

Custom database management system support for the OMOP CDM

Martí Català¹, Edward Burn¹

¹Centre for Statistics in Medicine (CSM), Nuffield Department of Orthopaedics, Rheumatology and Musculoskeletal Sciences (NDORMS), University of Oxford, Oxford, UK

Background

The OMOP Common Data Model (CDM) provides a means of standardising health care data so that researchers can conduct federated research across a network. While adopting the OMOP CDM leads to standardisation of data, the infrastructure on which this is stored and queried varies. Data partners use different database management systems, while the size of data sources and the available compute can vary dramatically.

So that the same analytic code can be run across the network most OHDSI software typically relies on generic SQL translations and the ODBC or JDBC drivers which provide standards that are used for connecting to databases. This approach allows for general purpose analytics that work across the network. However, this approach does bring the limitation that some functionalities of a particular database management are not utilised. For example, one platform that is increasingly used is Databricks, an optimised platform for Apache Spark analytics engine. While ODBC and JDBC can be used with Databricks, there are many aspects of the platform that are best utilised using native tools (such as user defined functions). Meanwhile, even for established database management systems, like PostgreSQL and SQL Server, optimal approaches to indexing can vary.

Our objective was to improve database specific support for OMOP CDM, creating a set of R packages designed to support particular backends.

Methods

We have created a set of R packages designed for particular database management systems. These include OmopOnDuckDB, that supports in-process duckdb databases, OmopOnPostgres, that supports PostgreSQL databases, OmopOnSqlServer, that supports SQL Server, and OmopSparkConnector, that provides native Spark on Databricks.

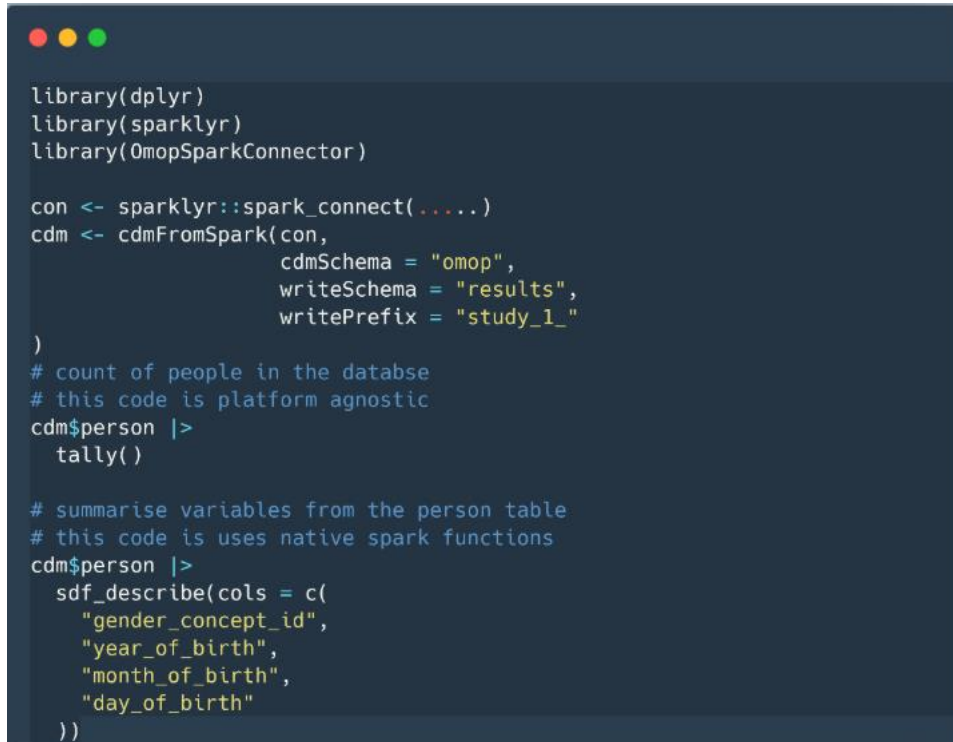
These packages implement core omopgenerics methods (<https://github.com/darwin-eu-dev/omopgenerics>). These include methods to insert tables from R into the database, to create temporary tables, to create permanent tables, and to collect data from the database back into R. By implementing these methods, downstream analytics packages to create study cohorts (<https://ohdsi.github.io/CohortConstructor/>), to calculate incidence rates and prevalence of health events (<https://darwin-eu.github.io/IncidencePrevalence/>), and in-depth characterisation of drug utilisation (<https://darwin-eu-dev.github.io/DrugUtilisation/>) can then be used most efficiently.

In addition, these packages also make it possible for users to make use of backend-specific functionality. For example, when working with OmopSparkConnector it is also possible for users to make use of Spark specific functions. These range from spark functions to summarise a table (https://spark.posit.co/packages/sparklyr/latest/reference/sdf_describe.html) to functions created to support machine learning (https://spark.posit.co/packages/sparklyr/latest/reference/ml_kmeans.html).

Results

We have created a set of R packages that provide customised database management support: OmopOnDuckDB, OmopOnPostgres, OmopOnSqlServer, and OmopSparkConnector.

All packages provide a similar interface. In the simple example code below, we can see how someone can work with the OmopSparkConnector package. After connecting to their database using the sparklyr package they would create a cdm reference, here using cdmFromSpark from OmopSparkConnector. From there general analytic code can be used in the same way as if the cdm reference had been created using any backend package. In addition, spark specific functionality can also be used, such as in this example for summarising data in the person table.

A screenshot of a dark-themed R console window. The code is written in a light blue/grey font. It starts with three library calls: library(dplyr), library(sparklyr), and library(OmopSparkConnector). Then, a connection 'con' is established using sparklyr::spark_connect(...). A cdm object is created with cdmFromSpark(con, cdmSchema = "omop", writeSchema = "results", writePrefix = "study_1_"). The code then performs two operations on the cdm\$person table: a tally() and an sdf_describe() for specific columns. The sdf_describe() call lists columns: gender_concept_id, year_of_birth, month_of_birth, and day_of_birth.

```
library(dplyr)
library(sparklyr)
library(OmopSparkConnector)

con <- sparklyr::spark_connect(... )
cdm <- cdmFromSpark(con,
  cdmSchema = "omop",
  writeSchema = "results",
  writePrefix = "study_1_"
)
# count of people in the database
# this code is platform agnostic
cdm$person |>
  tally()

# summarise variables from the person table
# this code is uses native spark functions
cdm$person |>
  sdf_describe(cols = c(
    "gender_concept_id",
    "year_of_birth",
    "month_of_birth",
    "day_of_birth"
  ))
```

Figure 1. Example code when working with OmopSparkConnector

Conclusion

The R packages we have created provide bespoke database support for the OMOP CDM. These packages allow users to both use of general cross-platform tools used in network studies while also providing the flexibility for researchers to use database specific functions when needed. This approaches also provides a straightforward approach for developers to add and maintain support for other database management systems.