JDBC Example Script

Your first Harvest

JDBCFetch allows you to ingest data from a database into VIVO. Unlike pubmed which is a national data source, the translation file for your ingest will need to be created by you. In addition, although a standard workflow exists in the example-script your ingest may be different. For the example-jdbc all of this work has been done for you. You'll notice that the jdbcfetch.config.xml file contains the select queries to harvest the data from demodb, and the jdbc-to-vivo.datamap.xsl file contains a datamap to map this data into the VIVO ontology. For your own harvest, you'll need to modify both of these files so your own data is harvested. Finally, you'll need to make sure databaseclone.config.xml contains the proper connection parameters to connect to your local database.

So, let's walkthru running the example-jdbc script. First, we'll talk about the data we are harvesting.

The Sample Database

- The sample HR database contains 5 tables: department, job, person, type, and user.
    - The person table stores information about people affiliated with the university, such as their names, phone, fax, preferred email, title, and name.
        - The person table also contains a flag that determines whether or not to publish the information
    - The department table stores very basic information about the organizational structure of this fictional university. Each organization can have an id, a name, a type, and a super organization.
        - There is no super_dept_id (super organization id) value for the university (amusingly named the University of Sample Data) since it is mapped as a top level org
    - Lets look at the type table. Note that the type table type_id value matches the department table type_id.
        - Each type has an id, a value (the label for that type), and can have a super type (note that in this table, top level types use 0 as their super type)
    - The user table simply contains the person_id, the person's login name, and whether this account is expired or not
    - The job table stores information about the current jobs that people hold at the university. Each record is a job, linking a department and a person (by their ids) and stores the type code (from type table) and start date.

Now, let's setup the harvester to harvest this data.

Harvest Setup

- Change directory to example-scripts/example-jdbc
- Load the demodb.sql file into your mysql database

```
mysql -u root -p (if your running on the VM, the default root password is vitro123)
create database if not exists demodb;
GRANT ALL PRIVILEGES ON demodb.* TO demodb@localhost IDENTIFIED BY 'demodb';
exit;
```

- Now the database has been created and a user called demodb has been created that can connect to it. Next, we'll load the demodb.sql to the new database.

```
mysql -u demodb -p demodb < demodb.sql
```

- When prompted enter your demodb user password, which we set above to demodb.
- Edit the vivo.model.xml file
    - Set the dbURL, dbUser, dbPass, and Namespace
    - For more information on these parameters and their use, please see Harvester vivo configuration file
- Edit changenamespace-departments.config.xml, changenamespace-people.config.xml, changenamespace-positions.config.xml files and set the namespace parameters in each one to be your vivo namespace
    - For more information on these parameters and their use, please see ChangeNamespace
- Edit the run-jdbc.sh and remove-last-jdbc-harvest.sh files and set the HARVESTER_INSTALL_DIR= to be the directory you unpacked the harvester in
- Please refer to http://issues.library.cornell.edu/browse/VIVOHARV-124 regarding a documented issue in databaseclone.config.xml
- Run bash run-jdbc.sh
- Restart tomcat and apache2. You may also need to force the index to rebuild to see the new data. The index can be rebuilt by issuing the following URL in a browser:http://your.vivo.address/vivo/SearchIndex. This will require site admin permission, and prompt you to login if your not already.

The first run

Three folders will be created

- logs
- data
- previous-harvest

The logs folder contains the log from the run, the data folder contains the data from each run, and the previous-harvest folder contains the old harvest data for use during the update process at the end of the script. While your testing, I would recommend treating each run as the first run (so no update logic will occur). You can do this by removing
the previous-harvest folder before running again.

Inside the data folder, you will find the raw records utilized during the ingest. To see what rdf statements went into VIVO, you can view the vivo-additions.rdf.xml file. Conversely, to view what the harvester removed (because of updated data), you can view the vivo-subtractions.rdf.xml file. This file will be blank on your first run, since you have no previous harvest to compare the incoming data against.

Optimizing

Once your ready to run a large dataset, it is advisable to the record storage from files to a database. Although this will make it harder to find individual records, speed and performance will be increased during the fetch and translate stage. To do so:

- Edit the nano raw-records.config.xml to use TDB, which is a semantic data store

```
<RecordHandler>
        <Param name="rhClass">org.vivoweb.harvester.util.repo.JenaRecordHandler</Param>
        <Param name="type">tdb</Param>
        <Param name="dbDir">data/raw-records</Param>
</RecordHandler>
```

- Edit the translated-records.config.xml to use TDB, which is a semantic data store

```
<RecordHandler>
        <Param name="rhClass">org.vivoweb.harvester.util.repo.JenaRecordHandler</Param>
        <Param name="type">tdb</Param>
        <Param name="dbDir">data/translated-records</Param>
</RecordHandler>
```