Package 'HadesResultsModel'

July 3, 2026

Type Package

Title HADES Results Model Definitions and Utilities

Version 0.1.0

Description Lightweight R package for working with OHDSI HADES results model resources, including module YAML definitions, release manifests, JSON schemas, and migration and DDL generation utilities.

License Apache License (>= 2)

Encoding UTF-8

LazyData true

URL https://ohdsi.github.io/HadesResultsModel/, https://github.com/OHDSI/HadesResultsModel

BugReports https://github.com/OHDSI/HadesResultsModel/issues

Imports yaml,
DBI,
SqlRender,
DatabaseConnector

Suggests testthat,
jsonlite,
jsonvalidate,
duckdb,
knitr,
rmarkdown

VignetteBuilder knitr

Config/testthat/edition 3

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.0.0

Contents

`applyMigrationSql` *Apply a migration SQL script to an active DBI connection*

Description

Reads a SQL migration file, renders it with SqlRender, and executes it against an existing DBI connection.

Usage

```
applyMigrationSql(connection, migrationFile, databaseSchema = "main")
```

Arguments

`connection`	An active DBI connection object.
`migrationFile`	Path to a SQL migration file.
`databaseSchema`	Schema name used when rendering SQL with SqlRender.

Value

Invisibly returns `TRUE` when execution succeeds.

`buildLatestRelease` *Build the latest quarterly release manifest*

Description

Builds a release manifest by selecting the latest semantic version folder for each module and writing the result to the package releases directory.

Usage

```
buildLatestRelease(
  modulesRoot = resolvePackageDir("modules"),
  releasesRoot = resolvePackageDir("releases"),
  releaseDate = Sys.Date()
)
```

Arguments

modulesRoot	Path to the modules root directory.
releasesRoot	Path to the releases output directory.
releaseDate	Date used to derive release version (vYYYY_QN).

Value

The full path to the generated release manifest YAML file.

---

convertCsvToYaml *Convert a legacy CSV results data model specification to a YAML definition*

Description

Reads a CSV file describing a module's results data model and writes a definition.yaml compatible with the HADES module schema to outputDir/moduleName/version/definition.yaml.

Usage

```
convertCsvToYaml(
  csvFile,
  outputDir,
  moduleName,
  prefix,
  version = "v1.0.0",
  addPrefix = NULL
)
```

Arguments

csvFile	Path to the input CSV file.
outputDir	Root output directory. The definition is written to outputDir/moduleName/version/definition.y
moduleName	Name of the module (e.g.\ "CohortMethod").
prefix	Table name prefix including trailing underscore (e.g.\ "cm_").
version	Target module version string (default "v1.0.0").
addPrefix	Whether to prepend prefix to table names that do not already carry it. NULL (default) auto-detects based on the table names found in the CSV.

Details

The CSV must contain at least columns for table name, column/field name, and data type. Optional columns for description, primary key flag, and deprecation flag are used when present. Column names are matched case-insensitively against common variants (e.g.\ table_name, column_name, data_type, description, primary_key).

Value

Invisibly returns the path to the written YAML file.

findLatestReleaseManifest

Find the latest release manifest file

Description

Selects the newest release manifest in `releasesDir` using year and quarter ordering based on file names like `release_vYYYY_QN.yaml`.

Usage

```
findLatestReleaseManifest(releasesDir = resolvePackageDir("releases"))
```

Arguments

`releasesDir`	Path to the releases directory.

Value

Full path to the latest release manifest, or `NA_character_` if none are found.

generateModuleDdl

Generate OHDSI SQL DDL for one or all HADES modules

Description

Produces OHDSI SQL `CREATE TABLE` statements for the specified module(s) at the chosen version, or for every module when `module` is `NULL`. The SQL is parameterized with `@database_schema` and can be rendered with `SqlRender::render(sql, database_schema = "mySchema")`.

Usage

```
generateModuleDdl(
  module = NULL,
  version = "latest",
  modulesRoot = resolvePackageDir("modules"),
  databaseSchema = "@database_schema"
)
```

Arguments

`module`	Character vector of module name(s) (e.g.\ `"CohortMethod"`), or `NULL` (default) to include all modules found under `modulesRoot`.
`version`	Version string such as `"v1.0.0"` or `"1.0.0"`, or `"latest"` (default) to select the highest semantic version for each module.
`modulesRoot`	Path to the package modules directory.
`databaseSchema`	Schema parameter to embed. Defaults to `"@database_schema"` for deferred rendering.

Value

A character string of OHDSI SQL `CREATE TABLE` statements.

`generateReleaseDdl` *Generate release-level CREATE TABLE DDL from a release manifest*

Description

Builds OHDSI SQL DDL for all tables in the selected release manifest and writes the resulting SQL script to disk. This is a maintainer utility; use `generateModuleDdl()` for programmatic DDL generation.

Usage

```
generateReleaseDdl(
  releaseFile = NULL,
  modulesRoot = resolvePackageDir("modules"),
  releasesRoot = resolvePackageDir("releases"),
  sqlRoot = file.path(getwd(), "sql")
)
```

Arguments

`releaseFile`	Optional path to a release manifest YAML file. When `NULL`, the latest manifest in `releasesRoot` is used.
`modulesRoot`	Path to the modules root directory.
`releasesRoot`	Path to the releases directory containing manifests.
`sqlRoot`	Output directory for generated SQL files.

Value

The full path to the generated SQL file.

`getMigrationPath` *Resolve the ordered list of migration SQL files between two module versions.*

Description

Traverses `inst/modules/{module}/` to build the chain of `migration.sql` files required to advance from `currentVersion` to `targetVersion`. Returns an empty list when the versions are equal or `currentVersion` is `"0.0.0"`.

Usage

```
getMigrationPath(
  module,
  currentVersion,
  targetVersion,
  modulesRoot = resolvePackageDir("modules")
)
```

Arguments

module	Module name matching a directory under modulesRoot.
currentVersion	Installed version string without leading v (e.g. "0.1.0").
targetVersion	Target version string without leading v (e.g. "1.0.0").
modulesRoot	Path to the package modules directory.

Value

An ordered list of absolute paths to migration.sql files.

---

getOrCreateRegistry *Get or create the central HADES version registry table.*

Description

Checks whether hades_result_version exists in the target schema. If it does not exist, the table is created and populated from inferCurrentVersions. If it already exists, the current version map is read and returned.

Usage

```
getOrCreateRegistry(
  connection,
  databaseSchema,
  modulesRoot = resolvePackageDir("modules")
)
```

Arguments

connection	An active DatabaseConnector connection.
databaseSchema	Schema name.
modulesRoot	Path to the package modules directory.

Value

A named list of module versions currently recorded in the registry.

inferCurrentVersions *Infer current module versions by fingerprinting an existing database schema.*

Description

Uses `DatabaseConnector::getTableNames()` to list tables in the schema, then issues `SELECT TOP 1 * FROM @database` for each table to obtain its column names. These are compared against the YAML definitions in `inst/modules/` to determine which version of each module is installed. When a module has no tables present, its version is reported as `"0.0.0"`.

Usage

```
inferCurrentVersions(
  connection,
  databaseSchema,
  modulesRoot = resolvePackageDir("modules")
)
```

Arguments

`connection`	An active DatabaseConnector connection.
`databaseSchema`	Schema name to inspect.
`modulesRoot`	Path to the package modules directory.

Value

A named list mapping module names to detected version strings.

migrateResultsModel *Migrate a HADES results database to a target calendar release.*

Description

Opens a connection using `connectionDetails`, detects or creates the central `hades_result_version` registry (using fingerprinting when the table is absent), resolves the SQL migration chain for each module that needs upgrading, and executes those migrations. The registry is updated after each successful module migration. The connection is closed on function exit.

Usage

```
migrateResultsModel(
  connectionDetails,
  databaseSchema = "main",
  targetRelease = "latest",
  modulesRoot = resolvePackageDir("modules"),
  releasesRoot = resolvePackageDir("releases")
)
```

Arguments

connectionDetails	A DatabaseConnector connection details object created with DatabaseConnector::createConnec
databaseSchema	Schema name for the results database.
targetRelease	Calendar release label such as "v2026_Q3", or "latest" to select the newest available manifest automatically. A full path to a release manifest YAML file is also accepted.
modulesRoot	Path to the package modules directory.
releasesRoot	Path to the package releases directory.

Value

Invisibly returns a named list of final module versions recorded in the registry after migration.

---

print.ValidationResult *Print method for ValidationResult objects*

Description

Print method for ValidationResult objects

Usage

```
## S3 method for class 'ValidationResult'
print(x, ...)
```

Arguments

x	A ValidationResult object.
...	Additional arguments (ignored).

---

testMigrationSql *Test that a migration SQL file correctly transforms a module schema*

Description

Creates an in-memory DuckDB database, builds the starting schema from the fromVersion YAML definition, applies the migration SQL, and then verifies that the resulting schema contains all tables and columns declared in toDefinition. Throws an informative error on any mismatch; returns invisible(TRUE) on success.

Usage

```
testMigrationSql(
  module,
  migrationFile,
  fromVersion = "latest",
  toDefinition,
  modulesRoot = resolvePackageDir("modules"),
  databaseSchema = "main"
)
```

Arguments

module	Module name as it appears under modulesRoot (e.g.\ "CohortGenerator").
migrationFile	Path to the migration.sql file to test.
fromVersion	Version string to start from (e.g.\ "v0.1.0") or "latest" (default) to use the highest registered version.
toDefinition	The expected post-migration schema. Accepts: • A version string such as "v1.0.0" (looked up in modulesRoot). • A path to a definition.yaml file (useful for unregistered versions). • A parsed list as returned by yaml::read_yaml().
modulesRoot	Path to the package modules directory.
databaseSchema	Schema name used when rendering the OHDSI SQL. Defaults to "main" (the DuckDB default schema).

Details

The migration SQL must be in OHDSI SQL format (i.e.\ parameterized with @database_schema) so it can be rendered by SqlRender::render().

Value

Invisibly returns TRUE when the migration test passes.

---

validateDatabase *Validate a database schema against a HADES results model release*

Description

Queries an existing database and validates that its schema conforms to the expected structure defined by a HADES results model release manifest. Checks table existence, column names, data types, primary keys, and column ordering.

Usage

```
validateDatabase(
  connection,
  databaseSchema,
  targetRelease = "latest",
  modulesRoot = resolvePackageDir("modules"),
  releasesRoot = resolvePackageDir("releases"),
  strict = FALSE
)
```

Arguments

`connection`	An active DatabaseConnector connection.
`databaseSchema`	Schema name to validate.
`targetRelease`	Release label such as `"v2026_Q3"`, or `"latest"` to select the newest available manifest automatically. A full path to a release manifest YAML file is also accepted.
`modulesRoot`	Path to the package modules directory.
`releasesRoot`	Path to the package releases directory.
`strict`	If `FALSE` (default), missing expected tables are reported as warnings. If `TRUE`, missing tables are errors and the overall result is `passed = FALSE`.

Details

By default validates against the latest released version, but users can override to target a specific version (e.g. `"v2026_Q1"`) or provide a full path to a custom release manifest YAML file.

Missing tables (for example SelfControlledCohort tables, which are not included in the example data) are reported as warnings by default. Set `strict = TRUE` to treat missing expected tables as errors.

Value

A list of class `"ValidationResult"` with components:

passed Logical indicating overall validation success.

errors Character vector of error messages.

warnings Character vector of warning messages.

moduleDetails Named list with per-module validation details.

releaseVersion The release version string that was validated against.

`yamlDefinitionToSql` *Convert a module YAML definition to OHDSI SQL CREATE TABLE statements*

Description

Reads a HADES module definition (file path or parsed list) and produces OHDSI SQL `CREATE TABLE` statements parameterized with `@database_schema`. Render the returned SQL with `SqlRender::render(sql, database_schema = "mySchema")` before execution.

Usage

```
yamlDefinitionToSql(
  definition,
  databaseSchema = "@database_schema",
  additionalTables = c("cg_cohort_definition", "database_meta_data")
)
```

Arguments

`definition`	Either a path to a `definition.yaml` file or a list as returned by `yaml::read_yaml()`.
`databaseSchema`	Schema parameter to embed. Defaults to the literal `"@database_schema"` for deferred rendering via `SqlRender::render()`.
`additionalTables`	Character vector of table names from other modules that may appear in foreign-key references. Defaults to the two standard cross-module tables.

Value

A single character string of OHDSI SQL `CREATE TABLE` statements.

Index