Package 'nJira'

March 2, 2020

Type Package

Version 0.1.1

Title SQL Like Query Interface for 'Jira'

Description SQL like query interface to fetch data from any 'Jira' installation. The data is fetched using 'Jira' REST API, which can be found at the following URL: <https://developer.atlassian.com/cloud/jira/platform/rest/v2>.

Author Nikhil Choudhry [aut, cre]

Maintainer Nikhil Choudhry <nikhil.choudhry@gmail.com>

License MIT + file LICENSE

RoxygenNote 7.0.2

Encoding UTF-8

LazyData TRUE

NeedsCompilation no

Imports httr, rjson, plyr

Repository CRAN

Date/Publication 2020-03-02 11:50:03 UTC

R topics documented:

jira.login *Jira Login Function*

Description

Authenticates the user to fetch data from the respective Jira installation.

Usage

```
jira.login(jira.env = NULL, jira.user = NULL, jira.pwd = NULL,
  jira.val = 0, logs = FALSE)
```

Arguments

jira.env	Web address of 'Jira' environment (e.g. https://issues.apache.org/jira)
jira.user	Jira User Name
jira.pwd	Jira Password
jira.val	0/1 how should the list values be returned in the query results
logs	debug logs required on not (Default = FALSE)

Value

The function authenticates the user into Jira installation and caches the Jira credentials.

Examples

```
jira.login(jira.env="https://issues.apache.org/jira",
jira.user="jiraTestUser", jira.pwd="jiraTestPwd")
```

jira.metadata *Jira Tables and Field Details*

Description

Returns the 'metadata' of Jira which includes 'table' and 'field' names, valid for respective Jira installation. These table and field names can be referred while creating a Jira Query.

Usage

```
jira.metadata(table = NULL, fields = NULL)
```

Arguments

table	Name of the Jira tables. If not specified, all the tables of the given interface are returned.
fields	List of field names whose details are required. If not specified, all the fields of the specified tables are returned.

Value

Data frame of Jira tables and field names.

Examples

```
fields <- jira.metadata()
fields <- jira.metadata(table = "history")
fields <- jira.metadata(table = "issues")
fields <- jira.metadata(table = "issues", fields = c("Created", "Date Required", "Dev Status"))
```

---

jira.query *Jira Query Interface*

Description

Query Jira using SQL like query syntax. The query response from Jira REST API is returned as a dataframe.

Usage

```
jira.query(table, fields = NULL, where = NULL, groupby = NULL)
```

Arguments

table	Name of Jira table from which data will be fetched.
fields	Comma separated names of the fields from the specified table whose values will be fetched.
where	specifies the where clause of the query. You can also pass your Jira JQL as-is in the where clause.
groupby	specifies the list of fields on which the data is grouped.

Details

For querying Jira 'history' table, the where clause must specify issue 'id'
Example : `where = "id = 'HIVE-22692'"`

Value

Data frame of results returned by the Jira query.

Examples

```
issues <- jira.query(table = "issues", fields = "id AS IssueId, Created, Status, Priority",
where = "project = 'HIVE' AND created >= '2019-01-01' AND created <= '2019-12-31' AND
Status IN ('Open', 'Closed', 'Resolved')")

issues <- jira.query(table = "issues", fields = "id AS IssueId, Created",
where = "'cf[10021]' = 'ABCD' AND Created > '2019-01-01'")

history <- jira.query(table = "history", where = "id = 'HIVE-22692'")

history <- jira.query(table = "history", fields = "id AS IssueId, toString AS Status,
COUNT(fromString) AS Count", where = "id = 'HIVE-22692' AND field = 'status'",
groupby = "id,toString")
```

---

rk.fields *Process Fields Clause*

Description

The function parses the fields clause and returns the modified string as per the specified mode. The fields clause supported format is represented by the following BNF:

```
<field.list> := <field.expr> ( DELIMIT.COMMA <field.expr> ) *
<field.expr> := ( FIELD.NAME | <aggr.func> LEFT.PAREN FIELD.NAME RIGHT.PAREN ) [ AS.ALIAS FIELD.NAME ]
<aggr.func> := FUNC.MIN | FUNC.MEDIAN | FUNC.AVG | FUNC.MAX | FUNC.COUNT | FUNC.SUM
```

Usage

```
rk.fields(fields, mode = "@")
```

Arguments

fields	clause following simplified sql syntax.
mode	specifies the parsing logic. The default value '@' returns the field list in perfmeter query format. The '+' value returns a field list used for grouping the dataframe with alias names. The '=' value returns a field list used for grouping the dataframe with original names. The '*' value returns the alias list used for renaming the columns. Any other value returns a field list used for selecting columns from a dataframe.

Value

The function returns the processed fields clause.

rk.groupby *Process GroupBy Clause*

Description

The function parses the groupby clause and returns the modified string as per the specified mode.

Usage

```
rk.groupby(groupby = NULL, mode = "@")
```

Arguments

groupby	clause following simplified sql syntax.
mode	specifies the parsing logic. The default value '@' returns the groupby clause in perfmeter format. Any other value returns the groupby fields used for aggregation.

Value

The function returns the processed groupby clause.

rk.metadata *Processing Meta Data*

Description

The function returns the list of tables, fields, and their descriptions.

Usage

```
rk.metadata(table = NULL, fields = NULL, gettabs, getflds,
  infofile = NULL)
```

Arguments

table	name of the interface tables. If not specified, all the tables of the given interface are returned.
fields	is the list of field names whose details are required. If not specified, all the fields of the specified tables are returned.
gettabs	is a function that returns the list of all the tables for the given interface.
getflds	is a function that returns the list of all the fields for the given table.
infofile	is the name of the file containing information about different tables and fields of the given interface.

Value

The function returns the resulting data frame.

`rk.query` *Data Processing Query*

Description

The function applies the given fields, where clause, and group by fields on the specified data frame.

Usage

```
rk.query(dframe, fields = NULL, where = NULL, groupby = NULL)
```

Arguments

`dframe`	data frame to be processed.
`fields`	fields to be filtered.
`where`	clause applied on the data.
`groupby`	used to aggregate the fields.

Value

The function returns the resulting data frame.

`rk.where` *Process Where Clause*

Description

The function parses the where clause and returns the modified string as per the specified mode. The where clause supported format is represented by the following BNF:

```
<where.cond> := <where.and> [ LOGICAL.OR <where.cond> ]
<where.and> := <where.not> [ LOGICAL.AND <where.and> ]
<where.not> := [ LOGICAL.NOT ] <where.clause>
<where.clause> := LEFT.PAREN <where.cond> RIGHT.PAREN | <where.expr>
<where.expr> := ( IDENTIFIER | QUOTE.STR ) ( [ LOGICAL.NOT ] ( OPERATOR.IN <value.list> | OPERATOR.LIKE <
<logic.cond> := ( EQUAL.TO | NOT.EQUAL | LESS.THAN | GREATER.THAN | LESS.EQUAL | GREATER.EQUAL ) <value.
<value.list> := LEFT.PAREN <value.const> ( DELIMIT.COMMA <value.const> ) * RIGHT.PAREN
<value.const> := | QUOTE.STR | NUMBER
```

Usage

```
rk.where(where = NULL, mode = "@", fields = NULL)
```

Arguments

`where`	clause following simplified sql syntax.
`mode`	specifies the parsing logic. The default value '@' returns the where clause in perfmeter format. The '=' value returns the where clause in IOD format. The '~' value returns the where clause in Jira format. The '' (empty string) value returns a where clause used with a sql statement. If a dataframe name is passed, the function returns the where clause for use with a dataframe.
`fields`	fields to be filtered.

Value

The function returns the processed where clause.

Index