

- Ta chỉ cần đơn giản chọn ngẫu nhiên 2 thực thể và chọn ngẫu nhiên quan hệ giữa 2 loại thực thể đấy và lưu vào model. Đến khi lượng triple trong model đủ lớn thì sẽ lưu vào server.

- Ví dụ ta chọn ngẫu nhiên được 2 thực thể là Event và Time, ta sẽ lấy random relationship trong listRelationship[Event][Time] và lưu vào model.

# Chương 3: Trình bày về chương trình của nhóm

## Package và các lớp

Chương trình của nhóm em xây dựng bao gồm 9 package là : connnection, createdata, entity, filereader, myinterface, query, setting, time. Mỗi package trong chương trình đảm nhận những nhiệm vụ khác nhau. Cụ thể như sau:

### Package entity



Package entity là một nhóm các lớp biểu diễn các thực thể được yêu cầu trong bài tập lớn. Các thực thể ở đây được trích rút từ các bài báo mạng bao gồm: con người (PERSON), tổ chức (ORGANIZATION), địa danh (LOCATION), đất nước (COUNTRY), sự kiện (EVENT) và thời gian (TIME). Các thực thể này đều có chung các thuộc tính như : định danh, tên hiển thị, mô tả, link trích rút, ngày tháng trích rút, do đó ta đặt 1 abtract class AEntity để đại diện cho các thực thể.

#### Class AEntity

* + Thuộc tính : private String dinhDanh, private tenHienThi, private String moTa, private String link, private String ngayThang
    - Các thuộc tính là các thuộc tính chung của các thực thể cần lưu. Các thuộc tính này được đặt là private và được lấy ra hoặc thay đổi bằng các phương thức getter, setter
  + Phương thức :
    - Các phương thức getter, setter dùng để lấy và thay đổi các thuộc tính private trong class
    - getListProperties : Phương thức dùng để lấy tất cả các thuộc tính của class, do các class kế thừa AEntity có các thuộc tính khác nhau, nên ta dùng hashmap để lưu các giá trị thuộc tính. Phương thức này dùng khi lưu dữ liệu vào server, nó sẽ lấy tất cả các thuộc tính của đối tượng thực thể và lưu vào server dưới dạng triple. Ví dụ, giá trị link được lưu ở dạng <Key="link", Value=getLink()>

/\*\*

\* Lưu các thuộc tính của Entity vào list.

\* Các lớp thực thể khác muốn thêm các thuộc tính mới có thể kế thừa phương thức này

\*/

public HashMap<Object, Object> getListProperties() {

HashMap<Object, Object> properties = new HashMap<Object, Object>();

properties.put("định\_danh", dinhDanh);

properties.put("thực\_thể", this.getClass().getName().replace("entity.", ""));

properties.put("tên\_hiển\_thị", tenHienThi);

properties.put("mô\_tả", moTa);

properties.put("link\_nhận\_dữ\_liệu", link);

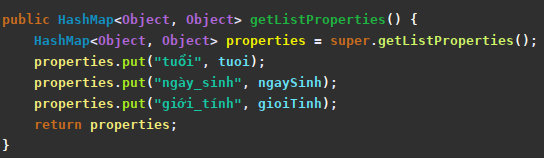
properties.put("ngày\_nhận\_dữ\_liệu", ngayThang);

return properties;

}

- Ngoài ra, với mỗi đặc trưng của từng thực thể, nhóm bổ sung thêm một vài các thuộc tính cho các thực thể. Cụ thể:

* + Class Person : tuổi, ngày sinh, giới tính.
  + Class Location : tọa độ.
  + Class Country : dân số, GDP, diện tích.
  + Class Organization : ngày thành lập
  + Phương thức : Ngoài phương thức được kế thừa từ AEntity, các class có thêm phương thức getter, setter cho các thuộc tính thêm, getListProperties
* getListProperties : Gọi phương thức getListEntity từ lớp cha, đồng thời thêm các thuộc tính chưa có vào HashMap



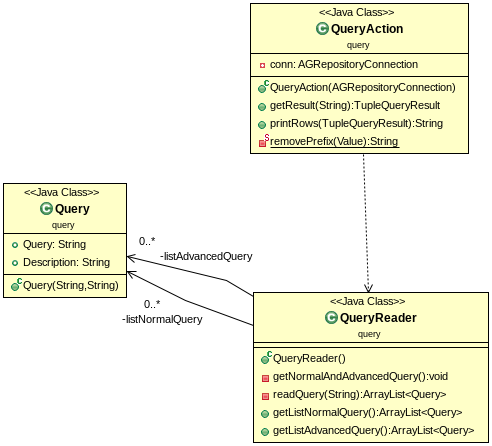
### Package filereader

Package filereader chứa 1 abtract class là AFileReader. Lưu các thuộc tính và phương thức cần thiết cho việc đọc file

- Class AFileReader : Là abtracs class dùng để đọc file, các lớp QueryReader, DataReader muốn đọc file phải kế thừa lớp này

* + Thuộc tính :
    - public File file : Đối tượng File
    - scanner: Đối tượng đọc dữ liệu từ môi trường File
* Phương thức :
  + protected boolean exists : Đầu vào là 1 path file, hàm sẽ kiểm tra xem file có tồn tại hay không

### Package query



Package query gồm 3 class Query, QueryAction, QueryReader. Các lass đều liên quan đến query như lưu cấu trúc của query, đọc query từ file và xử lý đọc query

#### Class Query

Lưu dạng của query, gồm 2 trường description và query, lưu miêu tả của query và câu query tương ứng.

* + Thuộc tính :
    - public String Description : Miêu tả query
    - public String Query : Nội dung query

#### Class QueryReader

Dùng để đọc query từ file

* + Thuộc tính :
    - listNormalQuery, private ArrayList listNormalQuery : Lưu list các câu truy vấn cơ bản
    - private ArrayList listAdvancedQuery : Lưu list các câu truy vấn phức tạp
  + Phương thức : private void getNormalAndAdvancedQuery, private ArrayList readQuery, public ArrayList getListNormalQuery, public ArrayList getListAdvancedQuery
    - readQuery: Đọc file lưu query, đầu ra là 1 list Query bao gồm cả miêu tả và nội dung câu query
    - getNormalAndAdvancedQuery : Gọi hàm readQuery để đọc 2 giá trị listNormalQuery và listAdvancedQuery từ file
    - getListNormalQuery: Trả ra listNormalQuery
    - getListAdvancedQuery: Trả ra listAdvancedQuery

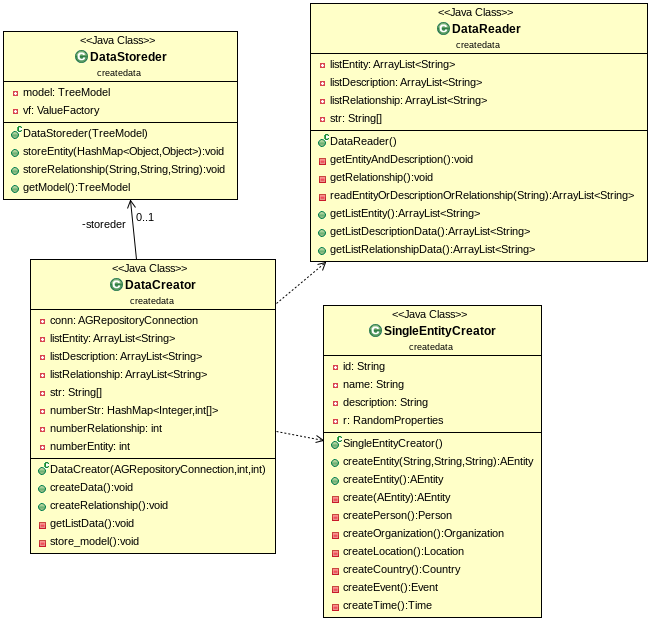
#### Class QueryAction

Chứa các phương thức để truy vấn câu query từ cơ sở dữ liệu

* + Thuộc tính :
    - private AGRepositoryConnection conn : Dùng để kết nối với repository, đối tượng này sẽ gửi query lên server và nhận lại kết quả
  + Phương thức : public String getResultNormalQuery, , , ,
    - private String removePrefix : Sau khi truy vấn, các giá trị trả ra là các IRI hoặc Literal, sẽ có các tiển tố hoặc hậu tố đi theo, phương thức này sẽ loại bỏ các tiền tố và hậu tố đó để chương trình hiển thị kết quả gần với ngôn ngữ con người nhất
    - public String printRow : Phương thức dùng để hiển thị kết quả các câu truy vấn
    - public TupleQueryResult getResult : Phương thức dùng để gửi câu truy vấn lên server và nhận result

Luồng dữ liệu : Lớp QueryReader sẽ đọc dữ liệu từ file vào các listQuery, những truy vấn đọc được sẽ được lưu ở dạng class Query. Sau khi có các listQuery, lớp QueryAction sẽ nhận các query đó, sử dụng các hàm có sẵn của mình để gửi truy vấn lên server, nhận kết quả, loại bỏ tiền tố hậu tố rồi đưa ra kết quả. Kết quả này sau đó sẽ được hiển thị trên console để test hoặc hiển thị trên giao diện người dùng

### Package createdata

Package createdata gồm các lớp để đọc dữ liệu các thực thể, miêu tả và quan hệ của chúng rồi sinh ngẫu nhiên các thực thể theo yêu cầu của đề bài.

#### Class DataReader

Dùng để đọc dữ liệu thực thể được chuẩn bị sẵn, gồm có các file tên thực thể, file miêu tả và file quan hệ giữa các thực thể.

* + Thuộc tính : private ArrayList[] listEntity, list Description[], list Relationship[]
    - Các thuộc tính dùng để lưu list tên thực thể, miêu tả và quan hệ giữa các thực thể đọc được từ file. Cấu trúc lưu trữ những list này được nêu chi tiết ở phần sinh dữ liệu
  + Phương thức :
  + Private ArrayList readEntityOrDescriptionOrRelationship(String path) : Hàm có tác dụng đọc list entity, description hoặc relationship từ file vào 1 list rồi trả ra list đó. Tham số path là đường dẫn tới file cần đọc.
  + Private void getListEntityAndDescription(): Hàm dùng để nhận list entity hoặc description sau khi đã được đọc từ hàm trên. Do hàm readEntityOrDescriptionOrRelationship chỉ đọc các list của từng thực thể. Ví dụ mỗi lần đọc chỉ có thể trả ra được 1 list Person hoặc 1 list Event ... Do đó cần gọi hàm này 6 lần để đọc hết được tên các thực thể (do file chỉ lưu tên thực thê), và 6 lần để đọc miêu tả các thực thể đó. Sau khi đọc, tên thực thể hoặc miêu tả sẽ được lưu vào biến listEntity hoặc listDescription. 2 thuộc tính này là các mảng của Arraylist gồm 6 phần tử, mỗi phần tử tương ứng với một thực thể. Miêu tả chi tiết mảng listEntity và listDescription được trình bày ở phần sinh dữ liệu

VD listEntity

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | listEntity[0] | listEntity[1] | listEntity[2] | listEntity[3] | listEntity[4] | listEntity[5] |
| 1 | 2018-06-11 | 3M | Brasília, Brazil | WWDC 2018 | Aruba | Shanti Wise |
| 2 | 2016-08-02 | AT&T | Huambo, Angola | Telefilm | Angola | Shri Pham |
| 3 | 2002-01-01 | Zara | Palmas, Brazil | VIETNAM EXPO | Afghanistan | Sidi Warren |
| ... | ... | ... | ... | ... | ... | ... |

* + - Private void getRelationship : hàm dùng để nhận tập relationship đọc được và lưu vào listRelationship. Đây là 1 mảng 2 chiều chứa các ArrayList, miêu tả chi tiết hàm này được nêu ở phần sinh dữ liệu
    - Public ArrayList[] getListEntity, ArrayList[] getListDescription, ArrayList[][] getListRelationship : hàm dùng để trả ra các giá trị private của hàm.

#### Class DataCreator

Hàm có chức năng tạo dữ liệu từ các list có được nhờ gọi dịch vụ của DataReader, tạo dữ liệu ngẫu nhiên cho các thực thể, tạo quan hệ cho các thực thể đồng thời lưu dữ liệu tạo được vào server

* + Thuộc tính
    - private AGRepositoryConnection conn : Đối tượng kết nối với repository, dùng để gửi dữ liệu lên server
    - private DataStoreder storeder : Đối tượng dùng để lưu dữ liệu vào model, lý do sử dụng model bởi tốc độc add triple vào model nhanh hơn nhiều so với tốc độ add triple vào conn. Khi model đã đủ lượn triple, ta mới add vào conn, như thế sẽ giảm thiểu tối đa số lần phải truy xuất vào server, từ đó tăng tốc độ lưu trữ.
    - private HashMap<Integer, int[]> numberStr : Đối tượng dùng để lưu số lượng mỗi thực thể cần phải tạo. Trong đó:
      * Key : là 1 giá trị Integer lưu tổng số thực thể cần tạo (cần tạo 100 thực thể).
      * Value : là 1 tập lưu số lượng mỗi thực thể riêng lẻ cần tạo. ({10, 10, 10, 10, 10, 50} là cần tạo 50 person, những thực thể còn lại cần tạo 10)
      * Khi ta gọi numberStr.get(100), hàm sẽ trả ra số lượng mỗi thực thể cần tạo khi muốn tạo 1 repository có 100 thực thể.
    - private String[] str : Đối tượng ánh xạ tập 6 thực thể sang số, nhằm mục đích thuận lợi hơn trong lập trình. Khi ta gọi str[0] nghĩa là ta đang muốn nói đến thực thể Time, gọi str[5] nghĩa là ta đang muốn nói đến thực thể Person: String[] str = { "Time", "Organization", "Location", "Event", "Country", "Person"};
    - private int numberRelationship, numberEntity : Thuộc tính lưu số lượng thực thể và số lượng quan hệ cần tạo.
    - Các thuộc tính listEntity, listDescription, listRelationship đã được nêu chi tiết trong class trước
  + Phương thức :
    - private void getListData : Được gọi trong constructor, hàm dùng để nạp dữ liệu vào các thuộc tính listEntity, listDescription, listRelationship từ class DataReader.
    - private void store\_model : Được sử dụng khi model trong thuộc tính storeded đã đầy, khi đó ta sẽ lưu add model vào server và xóa các triple trong model để tiếp tục add các triple mới.
    - public void createData : hàm dùng để tạo các thực thể, đã được nêu rõ trong phần sinh dữ liệu. Các thực thể sau khi được tạo sẽ được add vào model nhờ thuộc tính storeder. Mỗi lần add xong sẽ kiểm tra số lượng triple trong model, nếu đủ số triple sẽ gọi hàm store\_model để add model lên server.
    - public void createRelationship : hàm dùng để tạo các quan hệ giữa 2 thực thể được chọn bất kỳ, đã được nêu rõ trong phần sinh dữ liệu. Các thực thể sau khi đượ tạo sẽ được add vào model nhờ thuộc tính storeder. Mỗi lần add xong sẽ kiểm tra số lượng triple trong model, nếu đủ số triple sẽ gọi hàm store\_model để add model lên server.

#### Class SingleEntityCreator

Lớp dùng để tạo ra 1 thực thể từ id, tên hiển thị và miêu tả của đối tượng đó. Sử dụng upcasting

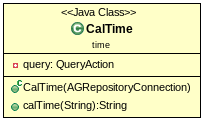
* + Thuộc tính
    - private String id, name, description : Lưu id, tên hiển thị, miêu tả của thực thể muốn tạo
    - private RandomProperties r : Đối tượng dùng để tạo ngẫu nhiên các thuộc tính của thực thể như tuổi, ngày sinh, giới tính ...
  + Phương thức
    - public AEntity createEntity (String, String, String) : Đầu vào là 3 String lưu lần lượt Id, tên hiển thị và miêu tả của thực thể. Hàm sẽ gán 3 String này cho 3 thuộc tính của class và thực thiện tiếp tục hàm createEntity()
    - public AEntity createEntity() : có chức năng tạo ra 1 thực thể. Đầu tiên khai báo 1 AEntity, sau đó tùy vào id mà ta tạo các thực thể tương ứng (upcasting). Với từng thực thể, ta sẽ sử dụng các hàm tạo dữ liệu riêng. Đã được nêu rõ trong phần tạo dữ liệu.
    - Các phươn thức create dùng đối tượng của lớp RandomPropertier để tạo ngẫu nhiên dữ liệu cho thực thể, đã nêu rõ ở phần tạo dữ liệu.

#### Class DataStoreder

Lớp dùng để lưu trữ dữ liệu vào model.

* + Thuộc tính :
* private TreeModel model : model dùng để tạm thời lưu trữ dữ liệu
* private ValueFactory vf : đối tượng dùng để tạo các IRI và Literal.
  + Phương thức :
    - public void storeEntity (HashMap properties) : Tham số properties có dạng HashMap lưu thông tin các thuộc tính của thực thể. Hàm có chức năng lấy lần lượt các thuộc tính đó, lưu dưới dạng triple rồi add vào model.
    - public void storeRelationship(String, String, String) : Tham số đầu vào là 3 String lưu lần lượt id thực thể 1, quan hệ giữa 2 thực thể, id thực thể 2. Hàm có chức năng tạo 1 triple quan hệ từ 3 String trên và add vào model.
    - public TreeModel getModel : Trả ra thuộc tính model.

### Package time

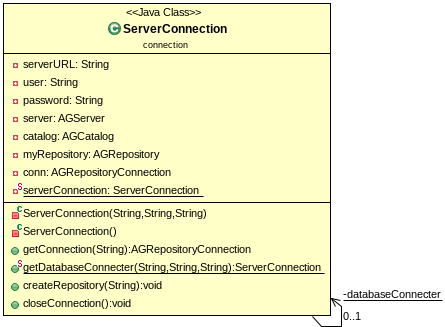


Package time chứa 1 class time dùng để đo thời gian truy vấn.

#### Class CalTime

* + Thuộc tính
    - private QueryAction queryAction : Dùng để thao tác với query được truyền vào, như tính kết quả truy vấn, in kết quả …
  + Phương thức :
    - public String calTime (String) : Đầu vào là 1 String chứa query cần truy vấn. Hàm sẽ lưu lại thời gian bắt đầu truy vấn và thời gian kết thúc truy vấn, từ đó tính được thời gian chạy truy vấn. Sau đó sẽ in ra kết quả câu truy vấn và thời gian chạy.

### Package connection



Package connection gồm 1 class dùng để kết nối với server, đồng thời có các phương thức để trả ra 1 kết nối với repository muốn thao tác, phương thức để đóng repository ...

#### Class ServerConnection

Dùng để kết nối với server. Có sử dụng singleton pattern để giới hạn luôn chỉ có 1 kết nối tới server. Từ đó chỉ có 1 kết nối tới repository, tránh bị xung đột khi muốn kết nối tới nhiều repository

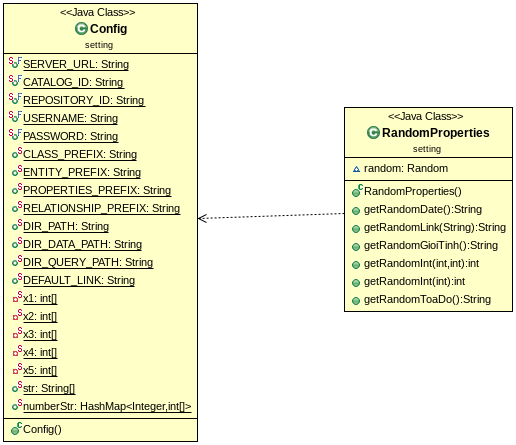
* + Thuộc tính :
    - private String serverURL, user, password dùng để xác định server cần đăng nhập cùng với tên user và password,
    - private AGServer server : Đối tượng server, sẽ được tạo ra trong constructor của class thông qua các tham số là serverURL, user, password
    - private AGCatalog catalog : Đối tượng catalog, sẽ được tạo trong constructor nhờ lấy root catalog của server.
    - private AGRepository myRepository : Đối tượng repository, lưu repository mà server đang kết nối tới
    - privateAGRepositoryConnection conn : Đối tượng connection, sử dụng để giao tiếp với repository.
    - private static ServerConnection serverConnection : Đối tượng dùng để kết nối với server, do sử dụng pattern singleton nên sẽ để là private static.
  + Phương thức
    - public static ServerConnection getDatabaseConnector (String, String, String) : Truyền vào 3 String lần lượt là serverURL, user, password.
      * Chương trình sẽ kiểm tra xem đã tồn tại đối tượng ServerConnection chưa, nếu chưa tồn tại sẽ gọi constructor để tạo đối tượng với 3 chỉ số tương ứng.
      * Trong trường hợp đã tồn tại đối tượng, hàm sẽ kiểm tra xem đối tượng đã tồn tại có 3 chỉ số trùng với tham số truyền vào không. Nếu không trùng tức làn người dùng muốn kết nối tới server khác, lúc này ta sẽ đóng server hiện tại và gọi contructor để kết nối với server cần kết nối
      * Trong trường hợp trùng với server hiện có, hàm trả ra đối tượng ServerConnection hiện có

Do lớp sử dùng pattern singleton nên phương thức này đóng vai trò như constructor, dùng để tạo (nếu chưa có) hoặc lấy đối tượng (nếu đã tồn tại)

* + - public AGRepositoryConnection getConnection (String) : Truyền vào tên của repository, hàm sẽ kiểm tra xem repository đã tồn tại chưa, nếu chưa tồn tại sẽ tạo 1 repository mới, rồi trả ra đối tượng conn đã kết nối với repository đó.
    - public void createRepository (String) : Tham số truyền vào là tên của repository cần tạo. Hàm sẽ kiểm tra xem repository đã tồn tại chưa, nếu đã tồn tại thì sẽ xóa repository và tạo 1 repository mới. Đồng thời thêm các prefix và cài đặt để repository tự xóa các triple trùng lặp.
    - public void closeConnection : Hàm dùng để đóng repository hiện tại.

### Package setting

Chứa lớp config và các lớp nền, đóng vai trò dịch vụ cho các lớp ở tầng trên sử dụng .



#### Class Config

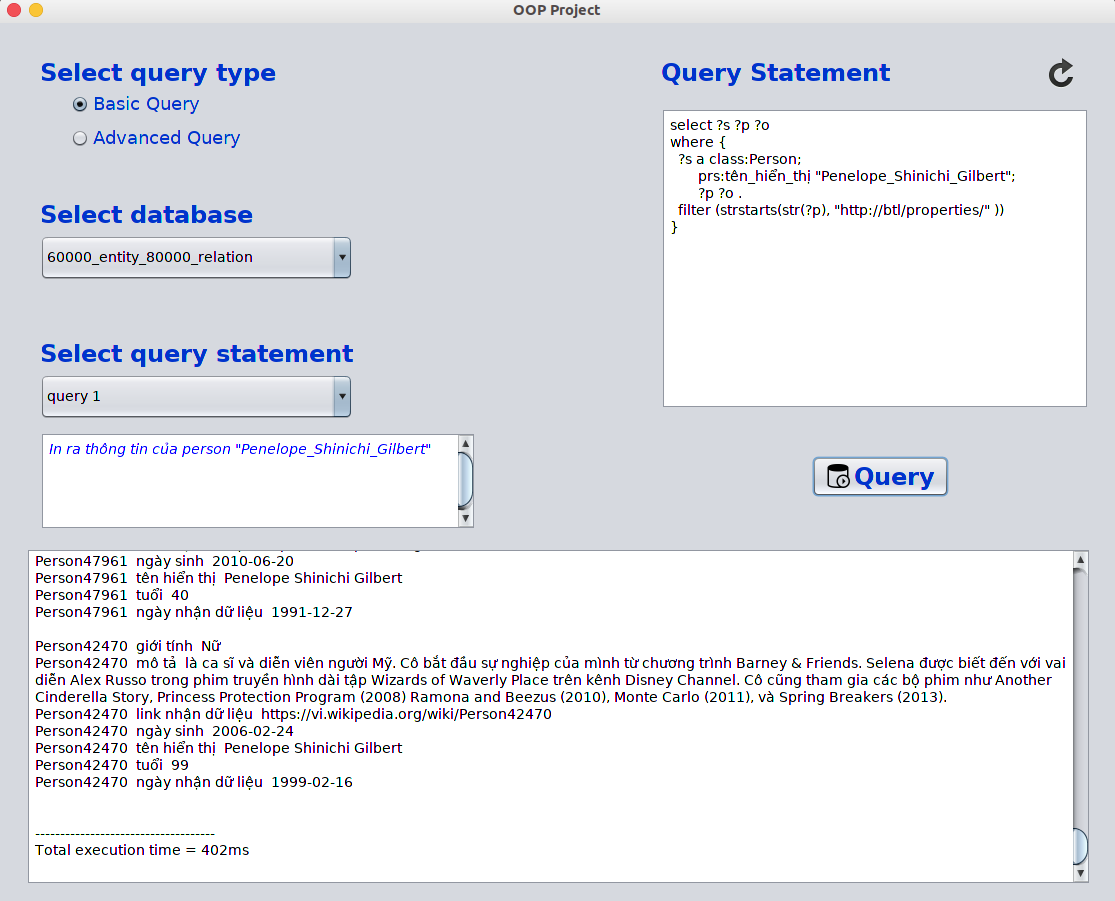
Chỉ chứa các thuộc tính là các final String, dùng để cấu hình chương trình, ví dụ như user mặc định, password mặc định, path dẫn đến các thư mục ..., các prefix …

#### Class RandomProperties

Chứa các hàm tạo ngẫu nhiên các thuộc tính của thực thể, ví dụ như tạo random giới tính, random ngày sinh, random tọa độ

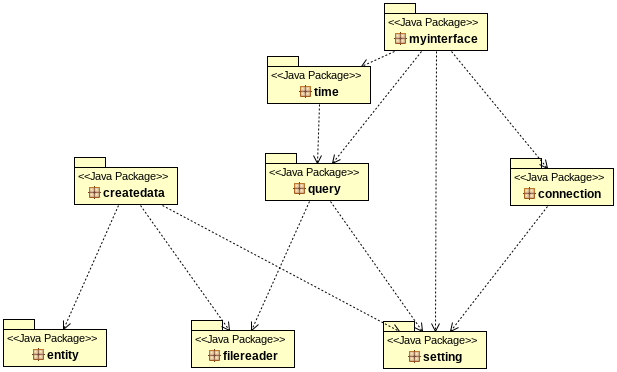
### Package myinterface

Chứa lớp MainForm đóng vai trò tạo giao diện cho chương trình. trong package này có



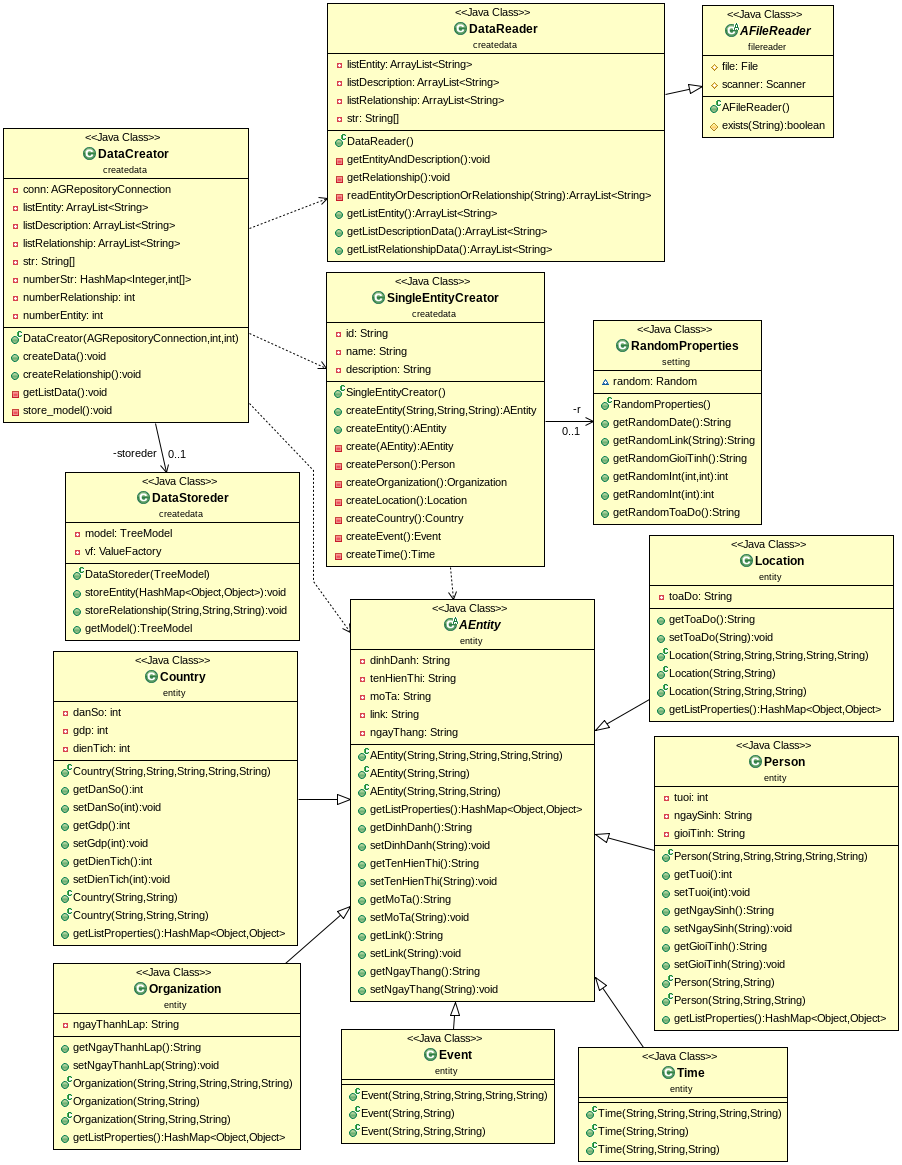
* + - Phương thức
      * comboChooseStatementActionPerformed bắt sự kiện của combobox: comboChooseStatement. Ví dụ khi kich chọn item: "query 1" thì trên giao diện sẽ hiển thị lên tên truy vấn và cú pháp truy vấn ứng với query1.
      * newQueryMouseewQueryMouseClicked bắt sự kiện click chuột vào label reset, lúc này câu truy vấn và cú pháp truy vấn sẽ được reset lại. Tại đây, người dùng có thể nhập truy vấn của họ vào để thực thi.
      * queryRunActionPerformed đóng vai trò thực hiện truy vấn theo cú pháp có trên màn hình. Sau khi kich vào button Query, sẽ nhận được kết quả của câu truy vấn như hình trên.

## Sơ đồ biểu thị sự phụ thuộc giữa các package



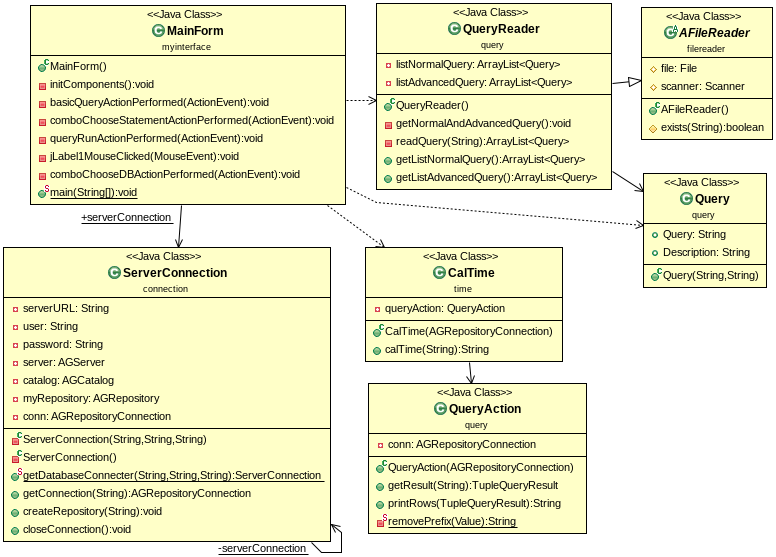
## Sơ đồ lớp

Chương trình chia thành 2 luồng chính : CreateData và Query

CreateData : 

Lớp DataReader là lớp con của AFileReader, có chức năng đọc dữ liệu từ file. Lớp DataCreator nhận dữ liệu do DataReader đọc, sử dụng các dịch vụ của SingleEntityCreator và DataStoreder để sinh dữ liệu và lưu vào server. Lớp SingleEntityCreator sử dụng RandomProperties để sinh ngẫu nhiên các thuộc tính cho thực thể đang tạo

Query :



Khi người dùng chọn các bộ dữ liệu, chương trình sẽ tạo kết nối đến repository tương ứng bằng ServerConnection. Sau đó người dùng chọn bộ câu hỏi (cơ bản, nâng cao), chương trình sẽ gọi lớp QueryReader để đọc truy vấn từ file và hiển thị. Khi người dùng chọn câu hỏi để truy vấn, chương trình sẽ gọi lớp CalTime, CalTime sẽ gọi lớp QueryAction để thực hiện truy vấn đồng thời đo thời gian truy vấn và gửi kết quả về cho mainform. Từ đó mainform sẽ hiển thị kết quả lên màn hình.

# Chương 4 : Kiểm thử hiệu năng

## Các truy vấn cơ bản

### Truy vấn 1

Mô tả : In ra thông tin của person "Penelope\_Shinichi\_Gilbert"

Ý tưởng : Đặt điều kiện trong phần where để tìm những người có tên thỏa mãn, đặt điều kiện trong filter để lọc và lấy những triple chứa thuộc tính (không chứa relationship) và in kết quả ra màn hình.

Nội dụng :

select ?s ?p ?o

where {

?s a class:Person;

prs:tên\_hiển\_thị "Penelope\_Shinichi\_Gilbert";

?p ?o .

filter (strstarts(str(?p), "http://btl/properties/" ))

}

### Truy vấn 2

Mô tả : In ra thông tin sự kiện "APPLE\_SPECIAL\_EVENT" được tổ chức đầu tiên tại "Norway"

Ý tưởng : Tìm id của “Norway”, sau đó tìm những sự kiện tên "APPLE\_SPECIAL\_EVENT” và được tổ chức đầu tiên tại id vừa tìm được. Dùng filter để lọc những triple chỉ chứa thuộc tính.

Nội dung :

select ?s ?p ?o

where {

?idx a class:Country;

prs:tên\_hiển\_thị "Norway".

?s a class:Event;

prs:tên\_hiển\_thị "APPLE\_SPECIAL\_EVENT";

rel:được\_tổ\_chức\_đầu\_tiên\_tại ?idx;

?p ?o .

filter (strstarts(str(?p), "http://btl/properties/"))

}

### Truy vấn 3

Mô tả : In thông tin 10 người đầu tiên sinh ra tại Location "Chongjin,\_North\_Korea"

Ý tưởng : Tìm id của location "Chongjin,\_North\_Korea", sau đó tìm id của những người đã sinh ra tại id vừa tìm được. Do yêu cầu 10 người nên ta đặt tất cả vào trong 1 sub query và đặt limit là 10. Sau khi kết thúc ta đã được id của 10 người thỏa mãn đề bài, ta tiến hành tìm thông tin của người đó. Dùng filter để lọc những triple chỉ chứa thuộc tính

Nội dung :

select ?s ?p ?o

where {

{

select ?s ?p ?o where {

?idx a class:Location;

prs:tên\_hiển\_thị "Chongjin,\_North\_Korea".

?s a class:Person;

rel:﻿sinh\_ra\_tại ?idx.

}

limit 10

}

?s ?p ?o.

filter (strstarts(str(?p), "http://btl/properties/" ))

}

### Truy vấn 4

Mô tả : In ra số nơi mà công ty "Heineken\_Brewery" phá sản

Ý tưởng : Tìm id của công ty "Heineken\_Brewery", đặt là idy. Tìm id của những nơi đã phá sản tại idx, đặt là idy. Đếm số idy vừa tìm được

Nội dung :

select (count(?idx) as ?s)

where {

?idx a class:Location.

?idy a class:Organization;

prs:tên\_hiển\_thị "Heineken\_Brewery";

rel:phá\_sản\_tại ?idx .

}

### Truy vấn 5

Mô tả : In ra số người sống ở Location "Iloilo\_City,\_Philippines"

Ý tưởng : Tìm id của location "Iloilo\_City,\_Philippines", đặt là idx. Tìm id của những người đã sống ở idx, đặt là idy. Đếm số idx vừa tìm được.

Nội dung :

select (count(?idy) as ?s)

where {

?idx a class:Location;

prs:tên\_hiển\_thị "Iloilo\_City,\_Philippines" .

?idy a class:Person;

rel:sống\_ở ?idx .

}

### Truy vấn 6

Mô tả : In ra thông tin những nước có dân số thấp hơn 100000 và có GPD cao hơn 5000

Ý tưởng : Tìm tất cả các country, lưu lại dân số và GDP của các nước đó vào các biến ds và gdp. Dùng filter để lọc những nước thỏa mãn, đồng thời lọc những triple chứa thuộc tính.

Nội dung :

select ?s ?p ?o

where {

?s a class:Country;

prs:dân\_số ?ds;

prs:GDP ?gdp;

?p ?o.

filter (?ds < 100000 && ?gdp > 5000 && strstarts(str(?p),"http://btl/properties/"))

}

### Truy vấn 7

Mô tả : In ra những người tên bắt đầu bằng "Pen" và kết thúc là "Smith"

Ý tưởng : Tìm tất cả các thực thể người. Lưu lại tên của từng người. Dùng filter regex để lọc tất cả những người thỏa mãn yên cầu đề bài

Nội dung :

select ?s ?p ?o

where {

?s a class:Person;

?p ?o.

filter (?p = prs:tên\_hiển\_thị && regex(?o, "^(Pen)[\\w\\d\\s]\*(Smith)$"))

}

### Truy vấn 8

Mô tả : In ra thông tin 5 sự kiện đầu tiên tìm thấy bị hoãn vào 2001-09-28

Ý tưởng : Tìm id của tất cả các time 2001-09-28, lưu vào idx. Tìm id của tất cả những sự kiện bị hoãn vào idx vừa tìm được. Do đề bài yêu cầu 5 sự kiện nên ta đưa truy vấn trên vào 1 truy vấn con và đặt limit là 5. Sau truy vấn con ta đã có 5 id của những người thỏa mãn đề bài. Ta in ra tất cả thông tin của những id ấy. Dùng filter để lọc tất cả những triple lưu thuộc tính

Nội dung :

select ?s ?p ?o

where {

{

select ?s where {

?idx a class:Time;

prs:tên\_hiển\_thị "2001-09-28".

?s a class:Event;

rel:bị\_hoãn\_vào ?idx.

}

limit 5

}

?s ?p ?o.

filter (strstarts(str(?p), "http://btl/properties"))

}

### Truy vấn 9

Mô tả : In ra số lượng các sự kiện được lưu trong CSDL

Ý tưởng : Tìm id của tất cả những event được lưu trong CSDL và đếm chúng

Nội dung :

select (count(?idx) as ?s)

where {

?idx a class:Event.

}

### Truy vấn 10

Mô tả : In ra tên hiển thị của 500 người đầu tiên có số lượng relationship >= 13 cùng với số lượng relationship của người đó

Ý tưởng : Tìm tất cả các triple có chủ người là Person, lọc và lấy tất cả các triple chứa quan hệ. Group by theo id, và đếm số quan hệ của từng người, dùng Having để lọc và lấy những người có nhiều hơn hoặc bằng 13 quan hệ. Đặt limit là 500 và in ra tên hiển thị cùng số lượng relationship của người đó

Nội dung :

select ?s (count(?\_p) as ?p)

where {

?o a class:Person;

?\_p ?id;

prs:tên\_hiển\_thị ?s.

filter (strstarts(str(?\_p), "http://btl/relationship/"))

}

group by ?s

having (count(?\_p) >= 13)

LIMIT 500

## Truy vấn nâng cao



### Truy vấn 1

Mô tả : In ra những người có quốc tịch ở một nơi nhưng sống ở nơi khác

Ý tưởng : Tìm id của tất cả các country 2 lần để được 2 tập idx và idy. Tìm id của tất cả những người có quốc tịch ở idx và sống ở idy. Dùng filter để lọc và lấy những người mà có cặp idx và idx khác nhau.

Nội dung :

select ?s ?p ?o

where {

?idx a class:Country;

prs:tên\_hiển\_thị ?p.

?idy a class:Country;

prs:tên\_hiển\_thị ?o.

?idz a class:Person;

rel:﻿có\_quốc\_tịch ?idx;

rel:sống\_ở ?idy;

prs:tên\_hiển\_thị ?s.

filter(?p!=?o)

}



### Truy vấn 2

Mô tả : In ra tên các chương trình bị một nước tạm hoãn và đã gây thiệt hại lớn cho "Amazon.com"

Ý tưởng : Tìm id của tất cả các nước, gọi là idx. Tìm id của tất cả công ty mà có tên là “Amazon.com”, gọi là idz. Tìm id của tất cả các chương trình mà tạm hoãn bởi idx, gây thiệt hại lớn cho idz. In ra màn hình kết quả tìm được

Nội dung :

select ?s ?lk1 ?p ?lk2 ?o

where {

?idz a class:Organization;

prs:tên\_hiển\_thị ?o.

?idy a class:Event;

?lk2 ?idz;

prs:tên\_hiển\_thị ?p.

?idx a class:Country;

prs:tên\_hiển\_thị ?s;

?lk1 ?idy.

filter (?o = "Amazon.com" && ?lk1=rel:tạm\_hoãn && ?lk2 = rel:gây\_thiệt\_hải\_lớn\_cho)

}

### Truy vấn 3

Mô tả : In ra tên và ngày mất của những người đã đến cả nước "Madagascar" và "Togo", mất sau năm 1990

Ý tưởng : Tìm id của 2 nước "Madagascar" và "Togo", đặt là idx và idy. Tìm id của tất cả các Time,đặt là idt, lưu lại tên hiện thị của nó. Tìm id của tất cả các person mà đã đến cả idx, idy, mất tại idz. Dùng filter regex để lọc tất cả nhừng người thỏa mãn mất sau năm 1990.

Nội dung :

select ?s ?p ?o

where {

?idx a class:Country;

prs:tên\_hiển\_thị "Madagascar".

?idy a class:Country;

prs:tên\_hiển\_thị "Togo".

?idt a class:Time;

prs:tên\_hiển\_thị ?o.

?idz a class:Person;

rel:sống\_ở ?idy;

rel:sống\_ở ?idx;

prs:tên\_hiển\_thị ?s;

?p ?idt.

filter (?p = rel:mất\_ngày && regex(?o, "^((199[0..9])|2)") )

}

### Truy vấn 4

Mô tả : In ra những Event trong tên có từ Vietnam, được tổ chức đầu tiên tại "Australia" và diễn ra vào sau năm 2009

Ý tưởng : Tìm id của tất cả những country tên là "Australia", đặt là idx. Tìm id của tất cả những time, đặt là idy. Tìm tất cả event được tổ chức đâu tiên tại idx, diễn tra vào idy. Dùng filter regex để lọc tất cả những event trong tên có vietnam, những năm bắt đầu bằng 201 (>= 2010)

Nội dung :

select ?s ?lk1 ?p ?lk2 ?o

where {

?id a class:Event;

?lk1 ?idx;

?lk2 ?idy;

prs:tên\_hiển\_thị ?s.

?idx a class:Country;

prs:tên\_hiển\_thị ?p .

?idy a class:Time;

prs:tên\_hiển\_thị ?o.

filter (regex(?s, "Vietnam") && ?lk1 = rel:được\_tổ\_chức\_đầu\_tiên\_tại && ?lk2 = rel:diễn\_ra\_vào && ?p = "Australia" && strstarts(str(?o), "201"))

}

### Truy vấn 5

Mô tả : In ra tên của 20 nước có GDP cao nhất và GDP của nước đó

Ý tưởng : Tìm tất cả id của các nước, lưu lại dgp của nước đó. Dùng order by desc để sắp xếp gdp theo chiều giảm dần và in ra 20 nước đầu tiên cùng gdp của nước đó

Nội dung :

select ?s ?p ?o

where {

?id rdf:type class:Country.

?id prs:tên\_hiển\_thị ?s.

?id ?p ?o.

filter (?p = prs:GDP)

}

order by desc (?o)

limit 20

### Truy vấn 6

Mô tả : In ra mô tả của nơi tiếp giáp với "Wichita,\_United\_States", trực thuộc "Tainan,Taiwan", và là nời kết thúc của chùm sự kiện "Vietnam\_Medi-Pharm\_Expo\_(MediPharm\_Expo)"

Ý tưởng : Tìm tất cả các location có tên là "Wichita,\_United\_States", "Tainan,Taiwan", đặt là idy, idz. Tìm các event "Vietnam\_Medi-Pharm\_Expo\_(MediPharm\_Expo)" đặt là idt. Tìm tất cả các location tiếp giáp với idy, trực thuộc idz, là nơi kết thúc của idt. Dùng filter để lọc và lấy những triple chứa thuộc tính của location

Nội dung :

select ?s ?p ?o

where {

?idy rdf:type class:Location;

prs:tên\_hiển\_thị "Wichita,\_United\_States".

?idz rdf:type class:Location;

prs:tên\_hiển\_thị "Tainan,Taiwan".

?idt rdf:type class:Event;

prs:tên\_hiển\_thị "Vietnam\_Medi-Pharm\_Expo\_(MediPharm\_Expo)".

?idx rdf:type class:Location;

rel:tiếp\_giáp\_với ?idy;

rel:trực\_thuộc ?idz;

rel:là\_nơi\_kết\_thúc\_của ?idt;

prs:tên\_hiển\_thị ?s;

?p ?o.

filter (?p = prs:mô\_tả)

}

### Truy vấn 7

Mô tả : In ra thông tin 3 công ty đầu tiên tìm thấy mà là đối thủ của 2 công ty "Vodafone", "Yahoo!" nhưng không là đối thủ của "United\_Nations"

Ý tưởng : Tìm những công ty là đối thủ của "Vodafone", "Yahoo!", sau đó tìm những công ty là đối thủ của "United\_Nations". Dùng minus để lấy phần bù, ta sẽ được những công ty thỏa mãn đề bài. Do bài cần tìm 3 công ty nên ta đặt tất cả trong 1 sub query và dùng limit 3. Sau đó in ra thông tin bằng cách filter để lọc ra những triple chứa thuộc tính của công ty.

Nội dung :

select ?s ?p ?o

where {

{

select ?s where {

{

?s a class:Organization;

rel:là\_đối\_thủ\_của ?idx;

rel:là\_đối\_thủ\_của ?idy.

?idx a class:Organization;

prs:tên\_hiển\_thị "Vodafone".

?idy a class:Organization;

prs:tên\_hiển\_thị "Yahoo!".

}

minus

{

?s a class:Organization;

rel:là\_đối\_thủ\_của ?idz.

?idz a class:Organization;

prs:tên\_hiển\_thị "United\_Nations".

}

}

limit 3

}

?s ?p ?o.

filter (strstarts(str(?p), "http://btl/properties/"))

}

### Truy vấn 8

Mô tả : In ra 3 người có tuổi cao nhất cùng với tuổi của họ, yêu quý nước "Falkland\_Islands" nhưng lại sống ở location "Nicosia,\_Cyprus"

Ý tưởng : Tìm những người yêu quý "Falkland\_Islands" nhưng lại sống ở location "Nicosia,\_Cyprus" (tương tự các truy vấn trên). Sau đó order by desc theo tuổi của họ và lấy limit 3

Nội dung :

select ?s ?o

where {

?id a class:Person;

rel:yêu\_quý ?idx;

rel:sống\_ở ?idy;

prs:tên\_hiển\_thị ?s;

prs:tuổi ?o.

?idx a class:Country;

prs:tên\_hiển\_thị "Falkland\_Islands".

?idy a class:Location;

prs:tên\_hiển\_thị "Nicosia,\_Cyprus".

}

order by desc (?o)

limit 3

### Truy vấn 9

Mô tả : In ra tên của những người đã sống ở nhiều hơn 1 quốc gia nhưng lại tham gia ít hơn 3 event

Ý tưởng : Tìm tất cả những người đã sống ở nhiều hơn 1 quốc gia, dùng group by và having để đếm số quốc gia mà người đó sống. Tìm tất cả những người them gia ít hơn 3 event, dùng group by và having để đếm số event mà người đó tham gia. Sau đó dùng minus để trừ 2 tập đã tìm được.

Nội dung :

select ?s ?p ?o

where {

{

select ?s

where {

?s a class:Person;

rel:sống\_ở ?idx.

?idx a class:Country.

}

group by ?s

having (count(?idx) > 1)

}

minus

{

select ?s

where {

?s a class:Person;

rel:tham\_gia ?idx.

?idx a class:Event.

}

group by ?s

having (count(?idx) > 3)

}

?s ?p ?o.

filter ( ?p = prs:tên\_hiển\_thị )

}

### Truy vấn 10

Mô tả : In ra 3 nước đầu tiên tìm thấy cùng tham dự 1 sự kiện bất kỳ và cùng hợp tác với "Republic\_of\_the\_Congo", trong đó GDP của nước thứ nhất lớn hơn nước thứ hai nhưng dân số lại nhỏ hơn nước thứ hai

Ý tưởng : Tìm id của tất cả các sự kiện, đặt là k1. Tìm id của tất cả các nước tên là "Republic\_of\_the\_Congo",đặt là k2. Tìm tất cả các nước đồng thời tham dự k1 và hợp tác với k2. Dùng filter để lọc ra những nước có diện tích lớn hơn nhưng dân số lại nhỏ hơn. Lấy limit 3

Nội dung :

select ?s ?o

where {

?idx a class:Country;

rel:tham\_dự ?k1;

rel:hợp\_tác\_với ?k2;

prs:GDP ?gdp1;

prs:dân\_số ?ds1;

prs:tên\_hiển\_thị ?s.

?idy a class:Country;

rel:tham\_dự ?k1;

rel:hợp\_tác\_với ?k2;

prs:GDP ?gdp2;

prs:dân\_số ?ds2;

prs:tên\_hiển\_thị ?o.

?k1 a class:Event.

?k2 a class:Country;

prs:tên\_hiển\_thị "Republic\_of\_the\_Congo".

filter (?gdp1 > ?gdp2 && ?ds1 < ?ds2)

}

limit 3

## Kiểm thử hiệu năng

### Truy vấn cơ bản

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (N, M) | TVCB 1 | TVCB2 | TVCB3 | TVCB4 | TVCB5 | TVCB6 | TVCB 7 | TVCB8 | TVCB 9 | TVCB 10 |
| 100,200 | 13ms | 15ms | 14ms | 18ms | 19ms | 14ms | 14ms | 17ms | 18ms | 30ms |
| 5000,7000 | 14ms | 16ms | 14ms | 18ms | 20ms | 21ms | 20ms | 16ms | 24ms | 85ms |
| 60K,80K | 16ms | 20ms | 21ms | 19ms | 19ms | 18ms | 93ms | 17ms | 20ms | 828ms |
| 100K,200K | 23ms | 23ms | 20ms | 22ms | 20ms | 21ms | 178ms | 18ms | 20ms | 1410ms |
| 500k,1M | 33ms | 23ms | 13ms | 21ms | 19ms | 17ms | 720ms | 27ms | 21ms | 6882ms |

### Truy vấn nâng cao

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| (N, M) | TVNC1 | TVNC2 | TVNC3 | TVNC4 | TVNC5 | TVNC6 | TVNC7 | TVNC8 | TVNC9 | TVNC 10 |
| 100,200 | 20ms | 16ms | 22ms | 15ms | 19ms | 22ms | 19ms | 14ms | 44ms | 16ms |
| 5000,7000 | 17ms | 16ms | 17ms | 17ms | 15ms | 19ms | 16ms | 16ms | 42ms | 17ms |
| 60K,80K | 18ms | 20ms | 17ms | 18ms | 20ms | 18ms | 17ms | 15ms | 63ms | 13ms |
| 100K,200K | 26ms | 18ms | 19ms | 18ms | 22ms | 20ms | 17ms | 17ms | 75ms | 23ms |
| 500k,1M | 281ms | 32ms | 18ms | 27ms | 20ms | 21ms | 23ms | 17ms | 176ms | 33ms |

## Đánh giá công nghệ

### Ưu điểm

* Server cài đặt nhanh chóng.
* Giao diện server thân thiện, dễ sử dụng.
* Có thể dễ dàng import, export reposiotory.
* Server có tốc độ ổn định. Tuy so với một số server như Virtuoso còn chậm hơn, nhưng nhìn chung vẫn ở mức cao so với các server khác.
* Hỗ trợ đa ngôn ngữ, kết nối API dễ dàng.

### Nhược điểm

* Bản miễn phí bị giới hạn tối đa 5 triệu triples.
* Chỉ hỗ trợ hệ điều hành Linux.