Data Filter

Background

This module provides a mechanism for filtering persistent data on a configurable and customisable basis. In a nutshell it enforces **metadata-based access to users and/or roles**.

For instance, out of the box, the module supports location-based and privilege-based access control to patient records and their clinical data. The initial implementation for those two use cases is restricted to the patients and their visits, encounters and observations.

Architectural Overview

The data model has been kept simple deliberately such that only two concepts matter: **1)** which entity is being restricted by the filtering and **2)** on what basis does this filtering restriction operate?

1. The **basis** is the metadata on which the data access is *based*.
   Eg. `Program` for program-based access, `Location` for location-based access, ... etc.

2. The **entity** is *what* is authorised to access the data.
   Eg. `Role` or `User` .

This is all controlled and stored in a single table modelled by `EntityBasisMap` and can be configured through `DataFilterService` .

Examples

Let us go through a couple of filters that are baked into the module. This will help highlight how much of the design of filters follows systematic rules, and how much of it is implementation-specific.

1. Location-based access

When this filter is enabled, users are prevented access to data based on the location to which the data is associated. In this case

- the basis is `Location`,
- and the entity is `User` .

This raises a first question: **how do we determine which locations a user has access to?**
This is handled natively through the `EntityBasisMap`, users are mapped with their basis through this built-in table of Data Filter.
Of course every user can be mapped to multiple locations, it does not have to be just one. Those will be the only locations that user will get to *see,* and log into, those will be the user *bases*. Note that the location hierarchy is respected i.e. if a user is granted access to a location, then they will have access to all the data associated to all its child locations too.

Then a second question: **how do we associate the data to a location** so that the filter knows, when it is looking at some data, to which location it pertains?
This type of assumption is implementation-specific and in the case of the location-based filter that is bundled with the module, the assumption is that this is made at the patient level. There is therefore a need to associate a patient with a location, and this is done through a specific person attribute type. This person attribute type points to a location that marks the "patient's location".

Best is to look at one of the filter's conditions, see here:

Location-based access to visits

```
{
  "name": "datafilter_locationBasedVisitFilter",
  "targetClass": "org.openmrs.Visit",
  "condition": "patient_id IN (SELECT DISTINCT pa.person_id FROM person_attribute pa WHERE pa.
person_attribute_type_id = :attributeTypeId AND pa.value IN (:basisIds) AND pa.voided = 0)",
  "parameters": [
    {
      "name": "attributeTypeId",
      "type": "integer"
    },
    {
      "name": "basisIds",
      "type": "string"
    }
  ]
}
```

The condition reads *include the visits for patients that have a specific "location" attribute type whose value is either of the bases (=locations)*. Of course almost identical filters are defined for patients, encounters and observations.
The logic here is that every piece of data (whether visit, encounter, observation or patient itself) refers to a `patiend_id` that is subject to the above filter condition. If the parameter takes an array or collection as a value of the type field of the parameter must be set to that of the elements themselves and not the collection type. Keep in mind that if it is an array parameter value, the value you set for the parameter should be an Object array. E.g. if we wanted to set the value of the basisIds parameter which happens to be an array of strings, you pass in new `new Object[]{"my string"}` and NOT `new String []{"my string"}`.

This leads to a last question: **how do we know which person attribute contains the value of the alocation to which a patient pertains?**
There is an in-built mechanism in Data Filter that helps configure such implementation-specific linkages between data and filtering bases. A global property `"datafilter.personAttributeTypeUuids"` exists to contain a list of UUIDs of person attribute types. Each of the person attribute types in that list is assumed to have a different *format*. For instance only one of the person attribute types listed in the global property can be of the format `Location` . In the filter definition above this means that `attributeTypeId` is of a type whose format is `Location` because that is the type of the bases. And in the context of location-based filtering the bases type is `Location` .
See here for how this specific logic is implemented for the location-based filters.

Note that in case of the built in location based filtering, any patients that are not associated to any locations, their records and any filtered clinical data will be invisible to all users except for privileged ones that by pass all filtering or if the filters are disabled, see the section below about disabling data filtering and how it works.

Disabling data filtering

The data filter module has a built in mechanism to by pass all the filters or a single filter.

Bypassing all filters

The module is implemented to skip all kinds of filtering when any of the following is true,

- The the code is being executed in a DaemonThread
- When the authenticated user has a super user role
- When the authenticated user has the **ByPass Data Filters** privilege.

Bypassing a single filter

The module has a built in global property based approach to disable a single filter for all users, the way it works is that you add a global property with a name that matches the filter's name with a **.disabled** suffix and set it's value to true. e.g if you want to disable location based filtering of visits using the example above, the global property name would be **datafilter_locationBasedVisitFilter.disabled**. Note that this behavior if going to be changed so that you instead disable filtering on class and not filter name basis.