JDBCFetch

JDBCFetch

Gathers information from a database and places it within a given RecordHandler organized according to how the data is arranged within the chosen database.

Reason for use

Connection with different types of databases is essential to effectively populating a VIVO instance with viable data. Java Database Connectivity is available and flexible, while allowing standard SQL queries to retrieve the information from the chosen database.

Parameters

The parameters for the tool can be placed in the specified config file.

wordiness

wordiness - (optional) sets the lowest level of log messages to be displayed to the console. The lower the log level, the more detailed the messages. Possible Values:

- <Param name="wordiness">OFF</Param> - Results in no messages being displayed.
- <Param name="wordiness">ERROR</Param> - Results in only messages from the ERROR level to be displayed.
    - Error messages detail when the tool has experienced an error preventing it from completing its task
- <Param name="wordiness">WARN</Param> - Results in only messages above and including WARN level messages to be displayed. Match does not produce any WARN level messages.
- <Param name="wordiness">INFO</Param> - (Default) Results in all messages above and including INFO level messages to be displayed. INFO level messages detail when the tool has started and ended and when it begins/ends a phase ('Finding matches' and 'Beginning Rename of matches') and how many matches have been found.
- <Param name="wordiness">DEBUG</Param> - Results in all messages above and including DEBUG level messages to be displayed. DEBUG level messages detail each matching input URI to its VIVO URI as they are processed. Additionally, it will display stacktrace information if an error occurs.
- <Param name="wordiness">ALL</Param> or <Param name="wordiness">TRACE</Param> - Results in all messages above and including TRACE level messages to be displayed, since trace is the lowest level it is the same as ALL in practice. TRACE level messages details every matching set as it is processed in each phase along with SPARQL queries and start and stop for their execution.

JDBCFetch Specific messages

Info:

- The number of records added
- The start of the run
- Command line usage

Debug:

- Finding data column names for each table
- Finding relation column names for each table
- Finding id column names for each table

Trace:

- All column names found
- Generated or user defined query
- Each record being added

relational database

The proper information for the connection to the relational database

- <Param name="driver"> - A JDBC driver is a java class which handles the interface between the program and a given database.The driver needs to exist within the classpath for the program to be able to use it.
    - EXAMPLES:
        - <Param name="driver">org.h2.Driver</Param>
- <Param name="connection"> - JDBC uses a connection string related to the Driver being used. It is in the general format "jdbc:somejdbcvendor:other data needed"
    - EXAMPLES :

        - Microsoft SQL server : <Param name="connection">jdbc:jtds:sqlserver://127.0.0.1:8080/databasename</Param>
        - H2 database (http://www.h2database.com) : <Param name="connection">jdbc:h2:directory/location</Param>

- <Param name="username"> - A valid login with proper permissions
- <Param name="password"> - The associated secure password
    - These are what the fetch uses when accessing the given database. These must be valid otherwise the harvest run will not have access to the database. By default we use "sa" for system administrator and a blank password.

The database table information

- <Param name="tableName"> - Each table has an identifying name associated with it. These names are case sensitive and should be checked several times before the first run.
- <Param name="id"> - To distinguish records from each other this should be a distinct non-null field. If such a field is not available, the configuration supports forming a concatenation of several fields (aka a composite key). The concatenation is used by supplying a comma separated list of field names rather than a single field name.
- <Param name="query"> - Within this tag is placed an SQL query which will be ran on the supplied database table. The names attributed during this query will supply the names applied to the data within the record handler.
- <Param name="fields"> This parameter signifies which fields to harvest. If a field is found on this list it must exist in the associated table. Make sure the table name is provided before the equals sign.
- <Param name="validTableType"> This parameter is used to determine which types of tables are to be harvester from
    - Possible values (If the parameter is not present it uses the default value):
        - TABLE - If set to this then the JDBCFetch is expecting a table {DEFAULT}
        - VIEW - this setting expects a view generated from the database.

Overview

JDBCFetch is used to ingest data from RDB/JDBC interface. Brings in data from relational database sources defined by the configuration file and converts them to XML, most likely within a RecordHandler.

Short Option	Long Option	Parameter Value Map	Description	Required
d	driver	JDBC_DRIVER	jdbc driver class	true
c	connection	JDBC_CONN	jdbc connection string	true
u	username	USERNAME	database username	true
p	password	PASSWORD	database password	true
o	output	CONFIG_FILE	config file for output record handler	true
O	outputOverride	VALUE	override the RH_PARAM of output record handler using VALUE	false
t	tableName	TABLE_NAME	a single database table name [have multiple -t for more table names]	false
Q	query	SQL_QUERY	use SQL_QUERY to select from TABLE_NAME	false
I	id	ID_FIELD_LIST	use columns in ID_FIELD_LIST [comma separated] as identifier for TABLE_NAME	false
F	fields	FIELD_LIST	fetch columns in FIELD_LIST [comma separated] for TABLE_NAME	false
R	relations	RELATION_PAIR_LIST	fetch columns in RELATION_PAIR_LIST [comma separated] for TABLE_NAME	false
W	whereClause	CLAUSE_LIST	filter TABLE_NAME records based on conditions in CLAUSE_LIST [comma separated]	false
T	tableFromClause	TABLE_LIST	add tables to use in from clauses for TABLE_NAME	false
	delimiterPrefix	DELIMITER	Prefix each field in the query with this character	false
	delimiterSuffix	DELIMITER	Suffix each field in the query with this character	false

Usage

JDBCFetch is often the first part of a harvest of data from a standard relational database. It pulls the information into a local RecordHandler which can then be Translated before being transfered into a jena model.

Define Alias

```
JDBCFetch="java $OPTS ~~Xms$MIN_MEM -Xmx$MAX_MEM -Dharvester.task=$HARVESTER_TASK -Dprocess.task=JDBCFetch -cp bin/harvester~~$VERSION.jar:bin/dependency/* org.vivoweb.harvester.fetch.JDBCFetch"
```

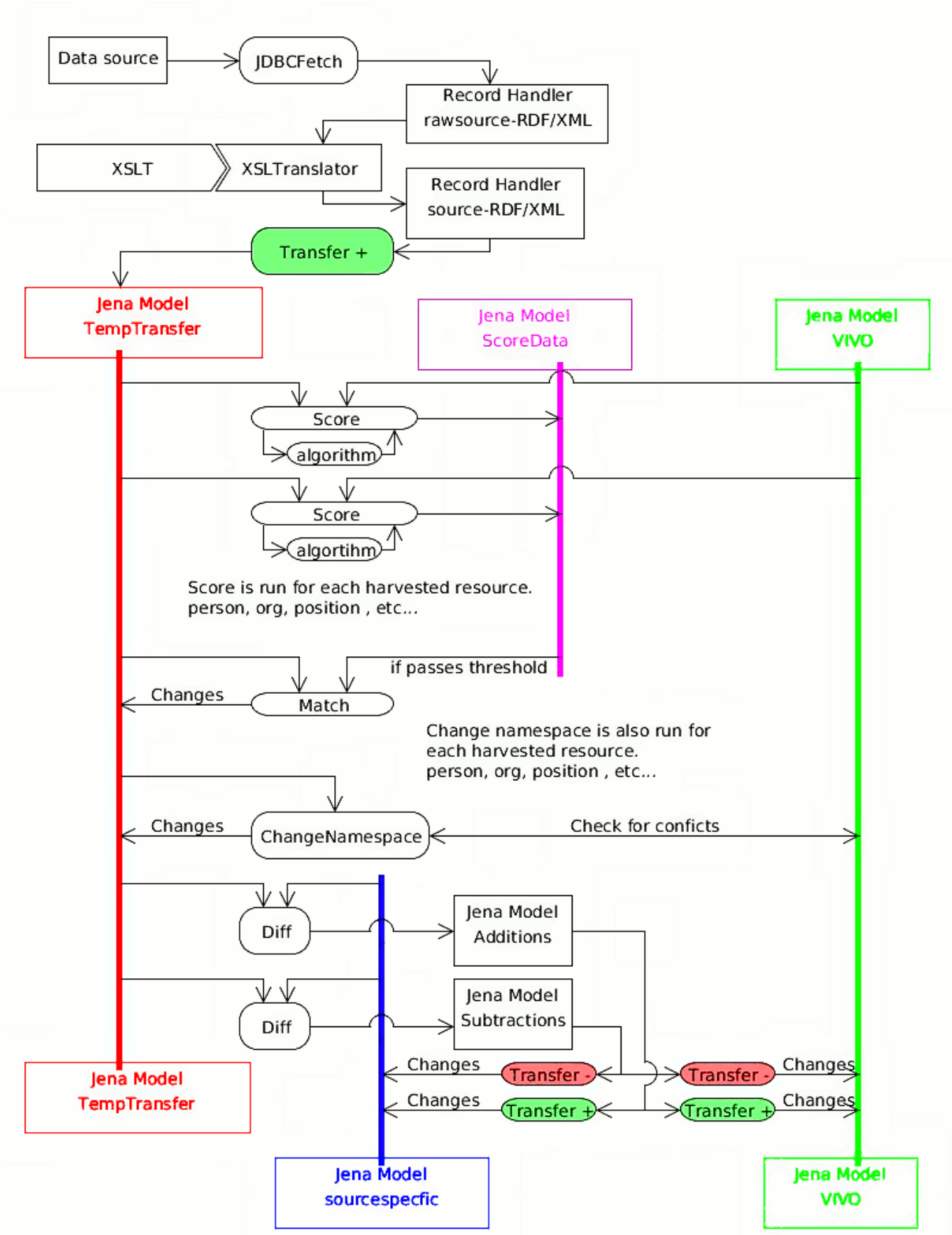
Invocation

```
$JDBCFetch -X config/tasks/DSR-JDBCFetch.xml
```

Configuration file example

```
<?xml version="1.0" encoding="UTF-8"?>
<Task type="org.vivoweb.ingest.fetch.JDBCFetch">
  <Param id="driver">com.mysql.jdbc.Driver</Param>
  <Param id="connection">jdbc:mysql://127.0.0.1:3306/jdbcdemoharvest</Param>
  <Param id="username">jdbcDemoHarvest</Param>
  <Param id="password">EFaY6nSxBNpL7cYb</Param>
  <Param id="output">config/recordHandlers/JDBCXMLRecordHandler.xml</Param>
</Task>
```

Flowchart


Data source
JDBCFetch
Record Handler
rawsource-RDF/XML
XSLT
XSLTranslator
Record Handler
source-RDF/XML
Transfer +
Jena Model
TempTransfer
Jena Model
ScoreData
Jena Model
VIVO
Score
algorithm
Score
algortihm
Score is run for each harvested resource.
person, org, position , etc...
if passes threshold
Changes
Match
Change namespace is also run for
each harvested resource.
person, org, position , etc...
Changes
ChangeNamespace
Check for conficts
Diff
Jena Model
Additions
Diff
Jena Model
Subtractions
Changes
Transfer -
Transfer -
Changes
Changes
Transfer +
Transfer +
Changes
Jena Model
TempTransfer
Jena Model
sourcespecfic
Jena Model
VIVO

Methods

getParser

1. parse the arguments from the parameter list above

buildSelect

1. start a stringbuilder
2. append and assemble the "select" part of the statement
3. append the relations
4. append the id fields of the table
5. append the "from" clauses
6. end query string with "where" clauses

execute

1. iterate over table names
    a. buildSelect assembles a SQL select statement string specific for each table
    b. executeQuery on table with the SQL string using the SQL Statement object
    c. Make an RDF/XML record for each result of the query

main

1. Start Logger
2. Run JDBCFetch.execute passing the args[] to the constructor
3. catch errors
    a. IllegalArgumentException
    b. IOException
    c. Exception