Package Design

James P. Gilbert

2024-08-20

Contents

Introduction

ResultModelManager is a package that aims to standardize the handling of data models across OHDSI Hades packages. This package is designed to have minimal requirements and be easily imported by others.

Problem statement

At the time of writing, OHDSI analytics packages have many different and specific results data models that can change between package versions. When changes occur to these data models occurs, old versions of dependent software become incompatible with new versions. Similarly, when applications that depend on these data models encounter these changes developers are either forced in to the following assumptions:

- Old results will never work with new features
- Old results cannot be easily "upgraded"
- That new features of bug fixes to web applications can't be applied to old data
- That users can generate new results - these may have been generated by network studies
- That results data models between two apps are immutable and cannot be readily changed
- That developers will have to make one-off 'hacks' to support fixes to legacy results data models
- That results generated at different times cannot be directly compared without significant time to massage data into a common format

Furthermore, as the Strategus package is being developed it has been noted that many existing analytics packages generate results that are in non standard formats. Any time data models are changed, this breaks any backwards compatibility.

Package name

Work in progress name: ResultModelManager (RMM or, phonetically, rum).

Package purpose

This goal of package is intended to provide common data model utilities that can be used across OHDSI packages. The initial focus will be on handling database migrations via a common interface that can be used within R packages to add new data models (where not existing), handle migrations to future versions, report the status of a given data model's version, get migrations that are to be run and execute SQL scripts that change a data model in sequential order.

In addition, common functionality for defining and querying data models can be provided in a way that allows easier access to results data models for the purpose of post-collection data analysis.

This is similar in function to a software utility like flyway (used successfully in OHDSI WebAPI) - however, this is geared towards `R` and the `OHDSI` landscape of packages, with results that have been generated by ohdsi analytics packages and are intended to be explored using OHDSI open source reporting tools such as shiny and Rmarkdown. From analysis - it is not simple to expect users to install flyway, and would not be trivial to store and handle migrations in this way.

Scope and Intended Use

This package only intends to provide utilities for R Ohdsi packages that require exploration of results in a common way.

This package will mainly be used by OHDSI package maintainers and is not intended for users of those software. Though some functionality (such as alerting users to the fact that a data model is out of date) it will be up to the package maintainers to implement the interfaces provided here. In terms of Data model migrations and querying this package will provide a series of abstract base classes/interfaces (note: this distinction is hard to enforce within R6) that can be implemented within individual packages. The packages themselves will create classes that can perform the required tasks.

System Features and Requirements

General

Package dependencies

The obvious initial dependencies are `ParrallelLogger`, `R6`, `DatabaseConnector` and `SqlRender`. Beyond this the package should be fairly lightweight and should not require packages from outside CRAN. It is not an initial requirement that this package is in CRAN but all development should conform to this unless a requirement makes this impossible.

Data Migration Manager (DMM) Class

The results migrator class will be defined as an `R6` class that has the following base functions:

- Init with parameters
- DatabaseConnector connection or connectionDetails object
- results database schema
- table prefix
- Location of migration files (e.g. a package or relative path)
- load migrations - return a data.frame of all migration file paths and the order they are to be executed in
- `status()` Get the status of a current data model - this should return the ordered migrations and flag those that have or haven't been executed
- `check()` Check that migrations conform to standards (can be used in package tests)

- `executeMigrations()` execute any migrations that have not been completed

Class definition

```
DataMigrationManager <- R6::R6Class(
  "DataMigrationManager",
  private = list(
    executeMigration = function(filePath) {
      # Load, render, translate and execute sql

      # Save migration in set of migrations

      # Error handling - stop execution, restore transaction
    }
  ),
  public = list(
    migrationPath = NULL,
    migrationFolder = NULL,
    resulultsDatabaseSchema = NULL,
    connection = NULL,
    initalize = function(connectionDetails,
                         resultsDatabaseSchema,
                         tablePrefix,
                         migrationsPath,
                         migrationRegexp = .defaultMigrationRegexp) {
      # Set required variables
    },
    getStatus = function() {
      # return data frame all migrations, including file name, order and
    },
    check = function() {
      # Check to see if files follow pattern
    },
    executeMigrations = function() {
      # load list of migrations
      # Load list of executed migrations
      # if migrations table doesn't exist, create it
      # execute migrations that haven't been executed yet
    }
  )
)
```

Example implementation for package

```
#' @inheritParams ResultModelManager::DatabaseMigrationManager - this will probably need to be a factory
#' @export`
getMigrationManager <- function(migrationsPath = system.file("sql", "sql_server", "migrations", package
  migrationManager <- ResultModelManager::DatabaseMigrationManager$new(migrationsPath = migrationsPath,
}
```

It should also be possible to load the manager from a directory structure outside of a package. However, this may have limitations when considering the way sql files are loaded with SqlRender, especially when including scripts that have platform specific changes.

Proposed structure of a migration SQL script

Naming convention

All data migrations are assumed to be in OHSI SQL and stored within a migration folder. For a package this should be `inst/sql/sql_server/migrations` (these will also likely have to be platform specific, e.g. `inst/sql/postgresql/migrations`). Inside this folder only migrations that conform to a regular expression such as `(Migration_[0-9]+)-(.+).sql`. Explicitly, this encodes several things:

- That the file is a migration and only intended to be executed once and by a DMM instance
- The position in the sequence in which the migration will be executed (i.e. a natural number)
- The string name of the migration
- The fact that its an sql file

For example, the following file names will work:

```
Migration_2-MyMigration.sql
Migration_2-v3.2whaterver.sql
Migration_4-TEST.sql
Migration_4-2018922-vAAAA.sql
```

However, the following would be invalid:

```
MyMigration.sql # Does not include Migration_1
Migration_2v3.2whaterver.sql # missing -
-TEST_Migration_1.sql # Wrong order
Migraton_4-a.sql # Migration spelt wrong
```

The `check()` call within the DMM should validate these. Package/module maintainers should also ensure that migrations conform to this by adding a unit test.

Required parameters

Any table names or other variables should be set using the SqlRender parameter `{DEFAULT @table_name = my_table_name}`. The script will have the variables:

```
@results_schema
@table_prefix
```

The use of `@table_prefix` when referencing tables is crucial. However, in general use of parameters is not seen as necessary at this time. (NOTE: I'm not sure how we would handle parameterized migrations, or if this is even desirable. Users can manage this with the `DEFAULT` keyword in SqlRender)

Example SQL

```
-- changes incidence_rate.person_years from bigint to float
{DEFAULT @migration = migration}
{DEFAULT @incidence_rate = incidence_rate}
{DEFAULT @table_prefix = ''}

ALTER TABLE @results_schema.@table_prefix@incidence_rate ALTER COLUMN person_years FLOAT;
```

Adding script to list of executed migrations

Following a migration the following SQL will execute:

```
INSERT INTO @results_schema.@table_prexif@migration (migration_completed, migration_order)
    VALUES ('<fileName>', <order>);
```

(Note - this should be completed automatically by the manager class)

Results Data model class

- Take a connection
- handle queries for pooled or non-pooled connections
- Common getter methods
- Getting the current data model version for use in shiny apps and reporting tools

Utilities

It is desirable, but not a hard requirement for the initial package version, to have the following helper utilities.

Add migration function

To provide the ability to add a new migration to an existing project

Create DDL design

Function to create a ddl csv that conforms to common standards

Create new DDL

In principle this could be added for new projects that will generate results. However, this may be of limited use.

Limitations and scope

This package will not be able to support the following:

- New features (e.g. an entirely new table) cannot be added to old results - handling the case where results do not exist should be downstream of this package
- Down migrations for existing data models
- Backup management - this utility may support immutable changes to data that may go wrong. It is up to individual organisations to manage their data backups through taking necessary steps to mitigate data loss
- Perform transformations to data in R or other software functions - this package will not support data manipulation methods that are handled in R; just the structure of the ddl. If manipulations in R are required, this should be handled through the package functions.
- This is not intended to manage changes to OHDSI vocabularies or CDMs!