CPRDTools: a CPRD GOLD data wrangling toolkit

James C. F. Schmidt

Abstract

The analysis of large scale data, and moreover the analysis of large scale electronic health records data has become more commonplace. The ease and ability of modern data generation and capture means there is the potential across almost all industries to capture more data now than ever before, and that equates to large data resources, with the Clinical Practice Research Datalink (CPRD) no exception. These large data resources, as attractive and appealing as they may be to researchers, pose a significant problem - how to manage all that data? `CPRDTools` is a collection of wrapper R functions intended to simplify the loading, extraction and management of CPRD GOLD specific and associated electronic health records data. Allowing for the loading of CPRD and non-CPRD data into a SQLite database, providing an efficient, secure and updatable repository for the data, keeing the origional source files intact. Through data queries, user-defined data are drawn allowing for subsetting, joining and filtering in a single step, creating alaysis ready data.

Introduction

The CPRD is one of the largest longitudinal medical records databases in the world, supported by the National Institute of Health (NIHR) and the Medicines and Healthcare products Regulatory Agency (MHRA). It was first established in 1987 as the Value Added Medical Products (VAMP) dataset, this grew into the General Practice Research Database (GPRD) in 1993, before its final transition into CPRD in 2012 (Herrett et al. 2015).

CPRD GOLD data are comprised of ten separate datasets: patient, practice, staff, consultation, clinical, additional clinical details, referral, immunisation, test and therapy (Padmanabhan 2017). These datasets contain their specific data and are linkable through a unique linkage key field, where key does not imply importance but a unique variable contained in two datasets allowing them to be joined, such as the CPRD-assigned and anonymised unique patient identifier `patid`.

Due to the size of CPRD, data extracted for research are spread over multiple text (`.txt`) files within each dataset to enable file transfer. This means that for CPRD clinical data, for instance, a researcher may receive their requested clinical data broken up over numerous individual text files. This is done to aid with file completeness and reduce turn-around times if errors are found. If an error occurred during the transfer of data between the data owner and the researcher or in the extraction of the requested data by the data owner, the error can potentially be limited to only select files, requiring only their replacement with the corrected/error-free versions.

Often these text files are additionally zipped, or compressed, requiring that these files first be uncompressed or unzipped. These multiple files from each dataset (clinical, referral etc.) then need to be grouped together and amalgamated into a single `table`. Tables are a collection of data of the same shape, from the same dataset. In CPRD, each separate dataset (patient, practice, clinical etc.) forms a table. These tables are then stored in the SQLite database.

SQLite is an opensource, SQL based database engine (SQLite 2017). The use of a SQLite database provides an efficient storage solution, allowing for the loading, updating and maintenance of the database, all while retaining the original *raw* data files unaltered. A SQLite database permits for rapid data extraction through the use of data queries, drawing the required data from the database, allowing filtering, sub-setting, limiting and the joining of data in a single execution step.

This document aims to provide a simple and introductory overview of `CPRDTools` and its application to arbitrary (fictional) data. This data though are provided in the manner in which many CPRD data extract are received, where data are spread over multiple files and often located in sub-folders.

`CPRDTools` is loaded using:

```
library(devtools)
install_github("JamesCFSchmidt/CPRDTools")
```

```
##
## * checking for file '/local/jcfs2/RtmpO31y2l/remotes5b94527882c4/JamesCFSchmidt-CPRDTools-a681e4d/DE
## * preparing 'CPRDTools':
## * checking DESCRIPTION meta-information ... OK
## * checking for LF line-endings in source and make files and shell scripts
## * checking for empty or unneeded directories
## Omitted 'LazyData' from DESCRIPTION
## * building 'CPRDTools_0.0.0.9000.tar.gz'
```

```
library(CPRDTools)
```

CPRDTools overview

The functions within 'CPRDTools' broadly fall into three groups, categorised by their general application area: (1) - loading, (2) - maintenance and other tasks and (3) - extraction. **Loading** encompasses all functions used in the reading, converting and writing of data into the database, including a function used to list all available files and all available CPRD files in a specified location. Functions used in database maintenance, query speed improvements and date conversion fall under **maintenance and other tasks**, and finally, functions used to, and in the process of, drawing and retrieving data from the database fall within **extraction**.

The data

The example data supplied with this package resembles CPRD GOLD data in layout and in file naming convention. Additionally included are example secondary files from 'Hospital Episode Statistics' (HES) patient information and 'Office for National Statistics' (ONS) actuarial life tables. The location of the file and the location where the database will be build must be provided

```
#database location
DB.path = "/home/j/jcfs2/Documents/Test"
#location of example data
FILE.path = dirname(system.file("extdata", "hes_patient_20336R.txt", package = "CPRDTools"))
```

Loading

Before loading any data into the database, it is best practice to understand the layout of the *raw* data. This can be achieved by either navigating to the location where the data are stored or by employing the `list_files` and `list_cprd` functions.

Listing available data

The `list_files` function provides a list of all files of a specified type in a specified location. To view all text (`.txt`) files in the data location

To view all compressed/*zipped* (`.zip`) files in the data location

And to view all files in the data location, excluding files in sub-folders

```
list_files(file_location = FILE.path,
           file_type = "all")
```

```
## $file_location
## [1] "/lustre/ahome3/j/jcfs2/R/x86_64-pc-linux-gnu-library/4.2/CPRDTools/extdata"
##
## $files
##                                Files
## 1 /1234.aa1_Extract_Practice_001.zip
## 2 /1234.aa2_Extract_Clinical_001.zip
## 3 /1234.aa2_Extract_Clinical_002.zip
## 4 /1234.aa2_Extract_Clinical_003.zip
## 5 /1234.aa2_Extract_Clinical_004.zip
## 6 /1234.aa2_Extract_Practice_001.zip
## 7            /hes_patient_20336R.txt
## 8                       /ons_lt.csv
## 9                          /patient
```

From the above it can be seen that there are six zipped text files (`.zip`), one excel file (`.xlsx`), one standard text file (`.txt`) and a folder, `patient`. In order to view CPRD GOLD specific files, corresponding to names CPRD GOLD datasets (*patient, practice, staff, consultation, clinical, additional clinical details, referral, immunisation, test and therapy*), the `list_cprd` function is used. This function provides the core functionality to the loading of CPRD GOLD data, generating a list of CPRD GOLD specific files in the specified location.

```
list_cprd(file_location = FILE.path,
          folder = F,
          zip = T)
```

```
## $file_location
## [1] "/lustre/ahome3/j/jcfs2/R/x86_64-pc-linux-gnu-library/4.2/CPRDTools/extdata"
##
## $all_files_tables
##                                Files    Table
## 2 /1234.aa2_Extract_Clinical_001.zip Clinical
## 3 /1234.aa2_Extract_Clinical_002.zip Clinical
## 4 /1234.aa2_Extract_Clinical_003.zip Clinical
## 5 /1234.aa2_Extract_Clinical_004.zip Clinical
## 1 /1234.aa1_Extract_Practice_001.zip Practice
## 6 /1234.aa2_Extract_Practice_001.zip Practice
##
## $tables
##      Table File_Count
## 1 Clinical          4
## 2 Practice          2
```

Here the folders input is defined as FALSE, showing only zipped data contained in the data location specified. When the folder input is TRUE, the available data found in the `patient` folder is displayed.

```
list_cprd(file_location = FILE.path,
          folder = T,
          zip = T)
```

```
## $file_location
## [1] "/lustre/ahome3/j/jcfs2/R/x86_64-pc-linux-gnu-library/4.2/CPRDTools/extdata"
```

```
##
## $all_files_tables
##                                  Files   Table
## 1 /patient/aa1_Extract_Patient_001.zip Patient
## 2 /patient/aa2_Extract_Patient_001.zip Patient
##
## $tables
##     Table File_Count
## 1 Patient          2
```

In the `$table` output, the `list_cprd` function defines the tables and the count of *raw* files associated with that table in the data location. This outputted list is core to the automated loading of CPRD data, using the tables and files in each table to load all or a selection of specified CPRD GOLD files.

Loading all available data

The `load_cprd` function, relying on the `list_cprd` function for file and table list from a specified data location is able to automatically load in a selection or all available CPRD GOLD files in a specified location. Importing, uncompressing (unzipping) and appending all files from a CPRD GOLD-specific table into a database.

To load the `patient` data into a new database, the `load_cprd` function is defined as

```
load_cprd(db_path = DB.path,
          file_location = FILE.path,
          tables_to_load = "Patient",
          folder = T,
          zip = T,
          load_mapping = T,
          overwrite = F)
```

```
## Rows: 12 Columns: 20
## -- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr  (4): frd, crd, tod, deathdate
## dbl (14): patid, vmid, gender, yob, marital, famnum, CHSreg, prescr, capsup,...
## lgl  (2): mob, CHSdate
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## ----------LOAD OF TABLE Patient, FILE No. 1 SUCCESSFUL----------

##
## Rows: 12 Columns: 20-- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr  (4): frd, crd, tod, deathdate
## dbl (14): patid, vmid, gender, yob, marital, famnum, CHSreg, prescr, capsup,...
## lgl  (2): mob, CHSdate
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## ----------LOAD OF TABLE Patient, FILE No. 2 SUCCESSFUL----------

##

## $database_location
## [1] "/home/j/jcfs2/Documents/Test"
```

```
##
## $files_in_location
##
## 1 /lustre/ahome3/j/jcfs2/R/x86_64-pc-linux-gnu-library/4.2/CPRDTools/extdata/patient/aa1_Extract_Pati
## 2 /lustre/ahome3/j/jcfs2/R/x86_64-pc-linux-gnu-library/4.2/CPRDTools/extdata/patient/aa2_Extract_Pati
##     table
## 1 Patient
## 2 Patient
##
## $tables_in_location
##     Table File_Count
## 1 Patient          2
##
## $loaded_tables
## [1] "Patient"
##
## $load_report
##     table load_total file_total size_Mb size_Gb
## 1 Patient          2          2   0.001       0
##
## $database_tables
## [1] "Patient"      "lookup_table"
##
## $load_start
## [1] "2022-10-04 12:28:57 BST"
##
## $load_end
## [1] "2022-10-04 12:28:58 BST"
##
## $load_time
## Time difference of 1.074941 secs
```

The `patient` data are found in a sub-folder within the data location () and the data are compressed (). As this is a new load of the database, no overwriting is required () and no CPRD specific mapping is needed ().

Alternatively to specifying each individual table to load (*additional, clinical, consultation, immunisation, patient, practice, referral, staff, test, therapy*), the specification of the loading of *all* tables can be performed.

```
load_cprd(db_path = DB.path,
          file_location = FILE.path,
          tables_to_load = "all",
          folder = F,
          zip = T,
          load_mapping = F,
          overwrite = F)
```

This returns an error

```
## Error in load_cprd(db_path = DB.path, file_location = FILE.path, tables_to_load = "all", : tables_to
```

The error is due to the `patient` table already being contained in the database. The should only be applied on new databases or in conjunction with . The remainder of the CPRD GOLD data are loaded with

```
load_cprd(db_path = DB.path,
          file_location = FILE.path,
          tables_to_load = c("Clinical","Practice"),
          folder = F,
```

```
          zip = T,
          load_mapping = F,
          overwrite = F)
```

```
## Rows: 12 Columns: 10
## -- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr (2): eventdate, sysdate
## dbl (8): patid, constype, consid, medcode, staffid, episode, enttype, adid
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## ----------LOAD OF TABLE Clinical, FILE No. 1 SUCCESSFUL----------
```

```
##
## Rows: 12 Columns: 10-- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr (2): eventdate, sysdate
## dbl (8): patid, constype, consid, medcode, staffid, episode, enttype, adid
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## ----------LOAD OF TABLE Clinical, FILE No. 2 SUCCESSFUL----------
```

```
##
## Rows: 12 Columns: 10-- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr (2): eventdate, sysdate
## dbl (8): patid, constype, consid, medcode, staffid, episode, enttype, adid
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## ----------LOAD OF TABLE Clinical, FILE No. 3 SUCCESSFUL----------
```

```
##
## Rows: 12 Columns: 10-- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr (2): eventdate, sysdate
## dbl (8): patid, constype, consid, medcode, staffid, episode, enttype, adid
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## ----------LOAD OF TABLE Clinical, FILE No. 4 SUCCESSFUL----------
```

```
##
## Rows: 5 Columns: 4-- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr (2): lcd, uts
## dbl (2): pracid, region
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
## ----------LOAD OF TABLE Practice, FILE No. 1 SUCCESSFUL----------
```

```
##
## Rows: 5 Columns: 4-- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr (2): lcd, uts
```

```
## dbl (2): pracid, region
## i Use ‘spec()‘ to retrieve the full column specification for this data.
## i Specify the column types or set ‘show_col_types = FALSE‘ to quiet this message.
## ----------LOAD OF TABLE Practice, FILE No. 2 SUCCESSFUL----------
##
## $database_location
## [1] "/home/j/jcfs2/Documents/Test"
##
## $files_in_location
##
## 2 /lustre/ahome3/j/jcfs2/R/x86_64-pc-linux-gnu-library/4.2/CPRDTools/extdata/1234.aa2_Extract_Clinica
## 3 /lustre/ahome3/j/jcfs2/R/x86_64-pc-linux-gnu-library/4.2/CPRDTools/extdata/1234.aa2_Extract_Clinica
## 4 /lustre/ahome3/j/jcfs2/R/x86_64-pc-linux-gnu-library/4.2/CPRDTools/extdata/1234.aa2_Extract_Clinica
## 5 /lustre/ahome3/j/jcfs2/R/x86_64-pc-linux-gnu-library/4.2/CPRDTools/extdata/1234.aa2_Extract_Clinica
## 1 /lustre/ahome3/j/jcfs2/R/x86_64-pc-linux-gnu-library/4.2/CPRDTools/extdata/1234.aa1_Extract_Practi
## 6 /lustre/ahome3/j/jcfs2/R/x86_64-pc-linux-gnu-library/4.2/CPRDTools/extdata/1234.aa2_Extract_Practi
##      table
## 2 Clinical
## 3 Clinical
## 4 Clinical
## 5 Clinical
## 1 Practice
## 6 Practice
##
## $tables_in_location
##      Table File_Count
## 1 Clinical          4
## 2 Practice          2
##
## $loaded_tables
## [1] "Clinical" "Practice"
##
## $load_report
##      table load_total file_total size_Mb size_Gb
## 1 Clinical          4          4   2e-03       0
## 2 Practice          2          2   5e-04       0
##
## $database_tables
## [1] "Clinical"     "Patient"      "Practice"     "lookup_table"
##
## $load_start
## [1] "2022-10-04 12:28:58 BST"
##
## $load_end
## [1] "2022-10-04 12:28:58 BST"
##
## $load_time
## Time difference of 0.7062566 secs
```

The `$loaded_tables` output shows the current tables loaded and available in the database ‘

Loading a single CPRD table

The `patient` table previously could have been loaded singularly using the `load_table` function. Similar to the `load_cprd` function, this will load CPRD-specific data tables, using the compiled list of available files and their respective tables generated in the `list_cprd` function. This function imports, unzips and appends the files specific to a CPRD GOLD table but performs the task on a single table at a time.

This may be of use when the data are stored in separate file locations, not within sub-folders in the same data location. The loading of the `practice` information is performed using

```
load_table(db_path = DB.path,
           file_location = FILE.path,
           table_name = "Practice",
           zip = T,
           overwrite = T)
```

```
## -------Overwrite = TRUE, DELETION OF Practice COMPLETE-------

##

## Rows: 5 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr (2): lcd, uts
## dbl (2): pracid, region
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## ----------LOAD OF TABLE Practice, FILE No. 1 SUCCESSFUL----------

##
## Rows: 5 Columns: 4-- Column specification --------------------------------------------------------
## Delimiter: "\t"
## chr (2): lcd, uts
## dbl (2): pracid, region
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## ----------LOAD OF TABLE Practice, FILE No. 2 SUCCESSFUL----------

##

## $database_location
## [1] "/home/j/jcfs2/Documents/Test"
##
## $files_in_location
##
## 2 /lustre/ahome3/j/jcfs2/R/x86_64-pc-linux-gnu-library/4.2/CPRDTools/extdata/1234.aa2_Extract_Clinica
## 3 /lustre/ahome3/j/jcfs2/R/x86_64-pc-linux-gnu-library/4.2/CPRDTools/extdata/1234.aa2_Extract_Clinica
## 4 /lustre/ahome3/j/jcfs2/R/x86_64-pc-linux-gnu-library/4.2/CPRDTools/extdata/1234.aa2_Extract_Clinica
## 5 /lustre/ahome3/j/jcfs2/R/x86_64-pc-linux-gnu-library/4.2/CPRDTools/extdata/1234.aa2_Extract_Clinica
## 1 /lustre/ahome3/j/jcfs2/R/x86_64-pc-linux-gnu-library/4.2/CPRDTools/extdata/1234.aa1_Extract_Practi
## 6 /lustre/ahome3/j/jcfs2/R/x86_64-pc-linux-gnu-library/4.2/CPRDTools/extdata/1234.aa2_Extract_Practi
##      table
## 2 Clinical
## 3 Clinical
## 4 Clinical
## 5 Clinical
```

```
## 1 Practice
## 6 Practice
##
## $tables_in_location
##      Table File_Count
## 1 Clinical          4
## 2 Practice          2
##
## $loaded_table
## [1] "Practice"
##
## $load_report
##      table load_total file_total size_Mb size_Gb
## 1 Clinical          2          2   5e-04       0
## 2 Practice          2          2   5e-04       0
##
## $database_tables
## [1] "Clinical"     "Patient"      "Practice"     "lookup_table"
##
## $load_start
## [1] "2022-10-04 12:28:58 BST"
##
## $load_end
## [1] "2022-10-04 12:28:59 BST"
##
## $load_time
## Time difference of 0.1845155 secs
```

Here as the practice table was previously loaded.

Updating a CPRD GOLD table

If you were to receive a new month or year of `patient` data, this would be append to the data already contained in the database using

```
update_table(db_path = DB.path,
             file_location = system.file("extdata", "aa1_Extract_Patient_001.zip", package = "CPRDTools"
             table_name = "PaTient")
```

Note that capitalisation of the table name for CPRD-specific tables in unimportant. In general, SQL and therefore SQLite is not case sensitive.

Updating an additional file

For peripheral or additional data, whether CPRD or non-CPRD specific, the use of the `load_additional` function is employed. This function loads a singular file into a table (new or existing).

```
load_additional(db_path = DB.path,
                file_location = system.file("extdata", "hes_patient_20336R.txt", package = "CPRDTools")
                type = ".txt",
                table_name = "hes_patient",
                overwrite = F)
```

```
## Rows: 24 Columns: 5
## -- Column specification --------------------------------------------------------
```

```
## Delimiter: "\t"
## dbl (5): patid, pracid, gen_hesid, n_patid_hes, match_rank
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.

## ----------LOAD OF TABLE hes_patient SUCCESSFUL----------

##

## $database_location
## [1] "/home/j/jcfs2/Documents/Test"
##
## $loaded_table
## [1] "hes_patient"
##
## $load_report
##         table size_Mb size_Gb
## 1 hes_patient   6e-04       0
##
## $database_tables
## [1] "Clinical"     "Patient"      "Practice"     "hes_patient"  "lookup_table"
##
## $load_start
## [1] "2022-10-04 12:28:59 BST"
##
## $load_end
## [1] "2022-10-04 12:28:59 BST"
##
## $load_time
## Time difference of 0.07646108 secs
```

The accepted files types are `.txt`, `.csv`, `.dta`, `.rds`, `.excel`, `.xl`, `.xls` and `.xlsx` with no additional input for file importing.

Updating an additional file

If sub-setting, manipulation or sheet selection, for instance, is required prior to the loading of a file into a table (new or existing) is required, it is first recommended to import the file in the usual manner into R. Once the data are in the format and layout that is required, the `load_global` function can be employed

```
## first import the life table data from .csv
life_table <- read.csv("/rfs/LRWE_Proj59/jcfs2/Database/Final.Datasets/ons_lt.csv")
##sub-set the data to the required fields
life_table <- life_table[,c("gender","year","age","pop_rate")]
## load into database
load_global(db_path = DB.path,
            file_to_load = life_table,
            table_name = "Life_table",
            overwrite = F)
```

```
## ----------LOAD OF TABLE Life_table SUCCESSFUL----------

##

## $database_location
## [1] "/home/j/jcfs2/Documents/Test"
```

```
##
## $loaded_table
## [1] "Life_table"
##
## $load_report
##        table size_Mb size_Gb
## 1 Life_table  0.1722   2e-04
##
## $database_tables
## [1] "Clinical"     "Life_table"   "Patient"      "Practice"     "hes_patient"
## [6] "lookup_table"
##
## $load_start
## [1] "2022-10-04 12:28:59 BST"
##
## $load_end
## [1] "2022-10-04 12:28:59 BST"
##
## $load_time
## Time difference of 0.02637196 secs
```

Extracting

With the database successfully loaded with our CPRD GOLD and associated data, we are now able to perform data extracts from the database. Before executing data extracts, it is *best practice* to first interrogate the databases for loaded table (ignoring that this is provided after each load iteration previously) and the for the structure and layout of a selection of tables.

List tables, fields and first rows

In order to list all loaded tables in the database, similar to the output in the `$loaded_tables`, returned when loading data via the data-loading function previously, specify

```
db_tables(db_path = DB.path)
```

```
## $database_location
## [1] "/home/j/jcfs2/Documents/Test"
##
## $tables
## [1] "Clinical"     "Life_table"   "Patient"      "Practice"     "hes_patient"
## [6] "lookup_table"
```

loaded are the `Clinical`, `Patient`, `Practice` and `hes_patient` tables.

To view the fields within the `Patient` table

```
table_fields(db_path = DB.path,
             table_name = "PaTient")
```

```
## $database_location
## [1] "/home/j/jcfs2/Documents/Test"
##
## $fields
##  [1] "patid"     "vmid"      "gender"    "yob"       "mob"       "marital"
##  [7] "famnum"    "CHSreg"    "CHSdate"   "prescr"    "capsup"    "frd"
```

```
## [13] "crd"       "regstat"   "reggap"    "internal"  "tod"       "toreason"
## [19] "deathdate" "accept"    "pracid"
```

To view the first 5 row of the `hes_patient` table

```
first_n(db_path = DB.path,
        table_name = "hes_patient",
        n = 5)
```

```
## $database_location
## [1] "/home/j/jcfs2/Documents/Test"
##
## $query
## [1] "SELECT * FROM hes_patient LIMIT 5;"
##
## $first_n_rows
##     patid pracid gen_hesid n_patid_hes match_rank
## 1 1000001      1   1236275           1          1
## 2 1001001      1   1235450           1          1
## 3 1002003      3   1235459           1          1
## 4 1003004      4   1236872           1          1
## 5 1004005      5   1235080           1          1
```

Extracting data

Data extraction can be performed via two functions (1) the user-input defined `query_builder` function and (2) `query_direct`, the more customisable and preferred method of data extraction.

The basic structure of the `query_bilder` function is

```
query_builder(db_path,
              unique_obs,
              field_list,
              from_table,
              join_table,
              join_fields,
              join_type,
              join_on,
              where_table,
              where_filter,
              order_column,
              order_type,
              limit_to)
```

This function allows for only a simple join between two tables, the main table (`from_table`) and secondary table (`join_table`). Fields can be extracted from both tables, including the field used to join the tables together (`join_on`), known as a *key* field. Additional filtering, sub-setting and restrictions of data can be done using where, order and limit inputs.

A simple extract is defined as

```
query_builder(db_path = DB.path,
              unique_obs = T,
              field_list = c("patid","gender","yob","deathdate"),
              from_table = "Patient",
              join_table = "hes_patient",
              join_fields = c("gen_hesid","match_rank"),
```

```
              join_on = "patid",
              limit_to = 5)
```

```
## $database_location
## [1] "/home/j/jcfs2/Documents/Test"
##
## $database_tables
## [1] "Clinical"     "Life_table"   "Patient"      "Practice"     "hes_patient"
## [6] "lookup_table"
##
## $query
## [1] "SELECT  DISTINCT  Patient.patid, Patient.gender, Patient.yob, Patient.deathdate, hes_patient.ge
##
## $query_data
##     patid gender  yob  deathdate gen_hesid match_rank
## 1 1000001      2 1999 2020-01-08   1236275          1
## 2 1001001      2 1989       <NA>   1235450          1
## 3 1002003      1 1993       <NA>   1235459          1
## 4 1003004      2 1984       <NA>   1236872          1
## 5 1004005      1 1982       <NA>   1235080          1
##
## $query_time
## Time difference of 0.005890131 secs
```

Here, a unique extract of `patid`, `gender`, `yob` and `deathdate` from the `Patient` table and `gen_hesid` and `match_rank` from the `hes_patient` table are extracted, joined together via the `patid` field and limited to the first 10 rows of outputted data. This data can be returned directly into a dataset using

```
data <- query_builder(db_path = DB.path,
                      unique_obs = T,
                      field_list = c("patid","gender","yob","deathdate"),
                      from_table = "Patient",
                      join_table = "hes_patient",
                      join_fields = c("gen_hesid","match_rank"),
                      join_on = "patid",
                      limit_to = 5)&query_data
```

Building on the previous extract, now filtered for only `gender==1` and ordered by `patid` in ascending order is defined as

```
query_builder(db_path = DB.path,
              unique_obs = T,
              field_list = c("paTid","gender","yob","deathdate"),
              from_table = "Patient",
              join_table = "hes_patient",
              join_fields = c("gen_hesid","match_rank"),
              join_type = "Left",
              join_on = "patiD",
              where_table = "PatieNt",
              where_filter = "geNder==1",
              order_field = "PATID",
              order_type = "aSC",
              limit_to = 5)
```

```
## $database_location
## [1] "/home/j/jcfs2/Documents/Test"
```

```
##
## $database_tables
## [1] "Clinical"     "Life_table"   "Patient"      "Practice"     "hes_patient"
## [6] "lookup_table"
##
## $query
## [1] "SELECT  DISTINCT  Patient.paTid, Patient.gender, Patient.yob, Patient.deathdate, hes_patient.ge
##
## $query_data
##     patid gender  yob deathdate gen_hesid match_rank
## 1 1002003      1 1993      <NA>   1235459          1
## 2 1004005      1 1982      <NA>   1235080          1
## 3 1005006      1 1989      <NA>   1235878          1
## 4 1007001      1 1987      <NA>   1236339          1
## 5 1009010      1 1986      <NA>   1235558          1
##
## $query_time
## Time difference of 0.006356716 secs
```

Note that SQLite is not case sensitive.

The preferred method for data extraction is through the direct execution of SQL code defined by the user. This allows for the most customisable, comprehensive and completed data extraction, providing all the functionality found in a SQLite `SELECT` statement.

A introductory overview of the `SELECT` statement can be found at https://www.sqlitetutorial.net/sqlite-select/2, with the basic syntax defined as

```
SELECT DISTINCT column_list
FROM table_list
  JOIN table ON join_condition
WHERE row_filter
ORDER BY column
LIMIT count OFFSET offset
GROUP BY column
HAVING group_filter;
```

To replicate the query executed using the `query_builder` function previously, the following is provided

```
query_direct(db_path = DB.path,
             query = "SELECT DISTINCT Patient.patid,
                                      Patient.gender,
                                      Patient.yob,
                                      Patient.deathdate,
                                      hes_patient.gen_hesid,
                                      hes_patient.match_rank
                      FROM Patient
                      LEFT JOIN hes_patient ON hes_patient.patid = Patient.patid
                      LIMIT 5 ;")
```

```
## $database_location
## [1] "/home/j/jcfs2/Documents/Test"
##
## $database_tables
## [1] "Clinical"     "Life_table"   "Patient"      "Practice"     "hes_patient"
## [6] "lookup_table"
##
```

```
## $query
## [1] "SELECT DISTINCT Patient.patid, \n                                                    Patient.gender, \n
##
## $query_data
##     patid gender  yob  deathdate gen_hesid match_rank
## 1 1000001      2 1999 2020-01-08   1236275          1
## 2 1001001      2 1989       <NA>   1235450          1
## 3 1002003      1 1993       <NA>   1235459          1
## 4 1003004      2 1984       <NA>   1236872          1
## 5 1004005      1 1982       <NA>   1235080          1
##
## $query_time
## Time difference of 0.001206636 secs
```

The `query` here is identical to the query returned in the `$query` output in the `query_builder` function. The table from which a field should be extracted is provided before the field such as `patient.yob`, extracting `yob` from `Patient`.

This query can be made more aesthetically pleasing by using an alias for the `Pateint` and `hes_patient` tables

```
query_direct(db_path = DB.path,
             query = "SELECT DISTINCT pat.patid,
                                      pat.gender as sex,
                                      pat.yob as bith_date,
                                      pat.deathdate as death,
                                      hes.gen_hesid as hesid,
                                      hes.match_rank
                      FROM Patient pat
                      LEFT JOIN hes_patient hes ON hes.patid = pat.patid
                      LIMIT 5 ;")$query_data
```

```
##     patid sex bith_date      death   hesid match_rank
## 1 1000001   2      1999 2020-01-08 1236275          1
## 2 1001001   2      1989       <NA> 1235450          1
## 3 1002003   1      1993       <NA> 1235459          1
## 4 1003004   2      1984       <NA> 1236872          1
## 5 1004005   1      1982       <NA> 1235080          1
```

To view the procedure in which a query will be executed, use `EXPLAIN QUERY PLAN`. This will provide an overview of the strategy or plan SQLite will use to execute a SQL query, providing

```
query_direct(db_path = DB.path,
             query = "EXPLAIN QUERY PLAN
                      SELECT DISTINCT pat.patid,
                                      pat.gender as sex,
                                      pat.yob as bith_date,
                                      pat.deathdate as death,
                                      hes.gen_hesid as hesid,
                                      hes.match_rank
                      FROM Patient pat
                      LEFT JOIN hes_patient hes ON hes.patid = pat.patid
                      LIMIT 5 ;")$query_data
```

```
##   id parent notused
## 1  5      0       0
## 2 22      0       0
```

```
## 3 53      0       0
##                                                          detail
## 1                                                      SCAN pat
## 2 SEARCH hes USING AUTOMATIC COVERING INDEX (patid=?) LEFT-JOIN
## 3                                  USE TEMP B-TREE FOR DISTINCT
```

Shows that first a full-table scan is used on the `Patient` table, followed by a subset search of the `hes_patient` table followed by a temporary b-tree structure used to defined the unique (`DISTINCT`) output data. A more complete overview of the `EXPLAIN QUERY PLAN` command can be found at https://www.sqlite.org/eqp.html.

Maintenace and other tasks

The removal and remaining of tables and the deletion of the databases, as well as query speed improvement function make up the final section of the `CPRDTools` package.

Speeding up a query

A method to improve the execution times of a query is indexing. Indexing a table or dataset creates a secondary data table containing the indexing variable and the row number (`rowid`) associated with that number. The index is initially search, returning the row numbers for the searched fields, before using these row numbers to return the required data from.

Consider the following query, returning data from the `Patient` and `Clinical` tables, where the `{sql} medcodes IN (1004,1046,1076,1146,1157,1314,1425,1454,1508,1842)`, defined as

```
query_direct(db_path = DB.path,
             query = "EXPLAIN QUERY PLAN
             SELECT pat.patid,
                    pat.yob,
                    pat.deathdate,
                    clin.eventdate,
                    clin.medcode
             FROM Patient pat
             INNER JOIN Clinical clin ON
                   pat.patid = clin.patid
             WHERE clin.medcode IN
             (1004,1046,1076,1146,1157,
              1314,1425,1454,1508,1842);")
```

```
## $database_location
## [1] "/home/j/jcfs2/Documents/Test"
##
## $database_tables
## [1] "Clinical"     "Life_table"   "Patient"      "Practice"     "hes_patient"
## [6] "lookup_table"
##
## $query
## [1] "EXPLAIN QUERY PLAN\n             SELECT pat.patid,\n                    pat.yob,\n
##
## $query_data
##   id parent notused                                          detail
## 1  3      0       0                                      SCAN clin
## 2 58      0       0 SEARCH pat USING AUTOMATIC COVERING INDEX (patid=?)
##
```

```
## $query_time
## Time difference of 0.001020908 secs
```

Executing the `EXPLAIN QUERY PLAN` command provides

```
##   id parent notused                                                detail
## 1  3      0       0                                          SCAN clin
## 2 58      0       0 SEARCH pat USING AUTOMATIC COVERING INDEX (patid=?)
```

Using a full-table scan of the clinical table for the desired `medcodes` before using an automatically generated, temporary index on `patid`.

An index can be defined on the `clinical` table using the `medcodes` as

```
add_index(db_path = DB.path,
          index_name = "clinical_index",
          index_table = "clinical",
          index_field = "medcode")
```

```
## $database_location
## [1] "/home/j/jcfs2/Documents/Test"
##
## $index
## [1] "clinical_index"
##
## $index_statement
## [1] "CREATE INDEX clinical_index ON clinical (medcode);"
##
## $query_time
## Time difference of 0.003913879 secs
```

Now the query plan shows

```
##   id parent notused                                                detail
## 1  4      0       0  SEARCH clin USING INDEX clinical_index (medcode=?)
## 2 60      0       0 SEARCH pat USING AUTOMATIC COVERING INDEX (patid=?)
```

First using the `clinical_index` to retrieve the desired `medcodes` before completing the rest of the query. The speed improvements are dependent on the complexity of the data, the query and the size of the data. There is a once-off execution time cost in indexing data but for large, complex datasets where queries rely on searching lists of medical codes, for instance, this may be beneficial to the user.

If a SQL statement must be passed directly to the database, specifically for maintenance or speed improvements, the `statement_direct` function is used. If the index created previously were to be deleted, the following is used

```
statement_direct(db_path = DB.path,
                 statement = "DROP INDEX clinical_index;")
```

```
## $database_location
## [1] "/home/j/jcfs2/Documents/Test"
##
## $database_tables
## [1] "Clinical"     "Life_table"   "Patient"      "Practice"     "hes_patient"
## [6] "lookup_table"
##
## $statement
## [1] "DROP INDEX clinical_index;"
##
```

```
## $query_time
## Time difference of 0.002125025 secs
```

`statement_direct` provides no output data and so should only be used for maintenance and speed improvement tasks such as `UPDATE, DELETE, INSERT INTO, DROP TABLE` etc.

Convert fields to dates

Often date fields are provided in datasets in the format of a date ('12/03/1989') but read as a character variable into the database. To convert a field to a date field use

```
data <- query_direct(db_path = DB.path,
                     query = "SELECT* FROM Practice;")$query_data
str(data)
```

```
## 'data.frame':    10 obs. of  4 variables:
##  $ pracid: num  1 2 3 4 5 6 7 8 9 10
##  $ region: num  8 8 7 8 7 5 5 1 1 2
##  $ lcd   : chr  "2015-11-21" "2016-05-09" "2015-11-20" "2016-08-26" ...
##  $ uts   : chr  "1999-09-11" "2001-10-01" "1989-11-17" "2001-02-23" ...
```

```
convert_data <- convert_dates(data = data,
                              date_fields = c("lcd","uts"),
                              format = "%Y-%m-%d")$convert_data
str(convert_data)
```

```
## 'data.frame':    10 obs. of  4 variables:
##  $ pracid: num  1 2 3 4 5 6 7 8 9 10
##  $ region: num  8 8 7 8 7 5 5 1 1 2
##  $ lcd   : Date, format: "2015-11-21" "2016-05-09" ...
##  $ uts   : Date, format: "1999-09-11" "2001-10-01" ...
```

Here the format provided is the format the fields are prior to conversion and not the desired output required, with the default set to '%d/%m/%Y'.

Maintenance

in order to rename a table in the database, use

```
rename_table(db_path = DB.path,
             old_name = "practice",
             new_name = "gp_pracs")
```

```
## $database_location
## [1] "/home/j/jcfs2/Documents/Test"
##
## $before_rename
## [1] "clinical"     "life_table"   "patient"      "practice"     "hes_patient"
## [6] "lookup_table"
##
## $old_table_name
## [1] "practice"
##
## $new_table_name
## [1] "gp_pracs"
##
```

```
## $after_rename
## [1] "Clinical"     "Life_table"   "Patient"      "gp_pracs"     "hes_patient"
## [6] "lookup_table"
```

While to remove an entire table from the database, the following is used

```
delete_table(db_path = DB.path,
           remove_tables = "gp_pracs")
```

```
## $database_location
## [1] "/home/j/jcfs2/Documents/Test"
##
## $before_drop
## [1] "clinical"     "life_table"   "patient"      "gp_pracs"     "hes_patient"
## [6] "lookup_table"
##
## $dropped_table
## [1] "gp_pracs"
##
## $after_drop
## [1] "Clinical"     "Life_table"   "Patient"      "hes_patient"  "lookup_table"
```

For the complete deletion of a database, use with caution

```
delete_db(db_path = DB.path)
```

This will require the inputting of confirmation of the deletion into the `Console`

```
## ----------WARNING, DELETION IN PROGRESS----------
```

```
##
```

```
## Press y+[enter] to proceed, n+[enter] to stop:
```

```
##
```

Conclusion

`CPRDTools` provides a comprehensive set of function to aid the general loading, maintenance and extraction of CPRD GOLD and its associated and peripheral data. These functions should provide a *toolkit* to any researcher, aiding in there ability to self-manage their data using an efficient and reliable SQLite data repository, providing tools to view, interrogate, load, extract, maintain, convert and delete in a single package.

References

Herrett, Emily, Arlene M. Gallagher, Krishnan Bhaskaran, Harriet Forbes, Rohini Mathur, Tjeerd van Staa, and Liam Smeeth. 2015. "Data Resource Profile: Clinical Practice Research Datalink (CPRD)." *International Journal of Epidemiology* 44 (3): 827–36. https://doi.org/10.1093/ije/dyv098.

Padmanabhan, Shivani. 2017. "CPRD Gold Data Specification." https://cprdcw.cprd.com/_docs/CPRD_GOLD_Full_Data_Specification_v2.0.pdf.

SQLite. 2017. *SQLite*. Charlotte, North Caolina: Hipp, Wyrick and Company, Inc. https://www.sqlite.org/index.html.