GenQuery2: A more standardized, powerful parser for the iRODS namespace

Kory Draughn
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
korydraughn@renci.org

Terrell Russell
Renaissance Computing Institute (RENCI)
UNC Chapel Hill
unc@terrellrussell.com

ABSTRACT

The iRODS GenQuery interface has long defined the way users and administrators can search the iRODS namespace, its storage systems, users, and metadata, while honoring the iRODS permission model. The next generation of GenQuery, GenQuery2, is now available for experimentation. However, there is still a lot of work to do. This paper will cover its expanded syntax and capabilities and what is to come.

Keywords

iRODS, general query, genquery, parser, SQL

PURPOSE

GenQuery2 is an experimental redesign (and implementation) of the iRODS GenQuery parser. Written in flex and bison, this new implementation provides full and clear control over the domain-specific language (DSL) that is used to power many of the internal processes within iRODS.

This project exists as a means for allowing the iRODS community to test the implementation and provide feedback so that the iRODS Consortium can produce a GenQuery parser that is easy to understand, maintain, and enhance all while providing a syntax that mirrors standard SQL as much as possible.

Once stable, the code will be merged into the iRODS server making it available within and alongside future releases of iRODS.

The current source can be found at: `https://github.com/irods/irods_api_plugin_genquery2`

The repository contains all source code for generating a package which includes the following three binaries: an API plugin, a rule engine plugin, and a new iCommand. Everything discussed in this paper can be found in the repository.

GENERAL FEATURES

The features of this new parser are listed here. Many of these are new or correctly supported in iRODS for the first time. They represent the vast majority of requested new features and bug fixes for GenQuery1 from the iRODS Community over the last decade and provide a flexible and powerful new way to interact with the iRODS Catalog.

- Enforces the iRODS permission model
- Logical AND, OR, and NOT
- Grouping via parentheses

iRODS UGM 2023 June 13-16, 2023, Chapel Hill, NC

- SQL CAST
- SQL GROUP BY
- SQL aggregate functions (e.g. count, sum, avg, etc)
- Per-column sorting via ORDER BY [ASC|DESC]
- SQL FETCH FIRST N ROWS ONLY (LIMIT offered as an alias)
- Metadata queries involving different iRODS entities (i.e. data objects, collections, users, and resources)
- Operators: =, !=, <, <=, >, >=, LIKE, BETWEEN, IS [NOT] NULL
- SQL keywords are case-insensitive
- Federation is supported and backwards compatible

COMPONENTS AND EXAMPLES

To provide this new parser to the different parts of iRODS, three components have been designed and implemented. These include an API plugin, a rule engine plugin, and a new iCommand, namely **iquery**.

API Plugin

The API Plugin wraps the parser and makes it available to all iRODS clients. It has a new API number of **1000001**, which may change in the future, and the following input parameters:

- `query_string` - The GenQuery2 string.
- `zone` - The name of the zone where the query should be executed.
- `sql_only` - An integer instructing the plugin to return the generated SQL, without execution.

This API Plugin returns a JSON string representing the `resultset` on success. On failure, it returns an iRODS error code.

At the moment, this API Plugin defaults to returning a maximum of 16 rows if the client does not specify the number of rows to return. This has not yet been tested heavily and will probably change with additional community input.

Rule Engine Plugin

The provided Rule Engine Plugin makes GenQuery2 available to the iRODS Rule Language and other rule engine plugins via four rules of its own:

- `genquery2_execute(*handle, *query_string)`
- `genquery2_next_row(*handle)`
- `genquery2_column(*handle, *index, *value)`
- `genquery2_destroy(*handle)`

These rules can be called in succession to execute a query, walk the results, and then clean up the memory allocations.

Example

The rules can be enabled by adding the following rule engine instance information to the `rule_engines` stanza of `server_config.json`.

```
{
    "instance_name": "irods_rule_engine-genquery2-instance",
    "plugin_name": "irods_rule_engine-genquery2",
    "plugin_specific_configuration": {}
}
```

Then, an example rule could look like the following:

```
genquery2_test_rule()
{
    # Execute a query. The results are stored in the Rule Engine Plugin.
    genquery2_execute(*handle, "select COLL_NAME, DATA_NAME order by DATA_NAME desc limit 1");

    # Iterate over the resutls.
    while (errorcode(genquery2_next_row(*handle)) == 0) {
        genquery2_column(*handle, '0', *coll_name); # Copy the COLL_NAME into *coll_name.
        genquery2_column(*handle, '1', *data_name); # Copy the DATA_NAME into *data_name.
        writeLine("stdout", "logical path => [*coll_name/*data_name]");
    }

    # Free any allocated resources. This is also handled automatically when the agent is shut down.
    genquery2_destroy(*handle);
}
```

This rule selects a single row of `COLL_NAME` and `DATA_NAME`, ordered by descending `DATA_NAME`, and then writes a constructed full logical path to `stdout`.

iCommand

The last component provided by this project is a new iCommand client which uses the new API Plugin. This new binary, `iquery`, enables execution of GenQuery2 queries against the iRODS Catalog from the command line.

The current help text:

```
Usage: iquery [OPTION]... QUERY_STRING

Queries the iRODS Catalog using GenQuery2.

QUERY_STRING is expected to be a string matching the GenQuery2 syntax. Failing
to meet this requirement will result in an error.

Mandatory arguments to long options are mandatory for short options too.

Options:
      --sql-only        Print the SQL generated by the parser. The generated
```

```
                        SQL will not be executed.
  -z, --zone=ZONE_NAME  The name of the zone to run the query against. Defaults
                        to the local zone.
  -h, --help            Display this help message and exit.

iRODS Version 4.3.0                iquery (experimental)
```

Examples

To list the number of replicas for all data objects, the following `iquery` command can be run. `jq` is used to cleanly format the resulting JSON received on `stdout`. There are two rows returned, with three columns each. These represent the collection name, data object name, and the count of how many times the compound grouping was found, in this case, 3 and 1 times, respectively.

```
$ iquery "select COLL_NAME, DATA_NAME, count(DATA_ID) group by COLL_NAME, DATA_NAME" | jq
[
  [
    "/tempZone/home/rods",
    "foo",
    "3"
  ],
  [
    "/tempZone/home/rods",
    "bar",
    "1"
  ]
]
```

The next example shows the SQL generated by the parser for the same query, but without executing that SQL. This time, `pg_format` is used to format the returned `stdout`.

```
$ iquery --sql-only \
     "select COLL_NAME, DATA_NAME, count(DATA_ID) group by COLL_NAME, DATA_NAME" | \
     pg_format -

SELECT DISTINCT
    t0.coll_name,
    t1.data_name,
    count(t1.data_id)
FROM
    R_COLL_MAIN t0
    INNER JOIN R_DATA_MAIN t1 ON t0.coll_id = t1.coll_id
    INNER JOIN R_OBJT_ACCESS pdoa ON t1.data_id = pdoa.object_id
    INNER JOIN R_TOKN_MAIN pdt ON pdoa.access_type_id = pdt.token_id
    INNER JOIN R_USER_MAIN pdu ON pdoa.user_id = pdu.user_id
    INNER JOIN R_OBJT_ACCESS pcoa ON t0.coll_id = pcoa.object_id
    INNER JOIN R_TOKN_MAIN pct ON pcoa.access_type_id = pct.token_id
    INNER JOIN R_USER_MAIN pcu ON pcoa.user_id = pcu.user_id
WHERE
    pdu.user_name = ?
    AND pcu.user_name = ?
    AND pdoa.access_type_id >= 1050
    AND pcoa.access_type_id >= 1050
GROUP BY
    t0.coll_name,
    t1.data_name FETCH FIRST 16 ROWS ONLY
```

REMAINING WORK

This project is relatively complete and fulfills many of the desired features, but there are a few items that must be resolved before making GenQuery2 available as part of the default iRODS server. The first is working to clean up the `CMakeLists.txt` file to be more in line with our recent efforts to modernize and modularize our approach to CMake across all our repositories. Second, this repository needs many tests to be considered ready for release and any kind of production use. Third, there is an open discussion about how a new parser should handle interacting with groups (and their constituent users) and tickets (and their permission model).

FUTURE PLANS

The future is GenQuery2, but we are not there yet. There are additional features that need to be considered and carefully implemented to follow the rest of the iRODS permission model, architecture, and design decisions. This list represents a snapshot in time in June 2023:

- Expose more SQL features
  - CASE, HAVING clauses
  - Sub-selects
  - Multi-argument functions
- Consider controlling various options through GenQuery2 syntax
  - e.g. iquery ”option distinct off; select DATA_NAME”
- Consider switching from `boost::variant` to `std::variant`
- Simplify pagination
  - Provide a utility library that manages the page information
  - Provide a document explaining how the utility may be implemented

COMMUNITY ENGAGEMENT

At this time, we are considering the idea of releasing GenQuery2 as an experimental package to be installable alongside an existing iRODS instance. This approach allows the community to try GenQuery2 and provide feedback and allows frequent updates that are not tied to a particular server release. We are seeking input and testing and look forward to learning what additional use cases this new parser can solve.