Upload Functionality

James P. Gilbert

2023-02-14

Contents

1 Introduction

This vignette describes the functionality for uploading results to a pre-created database schema. In the examples here, we assume the use of sqlite for simplicity. However, in principle any platform supported by the `DatabaseConnector` and `SqlRender` packages should work.

2 Creating a schema definition file

It is recommended that every analytics package that creates data output should contain a csv file. This is a requirement for packages that use the OHDSI `Strategus` library.

Schema definitions should conform to the following column headers:

```
table_name, column_name, data_type, is_required, primary_key
```

In addition, other packages may make use of additional fields such as `optional` or `empty_is_na`. These are not required for uploading to a schem.

An example csv file may look like this:

```
# File inst/settings/resulsDataModelSpecifications.csv
table_name,column_name,data_type,is_required,primary_key
table_1,database_id,varchar,Yes,Yes
table_1,analysis_id,bigint,Yes,Yes
table_1,analysis_name,varchar,Yes,No
table_1,domain_id,varchar(20),No,No
table_1,start_day,float,No,No
```

```
table_1,end_day,float,No,No
table_1,is_binary,varchar(1),Yes,No
table_1,missing_means_zero,varchar(1),No,No
table_2,database_id,varchar,Yes,Yes
table_2,analysis2_id,bigint,Yes,Yes
table_2,concept_id,int,Yes,No
table_2,logic_description,varchar,No,No
table_2,valid_start_date,Date,Yes,No
table_2,concept_name,varchar(255),Yes,No
table_2,p_10_value,float,Yes,No
table_3,database_id,varchar,Yes,Yes
table_3,analysis3_id,bigint,Yes,Yes
table_3,concept_id,int,Yes,No
table_3,logic_description,varchar,No,No
table_3,valid_start_date,Date,Yes,No
table_3,concept_name,varchar(255),Yes,No
table_3,p_10_value,float,Yes,No
```

Note the use of sql server data types in the `data_type` field as well as `yes` and `no` in the binary field `is_required` and `primary_key`.

We should also define a function for loading this file that converts the column headers to camel case format.

```
#'Get Results Data Model Specifcations
getResultsDataModelSpec <- function() {
  # For loading inside an R package
  specPath <- system.file("settings", "resulsDataModelSpecifications.csv", package = utils::packageName
  spec <- readr::read_csv(specPath, show_col_types = FALSE)
  colnames(spec) <- SqlRender::snakeCaseToCamelCase(colnames(spec))
  return(spec)
}
```

3 Creating a schema

This section describes how to use `RMM` to create a schema from a specifications file.

```
connectionDetails <- DatabaseConnector::createConnectionDetails("sqlite", server = "MySqliteDb.sqlite")
connection <- DatabaseConnector::connect(connectionDetails)
sql <- ResultModelManager::generateSqlSchema(schemaDefinition = getResultsDataModelSpec())
DatabaseConnector::renderTranslateExecuteSql(connection, sql, database_schema = "main", table_prefix =
DatabaseConnector::disconnect(connection)
```

Note that the SQL generated by `generateSqlSchema` contains the required parameter `database_schema` and the optional parameter `table_prefix`.

4 Uploading results

This section shows how to upload results conforming to the above specified schema. It is assumed that a zip file has been created with csv files corresponding to the table names provided in the `resulsDataModelSpecifications.csv` file.

```
ResultModelManager::unzipResults(zipFile = "MyResultsZip.zip", resultsFolder = "extraction_folder")
ResultModelManager::uploadResults(connectionDetails,
                                  schema = "main",
                                  resultsFolder = "extraction_folder",
                                  tablePrefix = "pre_",
                                  purgeSiteDataBeforeUploading = FALSE,
                                  specifications = getResultsDataModelSpec())
```

This will extract the zip file and upload results using `DatabaseConnector`. For most platforms some form of bulk loading is supported, the setup for this is contained in the documentation for `DatabaseConnector`, it is strongly advised that you set up and use this functionality for any platforms where you will store large data sets. However, this is not necessary for the sqlite example demonstrated here.