Creating Migrations

James P. Gilbert

2022-08-30

Contents

1 Introduction

Migrating existing data models can be a tricky process that often creates incompatibility between result viewers and existing result sets. This guide aims to show how to use the `ResultModelManager` class to create migrations for a given result model, either using a package or file structure. Please see the HADES library for more information on HADES packages.

2 Assumptions

This package assumes that you are familiar with R and OHDSI Hades packages in general. These examples make use of `DatabaseConnector` and `SqlRender`. The management of data integrity is left to the user, migrations should be designed and tested before deployment. Steps to maintain the data (such as backup plans) should be made prior to performing migrations in case of data corruption.

3 Creating the required file structure

The first step is creating a proper folder structure for migrations. The chosen path is dependent on the structure used, the most consistent and recommended way is to expose a function within an R package to allow users to upgrade a data model. However, a flat folder structure that does not require an R package to be installed is also supported.

3.1 In an R package

Data migrations should exist in an isolated folder within the `/inst/sql/` directory of a package. The recommended convention is to use `migrations` across all Hades package. As migrations are supported by multiple database platforms this folder should exist within the generic (and SqlRender OHDSI common sql) `sql_server` folder, `inst/sql/sql_server/migrations`. For any database specific migrations they should be

3.2 Using folder structure

4 Adding a migration

5 Unit testing

No specific advice is given for how to write unit tests for migrations, however, it is strongly advised that migrations are unit tested.

6 Common issues

The following is a list of expected issues when handling Data migrations:

6.1 Supporting all database platforms

It is likely a challenge to support all `SqlRender/DatabaseConnector` supported dbmses. Therefore, careful consideration with regards to supported platforms should be made. At the time of writing, for results handling, we recommend supporting the open source platforms of SqlRender and Postgresql. This decision is left to the package author.

6.2 SQLite column types

It is a not possible to change a data type within an Sqlite table (the `ALTER TABLE` command does not work). Consequently, you will likely have to rename the existing table, create a new table with the modified DDL and then copy the existing data across (using appropriate data transformations/casting).

For example, changing an `INT` column in the table `foo` to a float requires the sqlite specific transformation:

```
{DEFAULT @foo = foo}

ALTER TABLE @database_schema.@table_prefix@foo RENAME TO _foo_old;
```

```
CREATE TABLE @database_schema.@table_prefix@foo (
    id bigint,
    foo float
);

INSERT INTO @database_schema.@table_prefix@foo (id, foo)
SELECT * FROM _foo_old;
```

6.3 Non-existent data

The presence of a data model does not mean data is present. As packages are developed, it is expected that new data formats will be created. The recommended pattern for this case is to allow existing data to be upgraded but to handle the use case of missing data in downstream reports/web applications.