Using An Export Manager

James P. Gilbert

2023-09-14

Contents

1 Introduction

OHDSI studies often have very specific requirements in terms of exposing patient details.

2 Creating the export manager for a package

The table Specification must be definied for a table to be exported. Crucially, the data types, column names, primary keys and valid settings are always validated at the time of export. You cannot export data that does not conform to this model, so make sure that this model matches the result schema that the data are being imported in to. It is assumed that package developers will include this in the unit tests of their package.

```
library(ResultModelManager)

tableSpecification <- dplyr::tibble(
  tableName = c(
    "my_table", "my_table", "my_table", "my_table", "my_table", "my_table", "my_table",
    "my_andromeda_table", "my_andromeda_table", "my_andromeda_table"
  ),
  columnName = c(
    "database_id", "target_cohort_id", "comparator_cohort_id", "target_count", "comparator_count", "rr"
    "database_id", "covariate_id", "value"
  ),
```

```
  primaryKey = c(
    "yes", "yes", "no", "no", "no", "no", "no",
    "yes", "yes", "no"
  ),
  minCellCount = c(
    "no", "no", "no", "yes", "yes", "no", "no",
    "no", "no", "no"
  ),
  dataType = c(
    "varchar(255)", "int", "int", "int", "int", "float", "float",
    "varchar(255)", "bigint", "float"
  )
)

# Per database export folder is a good principle to follow
exportDir <- "output_folder/example_cdm"
exportManager <- createResultExportManager(
  tableSpecification = tableSpecification,
  exportDir = exportDir,
  databaseId = "example_cdm"
)
```

3 Saving large results sets with a batch operation

As data sets can easily exceed system memory, any operations should be performed in batch (via the export manager's exposed functions with a callback), or exporting from an Andromeda object.

3.1 Setup

First we will connect to a test database and create some test data:

```
connection <- DatabaseConnector::connect(server = ":memory:", dbms = "sqlite")
schema <- "main"

# Some made up counts
data <- data.frame(
  target_cohort_id = 1:100,
  comparator_cohort_id = 101:200,
  target_count = stats::rpois(100, lambda = 10),
  target_time = stats::rpois(100, 100000),
  comparator_count = stats::rpois(100, lambda = 5),
  comparator_time = stats::rpois(100, 100000)
)

DatabaseConnector::insertTable(connection, data = data, tableName = "result_table", databaseSchema = sc
```

3.2 Exporting a database query result

```
sql <- "SELECT * FROM @schema.result_table"
exportManager$exportQuery(connection = connection, sql = sql, exportTableName = "my_table", schema = sc
```

It is vital to ensure that the returned result set conforms to your data model, including the primary key columns specified. Otherwise, export validation will fail to prevent errors in exported csv files.

If you look at the file `output_folder/example_cdm/my_table.csv` you will notice that the database_id field is populated, you should not add this in SQL as it will be completed per database automatically.

Note that this result set is incomplete - we're not exporting fields that would be computed using an R function, just the values that are exported from an sql query.

3.2.1 Performing R operations

In order to perform R operations (for example, computing a rate ratio or p-value that would be difficult to compute in SQL) it is recommended that is performed inside a callback function to the `exportQuery` method. Modifying the above to include a rate ratio calculation using the `rateratio.test` package:

```
library(rateratio.test)

transformation <- function(rows, pos) {
  rrResult <- rateratio.test(
    x = c(row$target_count, row$comparator_count),
    n = c(row$target_time, row$comparator_time),
    RR = 1,
    conf.level = 0.95
  )

  row$rr <- rrResult$estimate
  row$p_value <- rrResult$p.value

  return(row)
}

exportManager$exportQuery(connection,
  sql,
  "my_table",
  transformFunction = transformation,
  transformFunctionArgs = list(),
  append = FALSE,
  schema = schema
)
```

3.3 Exporting an Andromeda result in batch

It is generally inadvisable to collect an entire andromeda table for export in to the R session before saving to disk. Instead, it is best practice to use batch operations as follows

```
andr <- Andromeda::andromeda()
andr$my_andromeda_table <- data.frame(covariate_id = 1:1e4, value = stats::runif(1e4))

first <- TRUE
```

```r
writeBatch <- function(batch) {
  exportManager$exportDataFrame(batch, "my_andromeda_table", append = first)
  first <<- FALSE
  # we don't want to return anything, just write the result to disk
  return(invisible(NULL))
}

Andromeda::batchApply(andr$my_andromeda_table, writeBatch)
```

4 Creating a results manifest file

Export manifests contain an sha256 hash of all files exported. This can be useful to see if a file was modified or corrupted before inclusion. To export the manifest of files within an export directory:

```r
exportManger$writeManifest(packageName = "analytics_package", packageVersion = packageVersion("analytic
```