

An introduction to tidy R programming with the OMOP common data model

Edward Burn, Adam Black, Berta Raventós, Yuchen Guo, Mike Du, Kim López Güe

2023-08-23T00:00:00+01:00

Table of contents

Preface	4
1 Getting started	5
1.1 A first data analysis in R with a database	5
1.2 Getting set up	5
1.3 Taking a peek at the data	6
1.4 Inserting data into a database	7
1.5 Translation from R to SQL	8
1.6 Example analysis	9
1.7 Further reading	13
2 Creating a reference to a database using the OMOP common data model	14
2.1 Connecting to a database from R using DBI	14
2.2 Creating a reference to the OMOP common data model	14
2.3 Database snapshot	17
2.4 Further reading	17
3 Exploring the CDM	18
3.1 Counting people	18
3.2 Counting records	20
3.3 Working with dates	21
3.4 Statistical summaries	22
4 Adding cohorts to the CDM	25
4.1 Cohort definitions	25
4.2 Defining cohorts programmatically	25
4.3 Adding a cohort to the CDM	34
4.4 Adding a concept- based cohort to the CDM	34
5 Cohort attributes	35
6 Custom cohort creation	36
7 Reconnecting to a database with existing cohorts	37

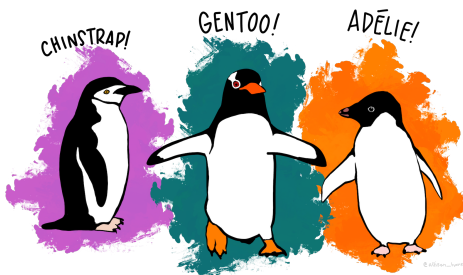
8 Describing a cohort	38
8.1 PatientProfiles	38
9 Sampling the cdm	39
9.1 Subset of CDM tables	39
9.2 Sample records in a table	39
9.3 Random sample of the cdm	39
9.4 Cohort-based subset of the cdm	39
10 Bringing data into memory	40
References	41

Preface

The source code for the book can be found at this [Github repository](#). Please open an issue there if you have a question or suggestion. Pull requests with suggested changes and additions are also most welcome.

1 Getting started

1.1 A first data analysis in R with a database



Artwork by @allison_horst

Before we start thinking about working with health care data spread across the OMOP common data model, let's first do a quick data analysis using a simpler dataset. For this we'll use data from [palmerpenguins package](#), which contains data on penguins collected from the [Palmer Station](#) in Antarctica.

1.2 Getting set up

Assuming that you have R and RStudio already set up, first we need to install a few packages not included in base R if we don't already have them.

```
install.packages("dplyr")
install.packages("ggplot2")
install.packages("DBI")
install.packages("duckdb")
install.packages("palmerpenguins")
```

Once installed, we can load them like so.

```
library(dplyr)
library(ggplot2)
library(DBI)
library(duckdb)
library(palmerpenguins)
```

1.3 Taking a peek at the data

We can get an overview of the data using the `glimpse()` command.

```
glimpse(penguins)
```

```
Rows: 344
Columns: 8
$ species      <fct> Adelie, Adelie, Adelie, Adelie, Adelie, Adelie, Adel~
$ island       <fct> Torgersen, Torgersen, Torgersen, Torgersen, Torgerse~
$ bill_length_mm <dbl> 39.1, 39.5, 40.3, NA, 36.7, 39.3, 38.9, 39.2, 34.1, ~
$ bill_depth_mm <dbl> 18.7, 17.4, 18.0, NA, 19.3, 20.6, 17.8, 19.6, 18.1, ~
$ flipper_length_mm <int> 181, 186, 195, NA, 193, 190, 181, 195, 193, 190, 186~
$ body_mass_g   <int> 3750, 3800, 3250, NA, 3450, 3650, 3625, 4675, 3475, ~
$ sex          <fct> male, female, female, NA, female, male, female, male~
$ year         <int> 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007, 2007~
```

Or we could take a look at the first rows of the data using `head()`

```
head(penguins, 5)
```

```
# A tibble: 5 x 8
  species island bill_length_mm bill_depth_mm flipper_l~1 body_~2 sex year
  <fct>   <fct>         <dbl>         <dbl>         <int>   <int> <fct> <int>
1 Adelie Torgersen      39.1           18.7           181     3750 male  2007
2 Adelie Torgersen      39.5           17.4           186     3800 fema~  2007
3 Adelie Torgersen      40.3            18            195     3250 fema~  2007
4 Adelie Torgersen      NA              NA              NA        NA <NA>   2007
5 Adelie Torgersen      36.7           19.3           193     3450 fema~  2007
# ... with abbreviated variable names 1: flipper_length_mm, 2: body_mass_g
```

1.4 Inserting data into a database

Let's put our penguins data into a duckdb database. We create the duckdb database, add the penguins data, and then create a reference to the table containing the data.

```
db<-dbConnect(duckdb::duckdb(), dbdir=":memory:")
dbWriteTable(db, "penguins", penguins)
penguins_db<-tbl(db, "penguins")
```

Now the data is in a database we could use SQL to get the first rows that we saw before

```
dbGetQuery(db, "SELECT * FROM penguins LIMIT 5")
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
1	Adelie	Torgersen	39.1	18.7	181	3750
2	Adelie	Torgersen	39.5	17.4	186	3800
3	Adelie	Torgersen	40.3	18.0	195	3250
4	Adelie	Torgersen	NA	NA	NA	NA
5	Adelie	Torgersen	36.7	19.3	193	3450

	sex	year
1	male	2007
2	female	2007
3	female	2007
4	<NA>	2007
5	female	2007

But we could also use the same R code as before

```
head(penguins_db, 5)
```

```
# Source:   SQL [5 x 8]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/:memory:]
  species island  bill_length_mm bill_depth_mm flipper_l~1 body_~2 sex    year
  <fct>   <fct>      <dbl>         <dbl>         <int>    <int> <fct> <int>
1 Adelie  Torgersen    39.1           18.7           181      3750 male   2007
2 Adelie  Torgersen    39.5           17.4           186      3800 fema~  2007
3 Adelie  Torgersen    40.3           18             195      3250 fema~  2007
4 Adelie  Torgersen    NA             NA             NA        NA <NA>   2007
5 Adelie  Torgersen    36.7           19.3           193      3450 fema~  2007
# ... with abbreviated variable names 1: flipper_length_mm, 2: body_mass_g
```

1.5 Translation from R to SQL

The magic here is provided by dbplyr which takes the R code and converts it into SQL, which in this case looks like

```
head(penguins_db, 1) %>%  
  show_query()
```

```
<SQL>  
SELECT *  
FROM penguins  
LIMIT 1
```

More complicated SQL can also be written in what might be familiar dplyr code, for example

```
penguins_db %>%  
  group_by(species) %>%  
  summarise(min_bill_length_mm=min(bill_length_mm),  
            median_bill_length_mm=median(bill_length_mm),  
            max_bill_length_mm=max(bill_length_mm)) %>%  
  mutate(min_max_bill_length_mm=paste0(min_bill_length_mm,  
                                       " to ",  
                                       max_bill_length_mm)) %>%  
  select("species",  
        "median_bill_length_mm",  
        "min_max_bill_length_mm")
```

```
# Source:   SQL [3 x 3]  
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/:memory:]  
  species   median_bill_length_mm min_max_bill_length_mm  
  <fct>                <dbl> <chr>  
1 Adelie                38.8 32.1 to 46.0  
2 Gentoo                47.3 40.9 to 59.6  
3 Chinstrap            49.6 40.9 to 58.0
```

with the corresponding SQL looking like

```
penguins_db %>%  
  group_by(species) %>%  
  summarise(min_bill_length_mm=min(bill_length_mm),
```



```

        median_bill_length_mm=median(bill_length_mm),
        max_bill_length_mm=max(bill_length_mm)) %>%
mutate(min_max_bill_length_mm=paste0(min, " to ", max)) %>%
select("species",
       "median_bill_length_mm",
       "min_max_bill_length_mm") %>%
show_query()

```

<SQL>

```

SELECT
  species,
  median_bill_length_mm,
  CONCAT_WS(' ', .Primitive("min"), ' to ', .Primitive("max")) AS min_max_bill_length_mm
FROM (
  SELECT
    species,
    MIN(bill_length_mm) AS min_bill_length_mm,
    PERCENTILE_CONT(0.5) WITHIN GROUP (ORDER BY bill_length_mm) AS median_bill_length_mm,
    MAX(bill_length_mm) AS max_bill_length_mm
  FROM penguins
  GROUP BY species
) q01

```

1.6 Example analysis

Let's start by getting a count by species

```

penguins_db %>%
  group_by(species) %>%
  count()

```

```

# Source:   SQL [3 x 2]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/:memory:]
  species      n
  <fct>      <dbl>
1 Adelie     152
2 Gentoo     124
3 Chinstrap   68

```

Now suppose we are particularly interested in the body mass variable. We can first notice that there are a couple of missing records for this.

```
penguins_db %>%  
  mutate(missing_body_mass_g = if_else(  
    is.na(body_mass_g), 1, 0  
  )) %>%  
  group_by(species, missing_body_mass_g) %>%  
  tally()
```

```
# Source:   SQL [5 x 3]  
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/:memory:]  
# Groups:   species  
  species   missing_body_mass_g      n  
  <fct>                <dbl> <dbl>  
1 Adelie                0    151  
2 Adelie                1      1  
3 Gentoo                0    123  
4 Gentoo                1      1  
5 Chinstrap            0     68
```

We can get the mean for each of the species (dropping those two missing records).

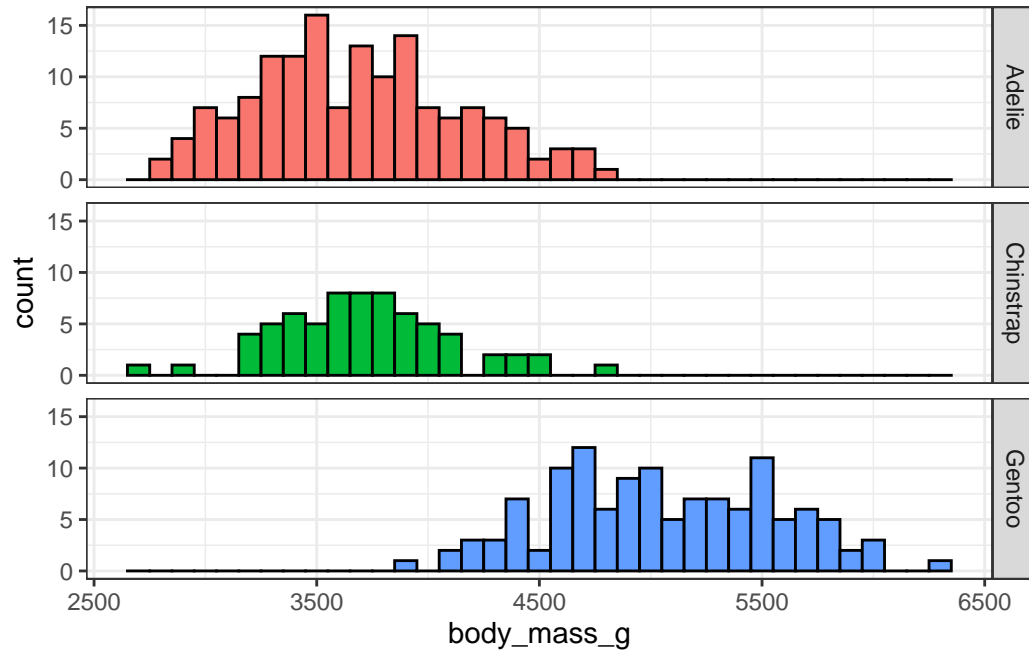
```
penguins_db %>%  
  group_by(species) %>%  
  summarise(mean_body_mass_g = round(mean(body_mass_g, na.rm=TRUE), 0))
```

```
# Source:   SQL [3 x 2]  
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/:memory:]  
  species   mean_body_mass_g  
  <fct>                <dbl>  
1 Adelie                3701  
2 Gentoo                5076  
3 Chinstrap            3733
```

We can then also do a histogram for each of the species. For this we need to bring the data into R so that we can work with `ggplot()`, and we use `collect()` to do this.

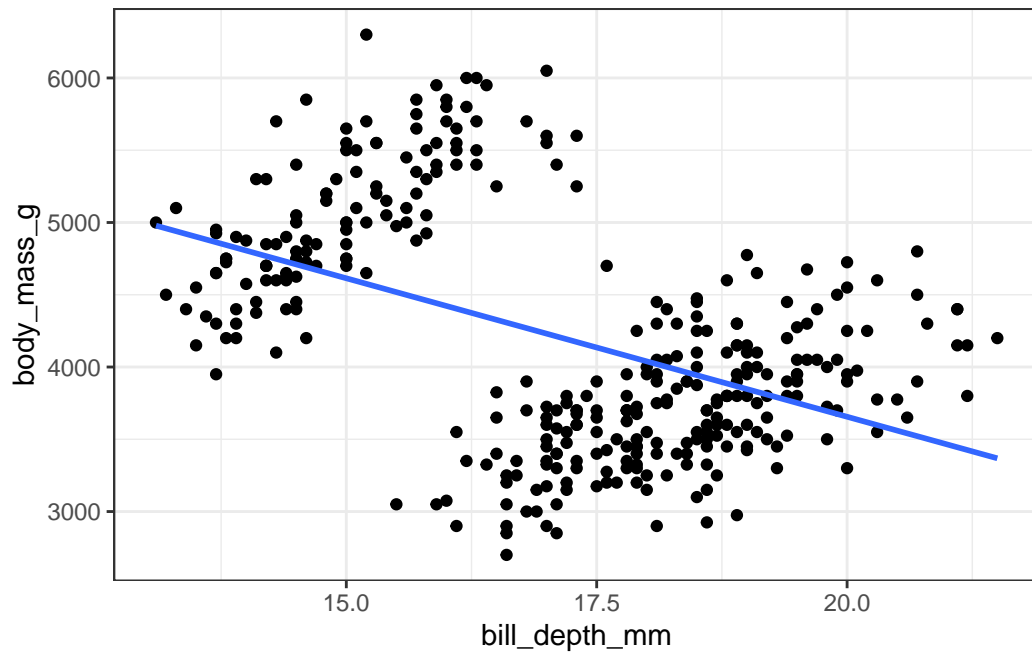
```
penguins_db %>%  
  collect() %>%
```

```
ggplot(aes(group=species, fill=species))+
  facet_grid(species~ .) +
  geom_histogram(aes(body_mass_g), colour="black", binwidth = 100)+
  theme_bw()+
  theme(legend.position = "none")
```



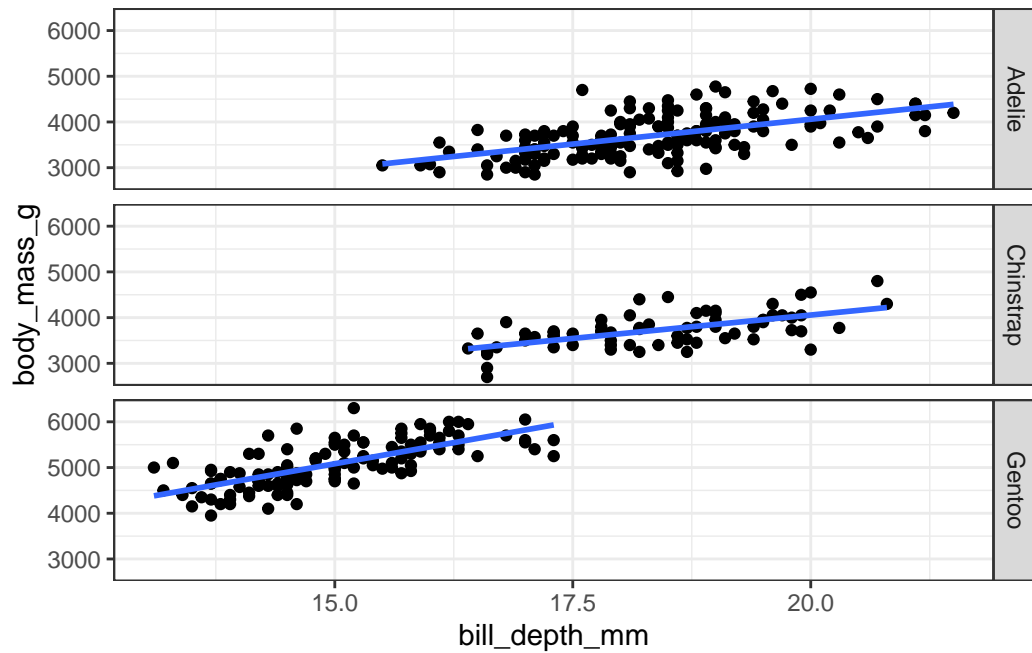
How about the relationship between body mass and bill depth?

```
penguins %>%
  collect() %>%
  ggplot(aes(x=bill_depth_mm,y=body_mass_g))+
  geom_point()+
  geom_smooth(method="lm",se=FALSE )+
  theme_bw()+
  theme(legend.position = "none")
```



But what about by species?

```
penguins %>%  
  collect() %>%  
  ggplot(aes(x=bill_depth_mm,y=body_mass_g))+  
  facet_grid(species~ .) +  
  geom_point()+  
  geom_smooth(method="lm",se=FALSE )+  
  theme_bw()+  
  theme(legend.position = "none")
```



As well as having an example of working with data in database from R, you also have an example of [Simpson's paradox](#)! And now we've reached the end of this example, we can close the database like so

1.7 Further reading

- [R for Data Science \(Chapter 13: Relational data\)](#)
- [Writing SQL with dbplyr](#)
- [Data Carpentry: SQL databases and R](#)

2 Creating a reference to a database using the OMOP common data model

2.1 Connecting to a database from R using DBI

Database connections from R can be made using the [DBI package](#). The back-end for DBI is facilitated by database specific driver packages. As an example, lets say we want to work with a local duckdb from R. In this case the we can use the duckdb R package as the driver.

```
library(DBI)
db<-dbConnect(duckdb::duckdb(), dbdir=":memory:")
```

If we instead wanted to connect to other database management systems, these connections would be supported by the associated back-end packages and could look something like the below example for Postgres:

```
# Postgres
db <- DBI::dbConnect(RPostgres::Postgres(),
                     dbname = Sys.getenv("CDM5_POSTGRESQL_DBNAME"),
                     host = Sys.getenv("CDM5_POSTGRESQL_HOST"),
                     user = Sys.getenv("CDM5_POSTGRESQL_USER"),
                     password = Sys.getenv("CDM5_POSTGRESQL_PASSWORD"))
```

2.2 Creating a reference to the OMOP common data model

As seen in the previous chapter, once a connection to the database has been created then we could create references to the various tables in the database and build queries using in a familiar dplyr style. However, as we already know what the structure of the OMOP CDM looks like, we can avoid the overhead of building *ad hoc* references by instead using the CDMConnector package to quickly create a reference to the OMOP CDM data as a whole.

If you don't already have it installed, the first step would be to install CDMConnector from CRAN.

```
install.packages("CDMConnector")
```

Once we have it installed, we can then load it as with other R packages.

```
library(CDMConnector)
```

For this example, we'll use the Eunomia example data contained in a duckdb database. First we need to download the data. And once downloaded, make sure to add the path to your Renviron.

```
# change pathToData to the location you want to save the data
CDMConnector::downloadEunomiaData(
  pathToData = here::here(),
  overwrite = TRUE
)
# once downloaded, save your pathToData to your Renviron (and then restart R)
# EUNOMIA_DATA_FOLDER="....."
```

```
db <- DBI::dbConnect(duckdb::duckdb(),
  dbdir = CDMConnector::eunomia_dir())
cdm <- CDMConnector::cdm_from_con(con = db,
  cdm_schema = "main")

cdm
```

```
# OMOP CDM reference (tbl_duckdb_connection)
```

Tables: person, observation_period, visit_occurrence, visit_detail, condition_occurrence, drug_exposure

Once we have created the our reference to the overall OMOP CDM, we can reference specific tables using the “\$” operator or [[“”]].

```
cdm$observation_period
```

```
# Source:   table<main.observation_period> [?? x 5]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmp]
  observation_period_id person_id observation_period_start~1 observat~2 perio~3
      <dbl>          <dbl> <date>                  <date>          <dbl>
1           6           6 1963-12-31          2007-02-06  4.48e7
2          13          13 2009-04-26          2019-04-14  4.48e7
3          27          27 2002-01-30          2018-11-21  4.48e7
```

```

4          16          16 1971-10-14          2017-11-02 4.48e7
5          55          55 2009-05-30          2019-03-23 4.48e7
6          60          60 1990-11-21          2019-01-23 4.48e7
7          42          42 1909-11-03          2019-03-13 4.48e7
8          33          33 1986-05-12          2018-09-10 4.48e7
9          18          18 1965-11-17          2018-11-07 4.48e7
10         25          25 2007-03-18          2019-04-07 4.48e7
# ... with more rows, and abbreviated variable names
#   1: observation_period_start_date, 2: observation_period_end_date,
#   3: period_type_concept_id

```

```
cdm[["observation_period"]]
```

```

# Source:   table<main.observation_period> [?? x 5]
# Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmp
  observation_period_id person_id observation_period_start~1 observat~2 perio~3
                <dbl>      <dbl> <date>                <date>      <dbl>
1                 6          6 1963-12-31          2007-02-06 4.48e7
2                13         13 2009-04-26          2019-04-14 4.48e7
3                27         27 2002-01-30          2018-11-21 4.48e7
4                16         16 1971-10-14          2017-11-02 4.48e7
5                55         55 2009-05-30          2019-03-23 4.48e7
6                60         60 1990-11-21          2019-01-23 4.48e7
7                42         42 1909-11-03          2019-03-13 4.48e7
8                33         33 1986-05-12          2018-09-10 4.48e7
9                18         18 1965-11-17          2018-11-07 4.48e7
10               25         25 2007-03-18          2019-04-07 4.48e7
# ... with more rows, and abbreviated variable names
#   1: observation_period_start_date, 2: observation_period_end_date,
#   3: period_type_concept_id

```

When we created our reference we could have also specified a subset of cdm tables that we want to read:

```

cdm <- CDMConnector::cdm_from_con(db,
                                   cdm_tables = c("person","observation_period"))
cdm

```

```
# OMOP CDM reference (tbl_duckdb_connection)
```

```
Tables: person, observation_period
```


Moreover, we can also specify a write schema and the tables that we are interested in it when creating our reference. For example, if we wanted to create a reference to the person and observation period tables in the common data model along with cohort tables in a schema we have write access to, we could do this like so:

```
cdm <- CDMConnector::cdm_from_con(db,
  cdm_schema = "main",
  cdm_tables = c("person", "observation_period"),
  write_schema = "results",
  cohort_tables = c("exposure_cohort", "outcome_cohort"))
```

2.3 Database snapshot

We can also use `CDMConnector` to provide a summary of the metadata for the OMOP CDM data we have connected to

```
cdm_from_con(con = db,
  cdm_schema = "main") %>%
  snapshot() %>%
  glimpse()
```

List of 7

```
$ cdm_source_name      : chr "Synthea synthetic health database"
$ cdm_version          : chr "v5.3.1"
$ cdm_holder           : chr "OHDSI Community"
$ cdm_release_date     : Date[1:1], format: "2019-05-25"
$ vocabulary_version   : chr "v5.0 18-JAN-19"
$ person_cnt           : num 2694
$ observation_period_cnt: num 5343
- attr(*, "class")= chr "cdm_snapshot"
```

2.4 Further reading

- [CDMConnector package](#)

3 Exploring the CDM

Let's first connect again to our Eunomia data and create the reference to the common data model.

```
library(dbplyr)
library(dplyr)
library(CDMConnector)
library(ggplot2)
```

3.1 Counting people

The OMOP CDM is person-centric, with the person table containing records to uniquely identify each person in the database. As each row refers to a unique person, we can quickly get a count of the number of individuals in the database like so

```
cdm$person %>%
  count() %>%
  pull()
```

```
[1] 2694
```

The person table also contains some demographic information, including a `gender_concept_id` for each person. We can get a count grouped by this variable, but as this uses a concept we'll also need to join to the concept table to get the corresponding concept name for each concept id.

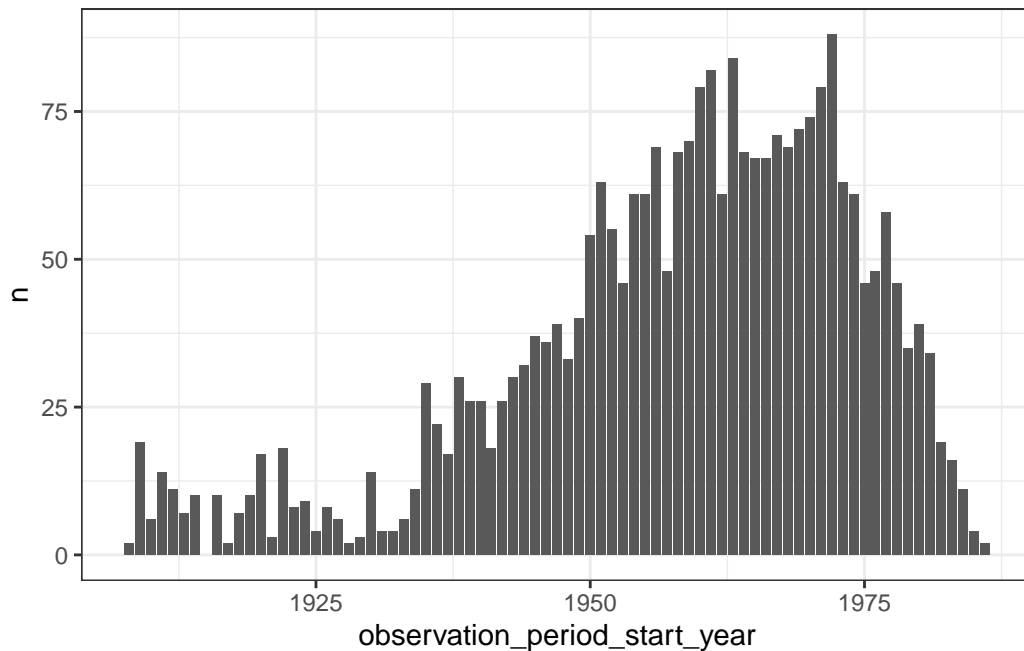
```
cdm$person %>%
  group_by(gender_concept_id) %>%
  count() %>%
  left_join(cdm$concept,
            by=c("gender_concept_id" = "concept_id")) %>%
    select("gender_concept_id", "concept_name", "n") %>%
  collect()
```

```
# A tibble: 2 x 3
  gender_concept_id concept_name      n
      <dbl> <chr>      <dbl>
1         8532 FEMALE      1373
2         8507 MALE       1321
```

The observation period table contains records indicating spans of time over which clinical events can be reliably observed for the people in the person table. Someone can potentially have multiple observation periods. So say we wanted a count of people grouped by the year during which their first observation period started. We could do this as below (note the use of `compute()` to store the results of the first query in a temporary table in the database)

```
first_observation_period <- cdm$observation_period %>%
  group_by(person_id) %>%
  mutate(seq = dplyr::row_number()) %>%
  filter(seq==1) %>%
  compute()

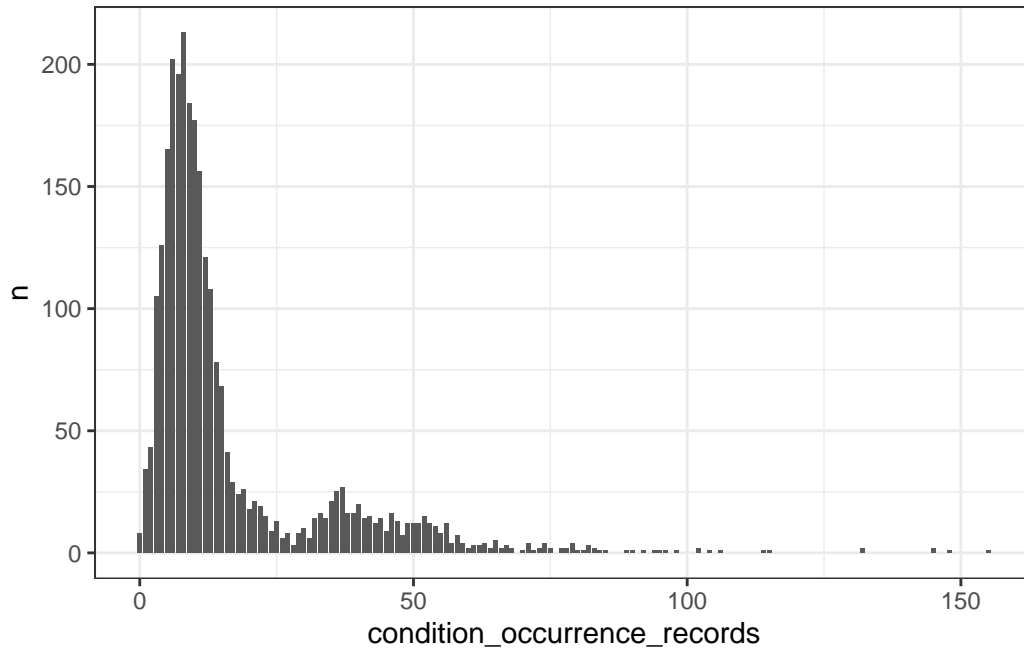
cdm$person %>%
  left_join(first_observation_period,
            by = "person_id") %>%
  mutate(observation_period_start_year=year(observation_period_start_date)) %>%
  group_by(observation_period_start_year) %>%
  count() %>%
  collect() %>%
  ggplot() +
  geom_col(aes(observation_period_start_year, n)) +
  theme_bw()
```



3.2 Counting records

Number of drug exposure records per person

```
cdm$person %>%
  left_join(cdm$measurement %>%
    group_by(person_id) %>%
    count(name = "condition_occurrence_records",
          by="person_id") %>%
    mutate(condition_occurrence_records = if_else(
      is.na(condition_occurrence_records), 0,
      condition_occurrence_records)) %>%
    group_by(condition_occurrence_records) %>%
    count() %>%
    collect() %>%
  ggplot() +
  geom_col(aes(condition_occurrence_records, n)) +
  theme_bw()
```

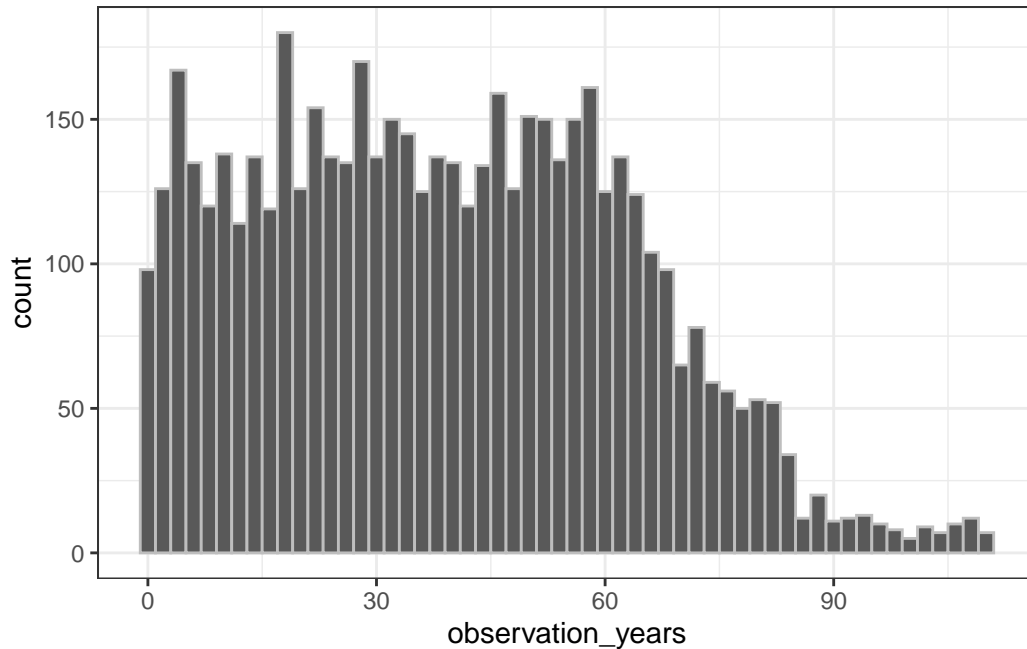


3.3 Working with dates

Dates are supported somewhat inconsistently by dbplyr, but CDMConnector provides some functions that provide more general support. We can use the `datediff` function from CDMConnector for example to calculate the difference between two dates. We can use this, for example, to get the number of years people's observation period last for.

```
cdm$observation_period %>%
  dplyr::mutate(observation_years =
    !!CDMConnector::datediff("observation_period_start_date",
                             "observation_period_end_date",
                             interval = "year")) %>%

  collect() %>%
  ggplot() +
  geom_histogram(aes(observation_years),
                 binwidth=2, colour="grey") +
  theme_bw()
```



3.4 Statistical summaries

We can also use summarise for various other calculations

```
cdm$person %>%
  summarise(min_year_of_birth = min(year_of_birth, na.rm=TRUE),
            q05_year_of_birth = quantile(year_of_birth, 0.05, na.rm=TRUE),
            mean_year_of_birth = round(mean(year_of_birth, na.rm=TRUE),0),
            median_year_of_birth = median(year_of_birth, na.rm=TRUE),
            q95_year_of_birth = quantile(year_of_birth, 0.95, na.rm=TRUE),
            max_year_of_birth = max(year_of_birth, na.rm=TRUE)) %>%
  glimpse()
```

Rows: ??

Columns: 6

Database: DuckDB 0.5.0 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\RtmpA

\$ min_year_of_birth <dbl> 1908

\$ q05_year_of_birth <dbl> 1922

\$ mean_year_of_birth <dbl> 1958

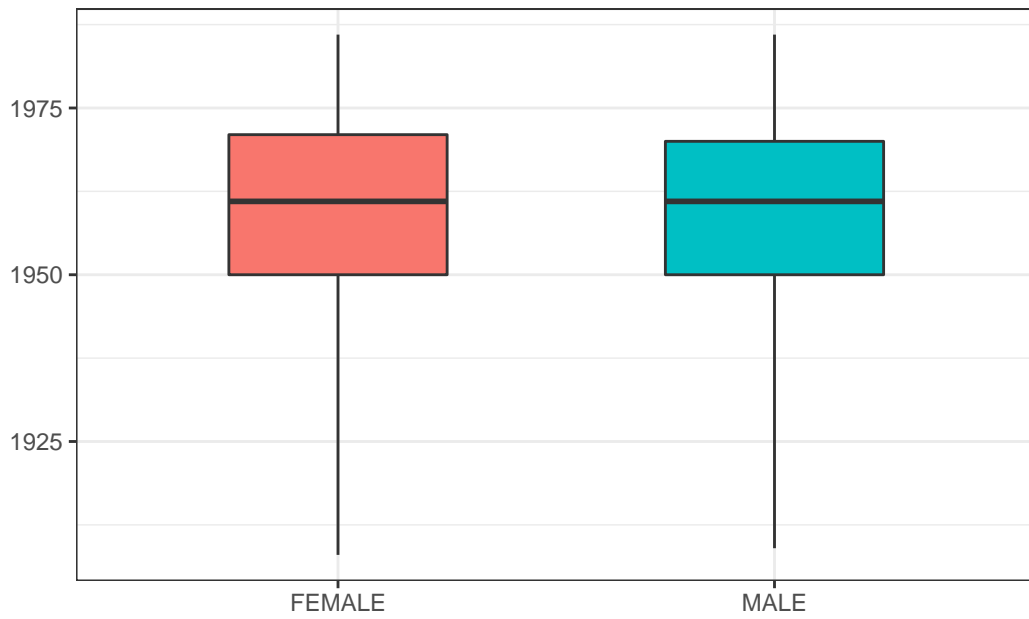
\$ median_year_of_birth <dbl> 1961

\$ q95_year_of_birth <dbl> 1979

```
$ max_year_of_birth    <dbl> 1986
```

As we've seen before, we can also quickly get results for various groupings or restrictions

```
cdm$person %>%
  group_by(gender_concept_id) %>%
  summarise(min_year_of_birth = min(year_of_birth, na.rm=TRUE),
            q25_year_of_birth = quantile(year_of_birth, 0.25, na.rm=TRUE),
            median_year_of_birth = median(year_of_birth, na.rm=TRUE),
            q75_year_of_birth = quantile(year_of_birth, 0.75, na.rm=TRUE),
            max_year_of_birth = max(year_of_birth, na.rm=TRUE)) %>%
  left_join(cdm$concept,
            by=c("gender_concept_id" = "concept_id")) %>%
  collect() %>%
  ggplot(aes(x = concept_name, group = concept_name,
            fill = concept_name)) +
  geom_boxplot(aes(
    lower = q25_year_of_birth,
    upper = q75_year_of_birth,
    middle = median_year_of_birth,
    ymin = min_year_of_birth,
    ymax = max_year_of_birth),
    stat = "identity", width = 0.5) +
  theme_bw()+
  theme(legend.position = "none") +
  xlab("")
```



4 Adding cohorts to the CDM

4.1 Cohort definitions

When performing research with the OMOP common data model we often want to identify groups of individuals who share some set of characteristics. The criteria for including individuals can range from the seemingly simple (e.g. people diagnosed with asthma) to the much more complicated (e.g. adults diagnosed with asthma who had a year of prior observation time in the database prior to their diagnosis, had no prior history of chronic obstructive pulmonary disease, and no history of use of short-acting beta-agonists). The set of people we identify are cohorts, and the OMOP CDM has a specific structure by which they can be represented, with a cohort table having four required fields: 1) cohort definition id (a unique identifier for each cohort), subject id (a foreign key to the subject in the cohort - typically referring to records in the person table), cohort start date, and cohort end date.

Cohorts can be defined using entirely bespoke code (so long as the output fits the cohort table specification). However because cohort definitions often follow a similar logic, tools have also been developed to facilitate cohort creation. In particular, [ATLAS](#) provides a graphical user interface which can be used to create cohort definitions that are expressed as JSON which can subsequently be rendered to SQL. The Capr R package, used below, provides a means of defining the JSON via R code instead.

4.2 Defining cohorts programmatically

We can define a cohort programmatically using the Capr package. In addition the Codelist-Generator package can be used to help find the codes to use in our cohort definitions.

Let's load the required packages and connect to the Eunomia data again.

```
library(CDMConnector)
library(dplyr)
library(Capr)
library(CodelistGenerator)

con <- DBI::dbConnect(duckdb::duckdb(), eunomia_dir())
```

Download completed!

```
cdm <- CDMConnector::cdm_from_con(  
  con = con,  
  cdm_schema = "main",  
  write_schema = "main"  
)
```

Say we want to create a cohort of people with a gastrointestinal hemorrhage. We'll start by getting the code that represents "gastrointestinal hemorrhage"

```
gibleed_codes <- getCandidateCodes(cdm = cdm,  
  keywords = "gastrointestinal hemorrhage",  
  domains = "condition",  
  exactMatch = TRUE,  
  includeDescendants = FALSE)  
gibleed_codes %>%  
  glimpse()
```

```
Rows: 1  
Columns: 6  
$ concept_id      <int> 192671  
$ concept_name    <chr> "Gastrointestinal hemorrhage"  
$ domain_id      <chr> "condition"  
$ concept_class_id <chr> "clinical finding"  
$ vocabulary_id   <chr> "snomed"  
$ found_from      <chr> "From initial search"
```

```
gibleed_concept_set <- cs(descendants(gibleed_codes$concept_id))
```

💡 Finding appropriate codes

The above gives the impression that identifying concepts that represent a particular clinical idea is straightforward. In practice, however, this is rarely the case. Identifying the codes that could represent a condition and then choosing which does though is typically a time consuming task

```
getCandidateCodes(cdm = cdm,
                  keywords = "fracture",
                  domains = "condition",
                  includeDescendants = TRUE)
```

A tibble: 9 x 6

	concept_id	concept_name	domain_id	concept_class_id	vocabulary_id	found_from
	<int>	<chr>	<chr>	<chr>	<chr>	<chr>
1	4048695	Fracture of ve~	condition	clinical finding	snomed	From init~
2	4142905	Fracture of rib	condition	clinical finding	snomed	From init~
3	4278672	Fracture of fo~	condition	clinical finding	snomed	From init~
4	4237458	Fracture of cl~	condition	clinical finding	snomed	From init~
5	4230399	Closed fractur~	condition	clinical finding	snomed	From init~
6	40480160	Pathological f~	condition	clinical finding	snomed	From init~
7	4066995	Fracture of ve~	condition	clinical finding	snomed	From init~
8	4059173	Fracture of an~	condition	clinical finding	snomed	From init~
9	4134304	Fracture sublu~	condition	clinical finding	snomed	From init~

Once we've identified our codes we'll create a concept set that includes this code or any of its descendants.

```
gibleed_concept_set <- cs(descendants(gibleed_codes$concept_id))
```

We can now use this concept set in a cohort definition. We'll look for anyone with a corresponding record in the condition occurrence table. We'll also require that this is their first such record.

```
entry_criteria <- entry(
  condition(gibleed_concept_set),
  primaryCriteriaLimit = "First"
)

gibleed_cohort_definition <- cohort(entry = entry_criteria)
```

We could though make things a little more complicated. What if we wanted to exclude anyone with rheumatoid arthritis (regardless of when they were diagnosed). To do this we'll first need to create another concept set, this time for rheumatoid arthritis.

```
rheumatoid_arthritis_codes <- getCandidateCodes(cdm = cdm,
                                                  keywords = "rheumatoid arthritis",
                                                  domains = "condition",
```

```

        exactMatch = TRUE,
        includeDescendants = FALSE)
rheumatoid_arthritis_codes %>%
  glimpse()

```

```

Rows: 1
Columns: 6
$ concept_id      <int> 80809
$ concept_name    <chr> "Rheumatoid arthritis"
$ domain_id      <chr> "condition"
$ concept_class_id <chr> "clinical finding"
$ vocabulary_id   <chr> "snomed"
$ found_from     <chr> "From initial search"

```

```

rheumatoid_arthritis_concept_set <- cs(descendants(
  rheumatoid_arthritis_codes$concept_id))

```

💡 JSON representation of a concept set

```

cat(as.json(gibleed_concept_set))

{
  "items": [
    {
      "concept": {
        "CONCEPT_ID": 192671,
        "CONCEPT_NAME": "",
        "STANDARD_CONCEPT": "",
        "STANDARD_CONCEPT_CAPTION": "",
        "INVALID_REASON": "",
        "INVALID_REASON_CAPTION": "",
        "CONCEPT_CODE": "",
        "DOMAIN_ID": "",
        "VOCABULARY_ID": "",
        "CONCEPT_CLASS_ID": ""
      },
      "isExcluded": false,
      "includeDescendants": true,
      "includeMapped": false
    }
  ]
}

```

```

    ]
  }

  cat(as.json(rheumatoid_arthritis_concept_set))

{
  "items": [
    {
      "concept": {
        "CONCEPT_ID": 80809,
        "CONCEPT_NAME": "",
        "STANDARD_CONCEPT": "",
        "STANDARD_CONCEPT_CAPTION": "",
        "INVALID_REASON": "",
        "INVALID_REASON_CAPTION": "",
        "CONCEPT_CODE": "",
        "DOMAIN_ID": "",
        "VOCABULARY_ID": "",
        "CONCEPT_CLASS_ID": ""
      },
      "isExcluded": false,
      "includeDescendants": true,
      "includeMapped": false
    }
  ]
}

```

And now we can add this exclusion criteria to our cohort definition.

```

gibleed_no_RA_cohort_definition <- cohort(
  entry = entry_criteria,
  attrition = attrition(
    exactly(0,
      condition(rheumatoid_arthritis_concept_set),
      duringInterval(eventStarts(-Inf, Inf))))
)

```

💡 JSON representation of a cohort

```
cat(as.json((gibleed_cohort_definition)))

{
  "ConceptSets": [
    {
      "id": 0,
      "name": "",
      "expression": {
        "items": [
          {
            "concept": {
              "CONCEPT_ID": 192671,
              "CONCEPT_NAME": "",
              "STANDARD_CONCEPT": "",
              "STANDARD_CONCEPT_CAPTION": "",
              "INVALID_REASON": "",
              "INVALID_REASON_CAPTION": "",
              "CONCEPT_CODE": "",
              "DOMAIN_ID": "",
              "VOCABULARY_ID": "",
              "CONCEPT_CLASS_ID": ""
            },
            "isExcluded": false,
            "includeDescendants": true,
            "includeMapped": false
          }
        ]
      }
    }
  ],
  "PrimaryCriteria": {
    "CriteriaList": [
      {
        "ConditionOccurrence": {
          "CodesetId": 0
        }
      }
    ]
  },
  "ObservationWindow": {
```

```

        "PriorDays": 0,
        "PostDays": 0
    },
    "PrimaryCriteriaLimit": {
        "Type": "First"
    }
},
"QualifiedLimit": {
    "Type": "First"
},
"ExpressionLimit": {
    "Type": "First"
},
"InclusionRules": [],
"CensoringCriteria": [],
"CollapseSettings": {
    "CollapseType": "ERA",
    "EraPad": 0
},
"CensorWindow": {},
"cdmVersionRange": ">=5.0.0"
}

cat(as.json((gibleed_no_RA_cohort_definition)))

{
  "ConceptSets": [
    {
      "id": 0,
      "name": "",
      "expression": {
        "items": [
          {
            "concept": {
              "CONCEPT_ID": 192671,
              "CONCEPT_NAME": "",
              "STANDARD_CONCEPT": "",
              "STANDARD_CONCEPT_CAPTION": "",
              "INVALID_REASON": "",
              "INVALID_REASON_CAPTION": "",
              "CONCEPT_CODE": "",

```

```

        "DOMAIN_ID": "",
        "VOCABULARY_ID": "",
        "CONCEPT_CLASS_ID": ""
    },
    "isExcluded": false,
    "includeDescendants": true,
    "includeMapped": false
}
]
}
},
{
    "id": 1,
    "name": "",
    "expression": {
        "items": [
            {
                "concept": {
                    "CONCEPT_ID": 80809,
                    "CONCEPT_NAME": "",
                    "STANDARD_CONCEPT": "",
                    "STANDARD_CONCEPT_CAPTION": "",
                    "INVALID_REASON": "",
                    "INVALID_REASON_CAPTION": "",
                    "CONCEPT_CODE": "",
                    "DOMAIN_ID": "",
                    "VOCABULARY_ID": "",
                    "CONCEPT_CLASS_ID": ""
                },
                "isExcluded": false,
                "includeDescendants": true,
                "includeMapped": false
            }
        ]
    }
}
],
"PrimaryCriteria": {
    "CriteriaList": [
        {
            "ConditionOccurrence": {

```



```

        "CodesetId": 0
    }
}
],
"ObservationWindow": {
    "PriorDays": 0,
    "PostDays": 0
},
"PrimaryCriteriaLimit": {
    "Type": "First"
}
},
"QualifiedLimit": {
    "Type": "First"
},
"ExpressionLimit": {
    "Type": "First"
},
"InclusionRules": [
    {
        "name": "rule1",
        "expression": {
            "Criteria": {
                "ConditionOccurrence": {
                    "CodesetId": 1
                }
            },
            "StartWindow": {
                "Start": {
                    "Coeff": -1
                },
                "End": {
                    "Coeff": 1
                },
                "UseIndexEnd": false,
                "UseEventEnd": false
            },
            "Occurrence": {
                "Type": 0,
                "Count": 0
            }
        }
    }
]

```

```

    }
  }
],
"CensoringCriteria": [],
"CollapseSettings": {
  "CollapseType": "ERA",
  "EraPad": 0
},
"CensorWindow": {},
"cdmVersionRange": ">=5.0.0"
}

```

4.3 Adding a cohort to the CDM

```

cdm <- generateCohortSet(
  cdm,
  list(gibleed = gibleed_cohort_definition,
        gibleed_no_RA = gibleed_no_RA_cohort_definition
      ),
  name = "gibleed",
  computeAttrition = TRUE,
  overwrite = TRUE
)
cdm$gibleed %>%
  glimpse()

```

Rows: ??

Columns: 4

Database: DuckDB 0.8.1 [eburn@Windows 10 x64:R 4.2.1/C:\Users\eburn\AppData\Local\Temp\Rtmpi

\$ cohort_definition_id <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1~

\$ subject_id <dbl> 264, 613, 757, 893, 1117, 1313, 1491, 1576, 1935, ~

\$ cohort_start_date <date> 1984-06-22, 1977-02-09, 1950-01-22, 1993-11-09, ~

\$ cohort_end_date <date> 2011-11-18, 2019-06-14, 1998-10-25, 2019-05-06, ~

4.4 Adding a concept- based cohort to the CDM

5 Cohort attributes

```
cohortSet(cdm$gibleed)
```

```
# A tibble: 2 x 2
  cohort_definition_id cohort_name
      <int> <chr>
1             1 gibleed
2             2 gibleed_no_RA
```

```
cohortCount(cdm$gibleed)
```

```
# A tibble: 2 x 3
  cohort_definition_id number_records number_subjects
      <int>          <dbl>          <dbl>
1             1             479             479
2             2             479             479
```

```
cohortAttrition(cdm$gibleed)
```

```
# A tibble: 3 x 7
  cohort_definition_id number_records number_subjects reason_id reason
      <int>          <dbl>          <dbl>    <dbl> <chr>
1             1             479             479      1 Qualifying init~
2             2             479             479      1 Qualifying init~
3             2             479             479      2 rule1
# i 2 more variables: excluded_records <dbl>, excluded_subjects <dbl>
```

6 Custom cohort creation

7 Reconnecting to a database with existing cohorts

And if we already had some permanent cohort tables in the results schema, we could include references to these when creating our cdm reference like so:

```
cdm <- cdm_from_con(db,  
  cdm_schema = "main",  
  write_schema = "results",  
  cohort_tables = c("exposure_cohort", "outcome_cohort"))
```

💡 Controlling use of temp vs permanent tables

TO DO - explain use of options to set behaviour

8 Describing a cohort

8.1 PatientProfiles

8.2

9 Sampling the cdm

9.1 Subset of CDM tables

Our cdm reference will include all of the OMOP CDM tables that could be found in our database. We may though only want a subset of these tables, in which case we can explicitly specify those of interest:

```
# cdm_1 <- cdm_from_con(db) %>%  
#   cdm_select_tbl("person","observation_period")  
# cdm_1
```

We can also select a group of tables, for example selecting only the vocabulary tables like so:

```
# cdm_2 <- cdm %>%  
#   cdm_select_tbl(tbl_group("vocab"))  
# cdm
```

9.2 Sample records in a table

9.3 Random sample of the cdm

9.4 Cohort-based subset of the cdm

10 Bringing data into memory

Into R with collect - either a table or the entire cdm reference

Stow - local duck db and/ or arrow

References