Package 'ResultModelManager'

December 2, 2022

Title Result Model Manager (RMM) for OHDSI packages

Version 0.1.1

Description Database data model management utilities for OHDSI packages.

License Apache License

Encoding UTF-8

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.1

Depends R6,
DatabaseConnector (>= 5.0.0)

Imports SqlRender,
ParallelLogger,
checkmate,
DBI,
pool

Suggests testthat (>= 3.0.0),
RSQLite,
withr,
knitr,
rmarkdown

Config/testthat/edition 3

R topics documented:

ConnectionHandler *ConnectionHandler*

Description

Class for handling DatabaseConnector:connection objects with consistent R6 interfaces for pooled and non-pooled connections. Allows a connection to cleanly be opened and closed and stored within class/object variables

Value

DatabaseConnector Connection instance close Connection

boolean TRUE if connection is valid queryDb

boolean TRUE if connection is valid executeSql

Public fields

`connectionDetails` DatabaseConnector connectionDetails object

`con` DatabaseConnector connection object

`isActive` Is connection active or not#'

`snakeCaseToCamelCase` (Optional) Boolean. return the results columns in camel case (default)

Methods

Public methods:

- `ConnectionHandler$new()`
- `ConnectionHandler$renderTranslateSql()`
- `ConnectionHandler$initConnection()`
- `ConnectionHandler$getConnection()`
- `ConnectionHandler$closeConnection()`
- `ConnectionHandler$finalize()`
- `ConnectionHandler$dbIsValid()`
- `ConnectionHandler$queryDb()`
- `ConnectionHandler$executeSql()`
- `ConnectionHandler$queryFunction()`
- `ConnectionHandler$executeFunction()`
- `ConnectionHandler$clone()`

Method `new()`:

Usage:

```
ConnectionHandler$new(
  connectionDetails,
  loadConnection = TRUE,
  snakeCaseToCamelCase = TRUE
)
```

Arguments:

`connectionDetails` DatabaseConnector::connectionDetails class

`loadConnection` Boolean option to load connection right away

`snakeCaseToCamelCase` (Optional) Boolean. return the results columns in camel case (default) Render Translate Sql.

Method `renderTranslateSql()`**:** Masked call to SqlRender

Usage:

```
ConnectionHandler$renderTranslateSql(sql, ...)
```

Arguments:

`sql` Sql query string

`...` Elipsis initConnection

Method `initConnection()`**:** Load connection Get Connection

Usage:

```
ConnectionHandler$initConnection()
```

Method `getConnection()`**:** Returns connection for use with standard DatabaseConnector calls. Connects automatically if it isn't yet loaded

Usage:

```
ConnectionHandler$getConnection()
```

Method `closeConnection()`**:** Closes connection (if active) close Connection

Usage:

```
ConnectionHandler$closeConnection()
```

Method `finalize()`**:** Closes connection (if active) db Is Valid

Usage:

```
ConnectionHandler$finalize()
```

Method `dbIsValid()`**:** Masks call to DBI::dbIsValid. Returns False if connection is NULL

Usage:

```
ConnectionHandler$dbIsValid()
```

Method `queryDb()`**:** query database and return the resulting data.frame

If environment variable LIMIT_ROW_COUNT is set Returned rows are limited to this value (no default) Limit row count is intended for web applications that may cause a denial of service if they consume too many resources.

Usage:

```
ConnectionHandler$queryDb(
  sql,
  snakeCaseToCamelCase = self$snakeCaseToCamelCase,
  overrideRowLimit = FALSE,
  ...
)
```

Arguments:

`sql` sql query string

`snakeCaseToCamelCase` (Optional) Boolean. return the results columns in camel case (default)

`overrideRowLimit` (Optional) Boolean. In some cases, where row limit is enforced on the system You may wish to ignore it.

`...` Additional query parameters

Method `executeSql()`: execute set of database queries

Usage:

```
ConnectionHandler$executeSql(sql, ...)
```

Arguments:

`sql` sql query string

`...` Additional query parameters query Function

Method `queryFunction()`: queryFunction that can be overriden with subclasses (e.g. use different base function or intercept query) Does not translate or render sql.

Usage:

```
ConnectionHandler$queryFunction(
  sql,
  snakeCaseToCamelCase = self$snakeCaseToCamelCase
)
```

Arguments:

`sql` sql query string

`snakeCaseToCamelCase` (Optional) Boolean. return the results columns in camel case (default) execute Function

Method `executeFunction()`: exec query Function that can be overriden with subclasses (e.g. use different base function or intercept query) Does not translate or render sql.

Usage:

```
ConnectionHandler$executeFunction(sql)
```

Arguments:

`sql` sql query string

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
ConnectionHandler$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`DataMigrationManager` *DataMigrationManager (DMM)*

Description

R6 class for management of database migration

Value

data frame all migrations, including file name, order and execution status Get connection handler

Public fields

`migrationPath` Path migrations exist in

`databaseSchema` Path migrations exist in

`packageName` packageName, can be null

`tablePrefix` packageName, can be null

Methods

Public methods:

- `DataMigrationManager$new()`
- `DataMigrationManager$migrationTableExists()`
- `DataMigrationManager$getMigrationsPath()`
- `DataMigrationManager$getStatus()`
- `DataMigrationManager$getConnectionHandler()`
- `DataMigrationManager$check()`
- `DataMigrationManager$executeMigrations()`
- `DataMigrationManager$isPackage()`
- `DataMigrationManager$finalize()`
- `DataMigrationManager$clone()`

Method `new()`:

Usage:

```
DataMigrationManager$new(
  connectionDetails,
  databaseSchema,
  tablePrefix = "",
  migrationPath,
  packageName = NULL,
  migrationRegexp = .defaultMigrationRegexp
)
```

Arguments:

`connectionDetails` DatabaseConnector connection details object

`databaseSchema` Database Schema to execute on

`tablePrefix` Optional table prefix for all tables (e.g. plp, cm, cd etc)

`migrationPath` Path to location of migration sql files. If in package mode, this should just be a folder (e.g. "migrations") that lives in the location "sql/sql_server" (and) other database platforms. If in folder model, the folder must include "sql_server" in the relative path, (e.g if migrationPath = 'migrations' then the folder 'migrations/sql_server' should exists)

`packageName` If in package mode, the name of the R package

`migrationRegexp` (Optional) regular expression pattern default is `(Migration_([0-9]+))-(.+).sql` Migration table exists

Method `migrationTableExists()`: Check if migration table is present in schema

Usage:

```
DataMigrationManager$migrationTableExists()
```

Returns: boolean Get path of migrations

Method `getMigrationsPath()`: Get path to sql migration files

Usage:

```
DataMigrationManager$getMigrationsPath(dbms = "sql server")
```

Arguments:

`dbms` Optionally specify the dbms that the migration fits under Get status of result model

Method `getStatus()`: Get status of all migrations (executed or not)

Usage:

```
DataMigrationManager$getStatus()
```

Method `getConnectionHandler()`: Return connection handler instance

Usage:

```
DataMigrationManager$getConnectionHandler()
```

Returns: ConnectionHandler instance Check migrations in folder

Method `check()`: Check if file names are valid for migrations Execute Migrations

Usage:

```
DataMigrationManager$check()
```

Method `executeMigrations()`: Execute any unexecuted migrations

Usage:

```
DataMigrationManager$executeMigrations(stopMigrationVersion = NULL)
```

Arguments:

`stopMigrationVersion` (Optional) Migrate to a specific migration number isPackage

Method `isPackage()`: is a package folder structure or not finalize

Usage:

```
DataMigrationManager$isPackage()
```

Method `finalize()`: close database connection

Usage:

```
DataMigrationManager$finalize()
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
DataMigrationManager$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

See Also

ConnectionHandler for information on returned class

generateSqlSchema *Schema generator*

Description

Take a csv schema definition and create a basic sql script with it.

Usage

```
generateSqlSchema(csvFilepath, sqlOutputPath = NULL, overwrite = FALSE)
```

Arguments

csvFilepath	Path to schema file. Csv file must have the columns: "table_name", "colum_name", "data_type", "is_required", "primary_key" Note -
sqlOutputPath	File to write sql to.
overwrite	Boolean - overwrite existing file?

Value

string containing the sql for the table

PooledConnectionHandler *Pooled Connection Handler*

Description

Transparently works the same way as a standard connection handler but stores pooled connections. Useful for long running applications that serve multiple concurrent requests.

Super class

ResultModelManager::ConnectionHandler -> PooledConnectionHandler

Methods

Public methods:

- PooledConnectionHandler$new()
- PooledConnectionHandler$initConnection()
- PooledConnectionHandler$closeConnection()
- PooledConnectionHandler$queryFunction()
- PooledConnectionHandler$clone()

Method new():

Usage:

```
PooledConnectionHandler$new(...)
```

Arguments:

... Elisis @seealsoConnectionHandler initialize pooled db connection

Method `initConnection()`: Overrides ConnectionHandler Call Close Connection

Usage:

```
PooledConnectionHandler$initConnection()
```

Method `closeConnection()`: Overrides ConnectionHandler Call query Function

Usage:

```
PooledConnectionHandler$closeConnection()
```

Method `queryFunction()`: Overrides ConnectionHandler Call. Does not translate or render sql.

Usage:

```
PooledConnectionHandler$queryFunction(
  sql,
  snakeCaseToCamelCase = self$snakeCaseToCamelCase
)
```

Arguments:

`sql` sql query string

`snakeCaseToCamelCase` (Optional) Boolean. return the results columns in camel case (default)

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
PooledConnectionHandler$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.