

# *Introducción al uso de O.MOP en R*

Agustina Giuliadori Picco  
[agiuliadori@idiapjgol.org](mailto:agiuliadori@idiapjgol.org)

Marzo 2025

Real-World Epidemiology (RWEpi) Group

## *Indice*

1. Bases de datos, SQL y R
2. Bases de datos OMOP
3. Conexión a la base de datos con R
4. Manipulación de datos con R
5. Instanciar cohortes en OMOP CDM
6. Práctica: OMOP con R



# ***1. Bases de datos, SQL y R***

# ***1. Bases de datos, SQL y R***

BBDD: Almacenar, gestionar y recuperar datos.

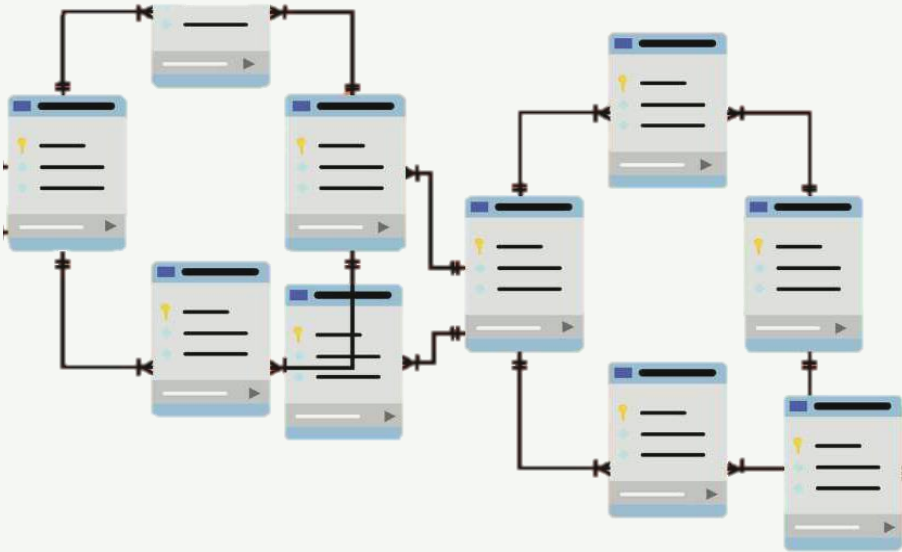
Definiciones básicas en una base de datos:

- 1. Modelo de datos:** Cómo se estructura la información → **BBDD relacional**
- 2. Sistema de gestión de bases de datos (dbms):** Software para administrar los datos.
- 3. Definición de esquemas:** Cómo se organiza la base de datos, definiendo tablas, relaciones y restricciones.

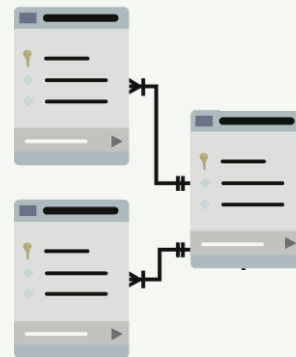
# ***1. Bases de datos, SQL y R***

## **Base de datos relacional**

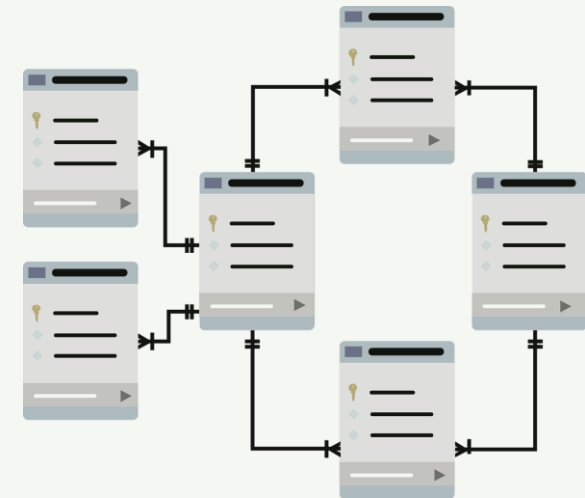
Esquema 1



Esquema 2



Esquema 3



# ***1. Bases de datos, SQL y R***



**SQL**: Lenguaje estándar para interactuar con la base de datos

Tipos de comandos en SQL:

1. DEFINICIÓN DE TABLAS (create, alter, drop tables)
2. MANIPULACIÓN DE DATOS (insert into, update, delete)
3. CONSULTA DE DATOS (select)
4. CONTROL DE DATOS

# ***1. Bases de datos, SQL y R***

## **SQL**



Lenguaje estándar por bases de datos.



SQL está optimizado para manejar y manipular grandes volúmenes de datos.



Problema: no tiene soporte para realizar operaciones estadísticas.

## **R**



Lenguaje de programación ampliamente utilizado en estadística.

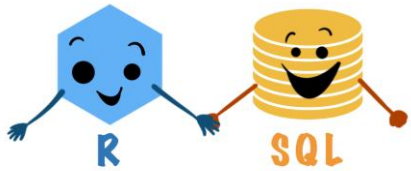


R es fuerte en la manipulación de datos, el análisis estadístico y la visualización.



Problema: podría no ser eficiente por grandes bases de datos.

# ***1. Bases de datos, SQL y R***



Combinar SQL y R para aprovechar las ventajas de cada una de las herramientas.

- **SQL**: creación y gestión de la base de datos, guardada en el **servidor**

- **R**:
  1. Connexión al servidor para acceder a la base de datos (BBDD).
  2. Manipulación y análisis de los datos (que estan en el servidor).
  3. Exportar del servidor a nuestro ordenador los resultados.

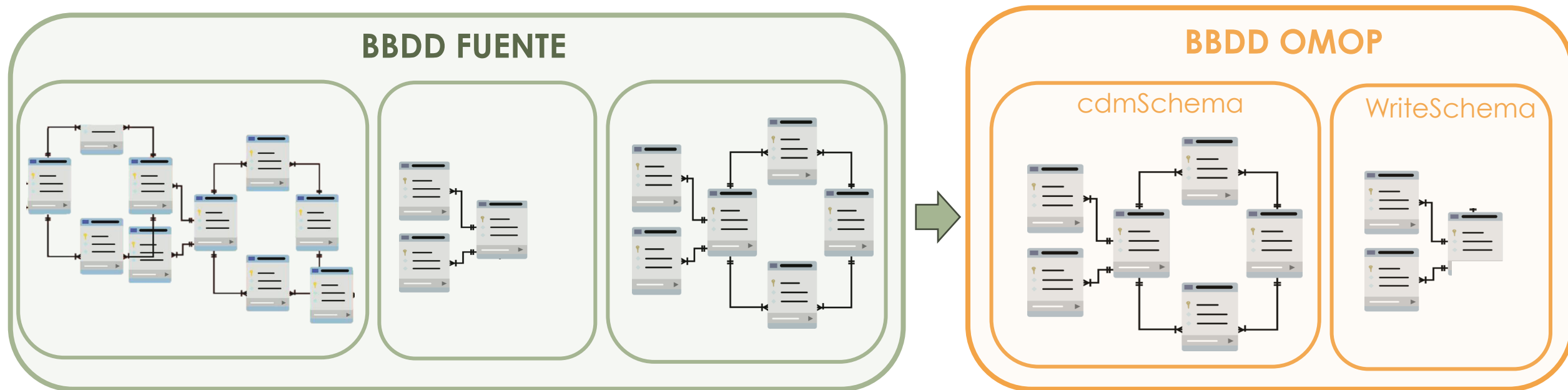


## ***2. Bases de datos OMOP***

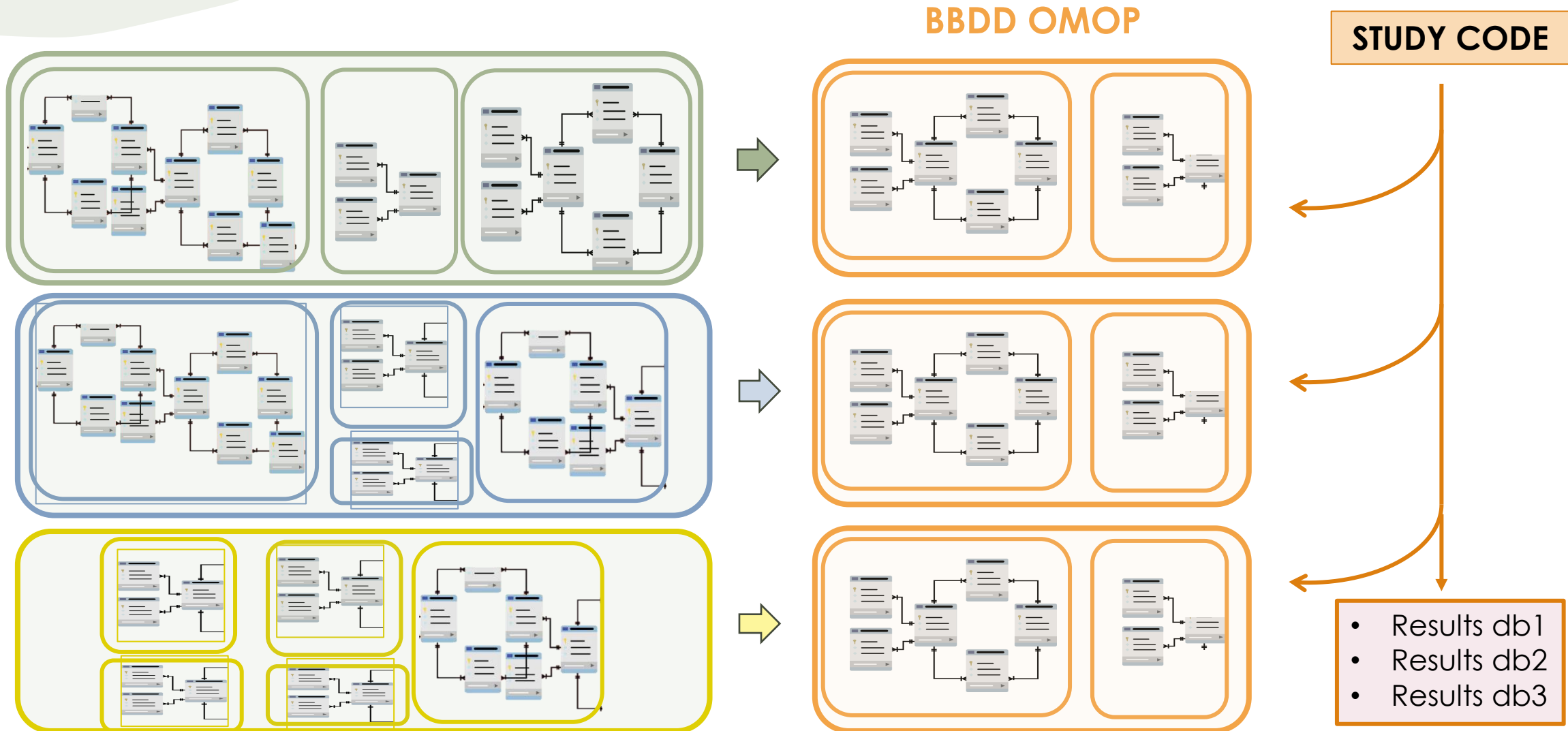
## ***2. Bases de datos OMOP***

### **OMOP CDM: (Observational Medical Outcomes Partnership Common Data Model)**

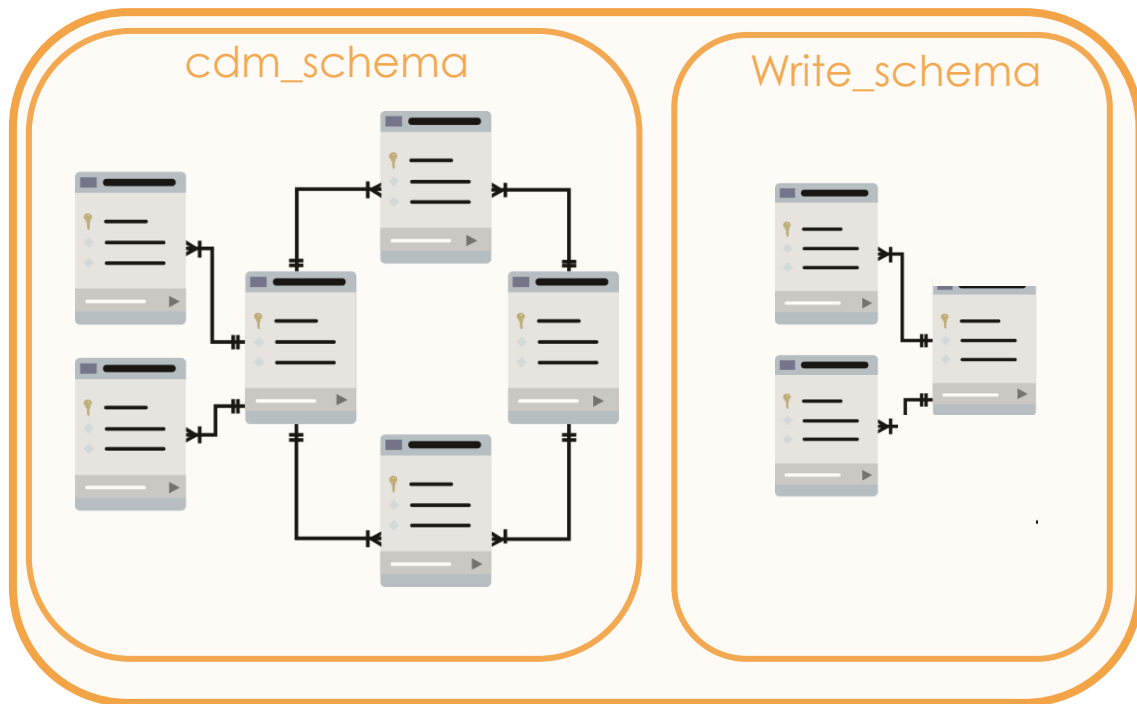
Modelo de datos común diseñado para **armonizar datos clínicos y administrativos** de diferentes fuentes de salud, permitiendo su análisis de manera estandarizada y reproducible.



## ***2. Bases de datos OMOP***



## ***2. Bases de datos OMOP***



En **OMOP CDM** diferenciamos dos **schemas**:

- **cdm schema**: Sólo lectura. Contiene todas las tablas con datos clínicos, administrativos, vocabularios, etc.
- **write schema**: Lectura/escritura. Podemos guardar tablas que hemos creado (por ej. cohortes).

## 2. Bases de datos OMOP

### TABLAS EN OMOP CDM

#### Datos clínicos:

- PERSON: **Pacientes** 🧑
- OBSERVATION\_PERIOD: **Período en observación** 🕒
- CONDITION\_OCCURRENCE **Condiciones médicas** 🩺
- VISIT\_OCCURRENCE: **Visitas médicas** 🏠
- DRUG\_EXPOSURE: **Medicación** 💊
- PROCEDURE\_OCCURRENCE **Procedimientos** 🏥
- MEASUREMENT: **Mediciones** 🧪
- OBSERVATION: **Observaciones** 🕒
- DEVICE\_EXPOSURE: **Dispositivos médicos** ⚙️

#### Terminologías:

- VOCABULARY
- CONCEPT
- CONCEPT\_RELATIONSHIP
- CONCEPT\_ANCESTOR
- SOURCE\_TO\_CONCEPT\_MAP
- ...

#### Sistema de salud

- LOCATION
- CARE\_SITE
- PROVIDER

- METADATA

#### Costos/Recursos

- COST
- PAYER\_PLAN\_PERIOD

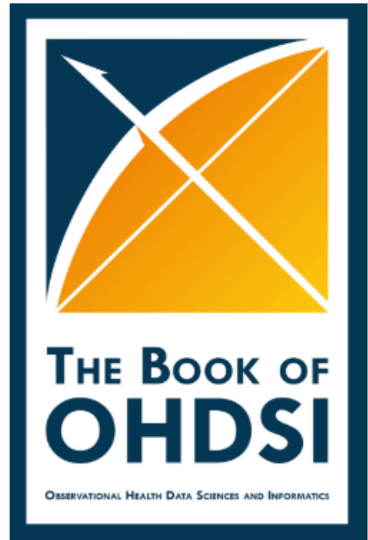
#### Tablas derivadas:

- DRUG\_ERA
- DOSE\_ERA
- CONDITION\_ERA

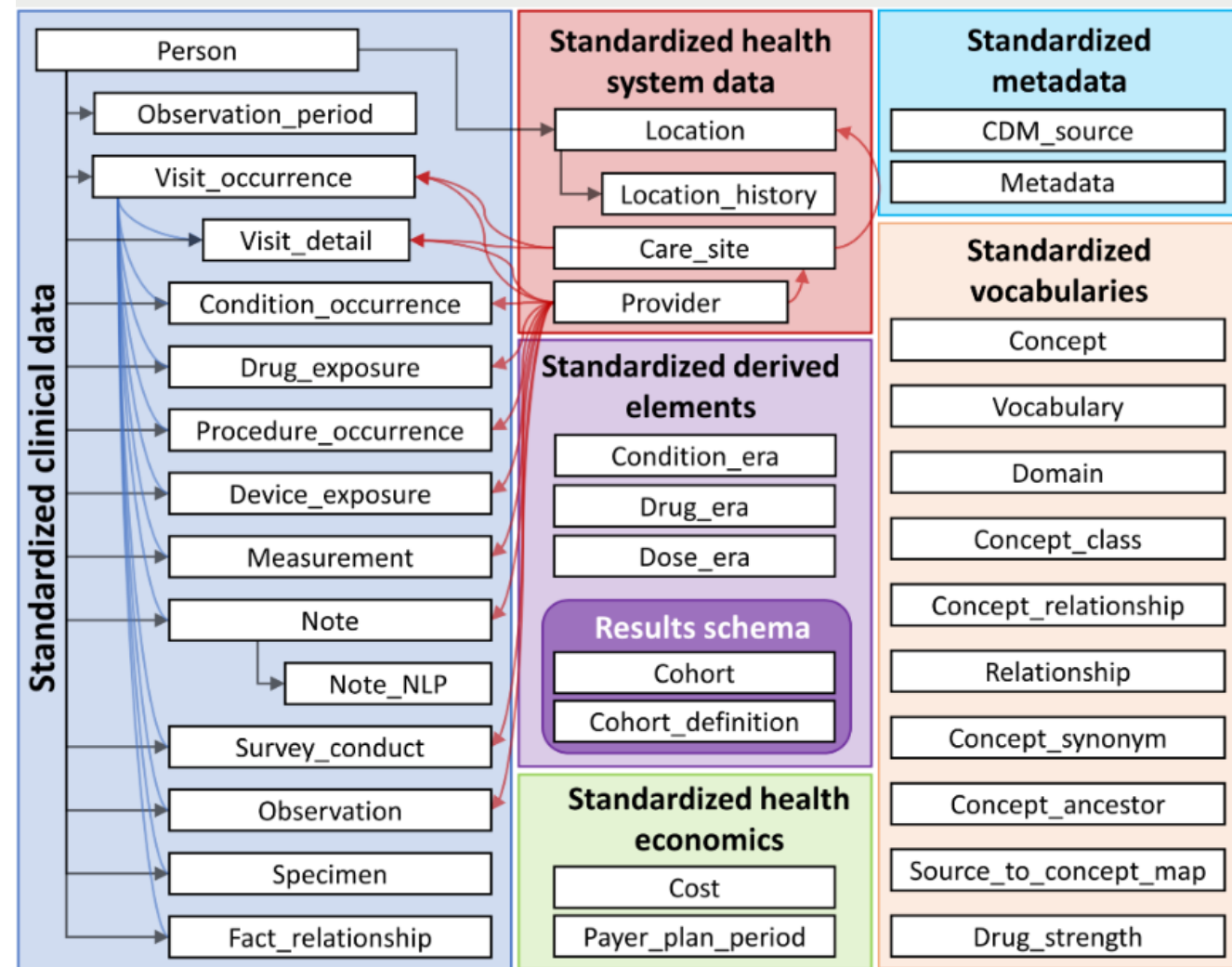
#### write schema:

- MY\_COHORTS

## 2. Bases de datos OMOP



<https://ohdsi.github.io/TheBookOfOhdsi/>



## 2. Bases de datos OMOP

- PERSON: Pacientes 🧑

```
> cdm$person
# Source:   table<main.person> [?? x 18]
# Database: DuckDB v0.10.2 [agiuliodori@windows 10 x64:R 4.4.0/C:\Users\agiuliodori\AppData\Local\Temp\RtmpGe
942.duckdb]
  person_id gender_concept_id year_of_birth month_of_birth day_of_birth birth_datetime race_concept_id
    <int>         <int>         <int>         <int>         <int>         <dtm>         <int>
1         1         8532         1970             4          24 1970-04-24 00:00:00         8527
2         2         8532         1929             3          18 1929-03-18 00:00:00         8527
3         3         8532         1970             4           4 1970-04-04 00:00:00         8527
4         4         8507         1966             2          26 1966-02-26 00:00:00         8527
5         5         8532         1936             6          10 1936-06-10 00:00:00         8527
6         6         8507         1996             5          29 1996-05-29 00:00:00         8516
7         7         8507         1923            11          14 1923-11-14 00:00:00         8527
8         8         8507         2018             8          20 2018-08-20 00:00:00         8527
9         9         8532         1933             2          11 1933-02-11 00:00:00         8527
10        10         8507         2010             3           7 2010-03-07 00:00:00         8527
# i more rows
# i 11 more variables: ethnicity_concept_id <int>, location_id <int>, provider_id <int>, care_site_id <int>,
#   person_source_value <chr>, gender_source_value <chr>, gender_source_concept_id <int>, race_source_value <chr>,
#   race_source_concept_id <int>, ethnicity_source_value <chr>, ethnicity_source_concept_id <int>
```

## 2. Bases de datos OMOP

- CONDITION\_OCCURRENCE: Condiciones médicas 🤒

```
> cdm$condition_occurrence
# Source:   table<main.condition_occurrence> [?? x 16]
# Database: DuckDB v0.10.2 [agiuliodori@Windows 10 x64:R 4.4.0/C:\Users\agiuliodori\AppData\Local\Temp\RtmpGeovpG\file34986b521942_duckdb1]
  condition_occurrence_id person_id condition_concept_id condition_start_date condition_start_datetime condition_end_date
      <int>          <int>          <int> <date>          <dtm>          <date>
1             1             2      381316 1986-09-08      1986-09-08 00:00:00 1986-09-08
2             2             6      321042 2021-06-23      2021-06-23 00:00:00 2021-06-23
3             3             7      381316 2021-04-07      2021-04-07 00:00:00 2021-04-07
4             4             8      37311061 2021-01-08      2021-01-08 00:00:00 2021-02-13
5             5             8      437663 2021-01-08      2021-01-08 00:00:00 2021-02-13
6             6             8      4089228 2021-01-08      2021-01-08 00:00:00 2021-02-13
7             7             8      254761 2021-01-08      2021-01-08 00:00:00 2021-02-13
8             8            16      381316 2020-02-11      2020-02-11 00:00:00 2020-02-11
9             9            16      313217 2021-10-05      2021-10-05 00:00:00 2021-10-05
10            10            18      317576 1993-08-08      1993-08-08 00:00:00 1993-08-08
# i more rows
# i 10 more variables: condition_end_datetime <dtm>, condition_type_concept_id <int>, condition_status_concept_id <int>,
#   stop_reason <chr>, provider_id <int>, visit_occurrence_id <int>, visit_detail_id <int>, condition_source_value <chr>,
#   condition_source_concept_id <int>, condition_status_source_value <chr>
```



### ***3. Conexión a la base de datos***

## ***3. Conexión a la base de datos***

Conexión desde R a BBDD SIDIAP:

1. Cargar librerías
2. Conexión a la base de datos
3. Creación del objeto **cdm**

RStudio

File Edit Code View Plots Session Build Debug Profile Tools Help

Go to file/function Addins

Practica1.R

```
1 library(DBI)
2 library(RPostgres)
3 library(CDMConnector)
4
5 db <- DBI::dbConnect(
6   Rpostgres::Postgres(),
7   dbname = server_dbi,
8   port = port,
9   host = host,
10  user = user,
11  password = password
12 )
13
14 cdm <- CDMConnector::cdmFromCon(
15   con = db,
16   cdm_schema = "omop",
17   write_schema = "results"
18 )
19
20
```

Run Source

Environment History Connections Tutorial

R Global Environment

Environment is empty

Files Plots Packages Help Viewer Presentat

so\_Introduccion\_OMOP\_2025-main > Dia1 > 3\_OMOP\_con\_R

|                          | Name                         | Size     | M  |
|--------------------------|------------------------------|----------|----|
|                          | ..                           |          |    |
| <input type="checkbox"/> | .RData                       | 3.1 MB   | M  |
| <input type="checkbox"/> | .Rhistory                    | 8.2 KB   | M  |
| <input type="checkbox"/> | .Rprofile                    | 27 B     | M  |
| <input type="checkbox"/> | 2_CDMConnector.Rproj         | 205 B    | Fe |
| <input type="checkbox"/> | CDMConnector_practica.html   | 905.9 KB | Fe |
| <input type="checkbox"/> | CDMConnector_soluciones.html | 900.9 KB | Fe |
| <input type="checkbox"/> | Cohortes                     |          |    |

Console Terminal Background Jobs

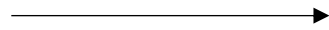
R 4.4.0 ~ /SEXUAL\_HEALTH/Preliminary\_code/

```
> class(cdm)
[1] "cdm_reference"
>
```

### ***3. Conexión a la base de datos***

Conexión desde R a BBDD SIDIAP:

1. Cargar librerías



```
library(Rpostgres)  
library(DBI)  
library(CDMConnector)
```

2. Conexión a la base de datos

3. Creación del objeto **cdm**

### ***3. Conexión a la base de datos***

Conexión desde R a BBDD SIDIAP:

1. Cargar librerías
2. Conexión a la base de datos
3. Creación del objeto **cdm**

Objeto de conexión

```
db <- DBI::dbConnect(  
  Rpostgres::Postgres(),  
  dbname = server_dbi,  
  port = port,  
  host = host,  
  user = user,  
  password = password  
)
```

Información del servidor

Credenciales

### ***3. Conexión a la base de datos***

Conexión desde R a BBDD SIDIAP:

1. Cargar librerías
2. Conexión a la base de datos
3. Creación del objeto **cdm** →

Objeto cdm

```
cdm <- CDMConnector::cdmFromCon(  
  con = db,   
  cdmSchema = "omop",  
  writeSchema = "results"  
)
```



### ***3. Conexión a la base de datos***

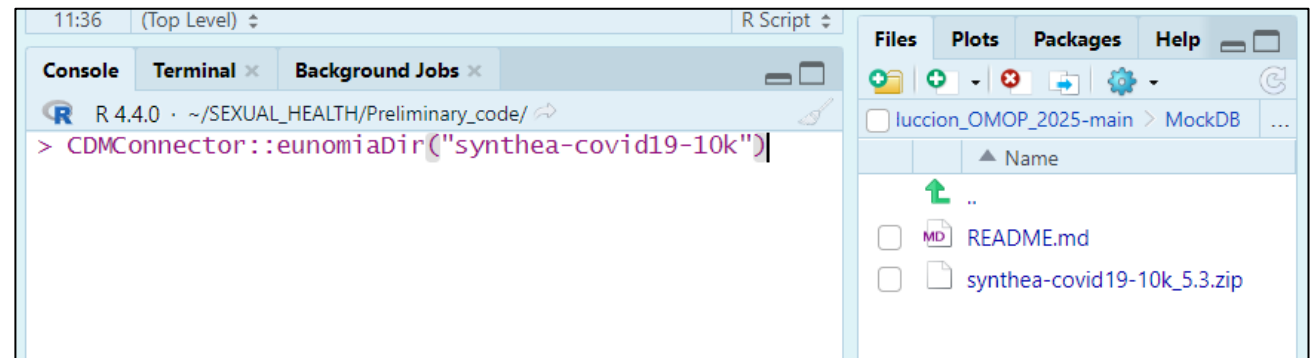
Conexión BBDD **sintética**:

1. Cargar librerías

2. Conexión a la base de datos

3. Creación del objeto **cdm**

```
# Synthetic database connection
db <- DBI::dbConnect(
  duckdb::duckdb(),
  CDMConnector::eunomiaDir("synthea-covid19-10k")
)
```




## ***3. Conexión a la base de datos***

Conexión BBDD **sintética**:

1. Cargar librerías

2. Conexión a la base de datos

3. Creación del objeto **cdm**

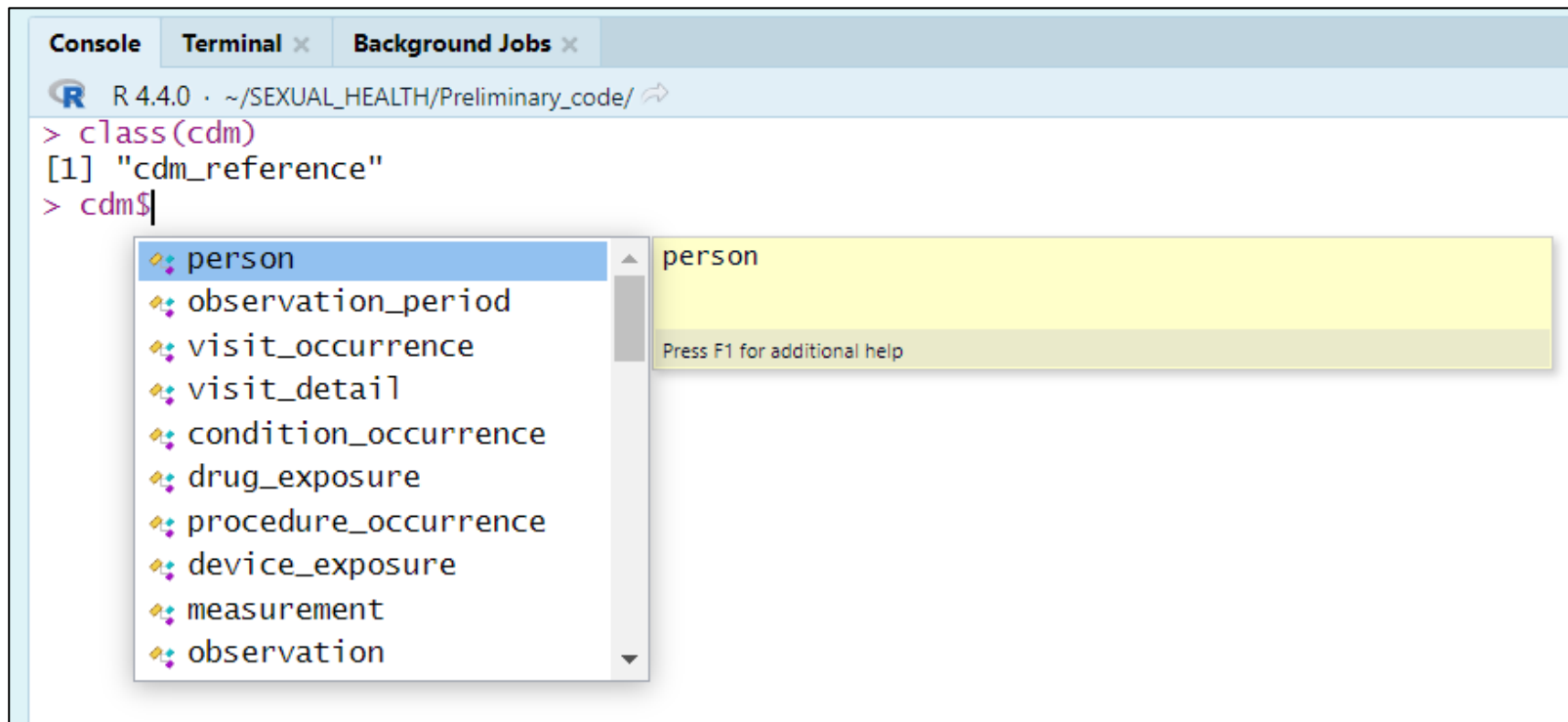


```
cdm <- CDMConnector::cdmFromCon(  
  con = db,  
  cdm_schema = "main",  
  write_schema = "main",  
  cdm_name = "SYNTHEA"  
)
```



## ***3. Conexión a la base de datos***

El **objeto cdm** contiene las tablas en formato OMOP CDM:



The screenshot shows an R console window with the following content:

```
R 4.4.0 · ~/SEXUAL_HEALTH/Preliminary_code/
> class(cdm)
[1] "cdm_reference"
> cdm$
```

A dropdown menu is open, showing a list of tables in the OMOP CDM format:

- person
- observation\_period
- visit\_occurrence
- visit\_detail
- condition\_occurrence
- drug\_exposure
- procedure\_occurrence
- device\_exposure
- measurement
- observation

The 'person' table is highlighted in yellow. A tooltip for 'person' is visible, showing the text 'Press F1 for additional help'.

## ***3. Conexión a la base de datos***

El **objeto cdm** contiene las tablas en formato OMOP CDM:

```
> print(cdm)

— # OMOP CDM reference (duckdb) of Synthea
• omop tables: person, observation_period, visit_occurrence, visit_detail, condition_occurrence, drug_exposure,
procedure_occurrence, device_exposure, measurement, observation, death, note, note_nlp, specimen, fact_relationship,
location, care_site, provider, payer_plan_period, cost, drug_era, dose_era, condition_era, metadata, cdm_source, concept,
vocabulary, domain, concept_class, concept_relationship, relationship, concept_synonym, concept_ancestor,
source_to_concept_map, drug_strength, cohort_definition, attribute_definition
• cohort tables: -
• achilles tables: -
• other tables: -
```

## 3. Conexión a la base de datos

Para acceder a las tablas, se procede igual que para acceder a listas en R:

```

Console Terminal x Background Jobs x
R 4.4.0 · ~/SEXUAL_HEALTH/Preliminary_code/
> class(cdm)
[1] "cdm_reference"
> cdm$person
# Source:   table<main.person> [?? x 18]
# Database: DuckDB v0.10.2 [agiuliodori@Windows 10 x64:R 4.4.0/C:\Users\agiuliodori\AppData\Local\Temp\RtmpGeovpG\file34981cc9494d.duckdb]
  person_id gender_concept_id year_of_birth month_of_birth day_of_birth birth_datetime race_concept_id
    <int>         <int>         <int>         <int>         <int> <dtm>          <int>
1         1           8532          1970             4           24 1970-04-24 00:00:00      8527
2         2           8532          1929             3           18 1929-03-18 00:00:00      8527
3         3           8532          1970             4            4 1970-04-04 00:00:00      8527
4         4           8507          1966             2           26 1966-02-26 00:00:00      8527
5         5           8532          1936             6           10 1936-06-10 00:00:00      8527
6         6           8507          1996             5           29 1996-05-29 00:00:00      8516
7         7           8507          1923            11           14 1923-11-14 00:00:00      8527
8         8           8507          2018             8           20 2018-08-20 00:00:00      8527
9         9           8532          1933             2           11 1933-02-11 00:00:00      8527
10        10           8507          2010             3            7 2010-03-07 00:00:00      8527
# i more rows
# i 11 more variables: ethnicity_concept_id <int>, location_id <int>, provider_id <int>, care_site_id <int>,
#   person_source_value <chr>, gender_source_value <chr>, gender_source_concept_id <int>, race_source_value <chr>,
#   race_source_concept_id <int>, ethnicity_source_value <chr>, ethnicity_source_concept_id <int>
# i Use `print(n = ...)` to see more rows

```

## ***4. Manipulación de datos con R***

## *4. Manipulación de datos con R*

- Para realizar la mayoría de las operaciones hay que usar funciones
- Todas las funciones de R están almacenadas en paquetes (R tiene unos paquetes básicos que se cargan automáticamente)
- Si queremos usar una función específica, podemos instalar el paquete que nos interesa y cargarlo:

```
install.package("nombre del paquete")  
library(nombre del paquete)
```

- Para acceder la ayuda de las funciones:

```
help(nombreFuncion)
```

## *4. Manipulación de datos con R*



dplyr

**dplyr** es una librería que nos permite transformar, filtrar, agrupar y resumir datos. Funciones principales:

- **glimpse()**: para “echar un vistazo” a un objeto.
- **filter()**: Filtra filas según condiciones.
- **select()**: Selecciona columnas específicas.
- **mutate()**: Crear nuevas columnas o modificar las existentes.
- **arrange()**: Ordena las filas por una o más columnas.
- **group\_by()**: Agrupar datos por una o más variables.
- **summarise()**: Resume datos con estadísticas (promedios, sumas, etc.)
- **tally()**: Cuenta el número de filas de una tabla

## 2. Bases de datos OMOP



dplyr

`dplyr::glimpse(cdm$person)`

```
> dplyr::glimpse(cdm$person)
Rows: ??
Columns: 18
Database: DuckDB v1.2.0 [agiuliodori@Windows 10 x64:R 4.4.0/C:\Users\agiuliodori\AppData\Local\Temp\Ry8kw\file2e02f7d6f76.duckdb]
$ person_id           <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, ...
$ gender_concept_id    <int> 8532, 8532, 8532, 8507, 8532, 8507, 8507, 8507, 8532, 8507, 853...
$ year_of_birth        <int> 1970, 1929, 1970, 1966, 1936, 1996, 1923, 2018, 1933, 2010, 198...
$ month_of_birth       <int> 4, 3, 4, 2, 6, 5, 11, 8, 2, 3, 3, 3, 3, 5, 12, 5, 11, 6, 5, 3, ...
$ day_of_birth         <int> 24, 18, 4, 26, 10, 29, 14, 20, 11, 7, 11, 25, 19, 4, 2, 2, 24, ...
$ birth_datetime       <dtm> 1970-04-24, 1929-03-18, 1970-04-04, 1966-02-26, 1936-06-10, 19...
$ race_concept_id      <int> 8527, 8527, 8527, 8527, 8527, 8516, 8527, 8527, 8527, 8527, 852...
$ ethnicity_concept_id <int> 38003564, 38003564, 38003564, 38003564, 38003564, 38003564, 380...
$ location_id          <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
$ provider_id          <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
$ care_site_id         <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ...
$ person_source_value  <chr> "00019d62-30d1-e285-e01c-68b371598db0", "00047b5c-a44f-00f1-ee1...
$ gender_source_value  <chr> "F", "F", "F", "M", "F", "M", "M", "M", "F", "M", "F", "M", "F"...
$ gender_source_concept_id <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
$ race_source_value    <chr> "white", "white", "white", "white", "white", "white", "black", "white", ...
$ race_source_concept_id <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
$ ethnicity_source_value <chr> "nonhispanic", "nonhispanic", "nonhispanic", "nonhispanic", "no...
$ ethnicity_source_concept_id <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
```

## 4. Manipulación de datos con R



dplyr

```
dplyr::tally(cdm$person)
```

```
> dplyr::tally(cdm$person)
# Source:   SQL [1 x 1]
# Database: DuckDB v0.10.2 [agiuliodori@windows
81cc9494d.duckdb]
      n
  <dbl>
1 10754
```

```
dplyr::summarise(cdm$person, fun(var))
```

```
> dplyr::summarise(cdm$person, median(year_of_birth))
# Source:   SQL [1 x 1]
# Database: DuckDB v0.10.2 [agiuliodori@windows 10 x64:R
81cc9494d.duckdb]
  `median(year_of_birth)`
      <dbl>
1          1970
```



## 4. Manipulación de datos con R



dplyr

```
dplyr::filter(cdm$person, year_of_birth > 2021)
```

| Person_id | Sex_id | Year_of_birth | Month_of_birth | Race_concept_id |
|-----------|--------|---------------|----------------|-----------------|
| 67        | 8532   | 1952          | 4              | 8527            |
| 98        | 8507   | 2022          | 5              | 8527            |
| 134       | 8507   | 1968          | 2              | 8527            |
| 235       | 8507   | 2007          | 12             | 8527            |
| 365       | 8532   | 2023          | 11             | 8527            |
| 487       | 8532   | 1998          | 5              | 8527            |

## 4. Manipulación de datos con R



dplyr

```
dplyr::filter(cdm$person, year_of_birth > 2021)
```

| Person_id | Sex_id | Year_of_birth | Month_of_birth | Race_concept_id |
|-----------|--------|---------------|----------------|-----------------|
| 98        | 8507   | 2022          | 5              | 8527            |
| 365       | 8532   | 2023          | 11             | 8527            |

## *4. Manipulación de datos con R*



dplyr

```
dplyr::select(
  cdm$person,
  c(person_id, sex_id, year_of_birth)
)
```

| Person_id | Sex_id | Year_of_birth | Month_of_birth | Race_concept_id |
|-----------|--------|---------------|----------------|-----------------|
| 67        | 8532   | 1952          | 4              | 8527            |
| 98        | 8507   | 2022          | 5              | 8527            |
| 134       | 8507   | 1968          | 2              | 8527            |
| 235       | 8507   | 2007          | 12             | 8527            |
| 365       | 8532   | 2023          | 11             | 8527            |
| 487       | 8532   | 1998          | 5              | 8527            |

## *4. Manipulación de datos con R*



dplyr

```
dplyr::select(  
  cdm$person,  
  c(person_id, sex_id, year_of_birth)  
)
```

| Person_id | Sex_id | Year_of_birth |
|-----------|--------|---------------|
| 67        | 8532   | 1952          |
| 98        | 8507   | 2022          |
| 134       | 8507   | 1968          |
| 235       | 8507   | 2007          |
| 365       | 8532   | 2023          |
| 487       | 8532   | 1998          |

## 4. Manipulación de datos con R



dplyr

Proporciona el “operador pipe” `%>%` que permite encadenar funciones:

```
dplyr::tally(
  dplyr::group_by(
    cdm$condition_occurrence,
    person_id
  )
)
```



|               | person_id          | n                  |
|---------------|--------------------|--------------------|
|               | <i>&lt;int&gt;</i> | <i>&lt;dbl&gt;</i> |
| 1             | 6                  | 1                  |
| 2             | 8                  | 4                  |
| 3             | 25                 | 1                  |
| 4             | 40                 | 1                  |
| 5             | 42                 | 3                  |
| 6             | 47                 | 5                  |
| 7             | 53                 | 2                  |
| 8             | 58                 | 4                  |
| 9             | 71                 | 2                  |
| 10            | 74                 | 2                  |
| # i more rows |                    |                    |

## 4. Manipulación de datos con R



dplyr

Proporciona el “operador pipe” `%>%` que permite encadenar funciones:

```
cdm$condition_occurrence %>%  
  dplyr::group_by(person_id) %>%  
  dplyr::tally( )
```



|               | person_id<br><int> | n<br><dbl> |
|---------------|--------------------|------------|
| 1             | 6                  | 1          |
| 2             | 8                  | 4          |
| 3             | 25                 | 1          |
| 4             | 40                 | 1          |
| 5             | 42                 | 3          |
| 6             | 47                 | 5          |
| 7             | 53                 | 2          |
| 8             | 58                 | 4          |
| 9             | 71                 | 2          |
| 10            | 74                 | 2          |
| # i more rows |                    |            |

## 4. Manipulación de datos con R



dplyr

**dplyr::compute()** ejecuta la consulta, y el resultado de la consulta se guarda en una tabla temporal en la base de datos.

**dplyr::collect()** ejecuta la consulta y devuelve el resultado a R. Podemos asignar `<-` el resultado de mi consulta a una variable local.

**dplyr::pull()** ejecuta la consulta y devuelve una columna específica de la tabla. Podemos asignar `<-` el resultado de mi consulta a una variable local.

## 4. Manipulación de datos con R

### R (dplyr)

```
pacientes_select <- pacientes %>%  
  select(id, nombre, edad)
```

```
pacientes_filtrados <- pacientes %>%  
  filter(edad > 40)
```

```
pacientes_mod <- pacientes %>%  
  mutate(edad_en_meses = edad * 12)
```

dplyr::show\_query()

### SQL

```
SELECT id, nombre, edad  
FROM pacientes;
```

```
SELECT * FROM pacientes  
WHERE edad > 40;
```

```
SELECT *, edad * 12 AS edad_en_meses  
FROM pacientes;
```



## 4. Manipulación de datos con R



dplyr

Ejemplo 1: Asignar `<-` una consulta a una variable local sin hacer `collect()`.

```
my_var <- cdm$condition_occurrence %>%  
  dplyr::filter(condition_concept_id == 37311061)
```

Practica1.R x my\_var x

Show Attributes

| Name       | Type                                 | Value             |
|------------|--------------------------------------|-------------------|
| my_var     | list [NA x 16] (S3: cdm_table, tbl_  | List of length 2  |
| src        | list [2] (S3: src_duckdb_connectic   | List of length 2  |
| lazy_query | list [12] (S3: lazy_select_query, la | List of length 12 |

(No selection)

## 4. Manipulación de datos con R



dplyr

Ejemplo 2: Asignar `<-` una consulta a una variable local haciendo **collect()**.

```
my_var_local <- cdm$condition_occurrence %>%
  dplyr::filter(condition_concept_id == 37311061) %>%
  dplyr::collect()
```

|   | condition_occurrence_id | person_id | condition_concept_id | condition_start_date | condition_start_datetime | condition_end_date |
|---|-------------------------|-----------|----------------------|----------------------|--------------------------|--------------------|
| 1 | 4                       | 8         | 37311061             | 2021-01-08           | 2021-01-08               | 2021-02-13         |
| 2 | 13                      | 36        | 37311061             | 2020-10-09           | 2020-10-09               | 2020-11-07         |
| 3 | 23                      | 47        | 37311061             | 2020-11-26           | 2020-11-26               | 2020-12-25         |
| 4 | 29                      | 52        | 37311061             | 2020-12-08           | 2020-12-08               | 2021-01-08         |
| 5 | 35                      | 58        | 37311061             | 2020-07-12           | 2020-07-12               | 2020-08-02         |
| 6 | 46                      | 72        | 37311061             | 2020-12-02           | 2020-12-02               | 2021-01-06         |



dplyr

## *4. Manipulación de datos con R*

Brinda funciones para **UNIR tablas** basadas en una o más columnas comunes entre sí.

1) **dplyr::inner\_join()** devuelve solo las filas que tienen coincidencias en ambas tablas.

2) **dplyr::left\_join()** devuelve todas las filas de la primera tabla y solo las coincidencias de la segunda.

3) **dplyr::right\_join()** devuelve todas las filas de la segunda tabla y solo las coincidencias de la primera.

4) **dplyr::full\_join()** devuelve todas las filas de ambas tablas.

## 4. Manipulación de datos con R



dplyr

1) **dplyr::inner\_join()** devuelve solo las filas que tienen coincidencias en ambas tablas.

CONDITION\_OCCURRENCE:

| Condition_id | Person_id | Condition concept id | Condition start date |
|--------------|-----------|----------------------|----------------------|
| 1326589      | 67        | 37311061             | 2020-11-27           |
| 56325853     | 365       | 37311061             | 2021-12-01           |
| 56325665     | 522       | 37311061             | 2022-06-29           |

Diagnóstico de covid

PERSON:

| Person_id | Year_of_birth | Sex_id |
|-----------|---------------|--------|
| 67        | 1952          | 8532   |
| 98        | 1973          | 8507   |
| 134       | 1968          | 8507   |
| 235       | 2007          | 8507   |
| 365       | 1994          | 8532   |
| 487       | 1998          | 8532   |

## 4. Manipulación de datos con R



dplyr

1) **dplyr::inner\_join()** devuelve solo las filas que tienen coincidencias en ambas tablas.

```
inner_join(CONDITION_OCCURRENCE, PERSON, by="person_id")
```

| Condition_id | Person_id | Condition concept id | Condition start date | Year_of_birth | Sex_id |
|--------------|-----------|----------------------|----------------------|---------------|--------|
| 1326589      | 67        | 37311061             | 2020-11-27           | 1952          | 8532   |
| 56325853     | 365       | 37311061             | 2021-12-01           | 1994          | 8532   |

## 4. Manipulación de datos con R



dplyr

2) **dplyr::left\_join()** devuelve todas las filas de la primera tabla y solo las coincidencias de la segunda.

CONDITION\_OCCURRENCE:

| Condition_id | Person_id | Condition concept id | Condition start date |
|--------------|-----------|----------------------|----------------------|
| 1326589      | 67        | 37311061             | 2020-11-27           |
| 56325853     | 365       | 37311061             | 2021-12-01           |
| 56325665     | 522       | 37311061             | 2022-06-29           |

PERSON:

| Person_id | Year_of_birth | Sex_id |
|-----------|---------------|--------|
| 67        | 1952          | 8532   |
| 98        | 1973          | 8507   |
| 134       | 1968          | 8507   |
| 235       | 2007          | 8507   |
| 365       | 1994          | 8532   |
| 487       | 1998          | 8532   |

## 4. Manipulación de datos con R



dplyr

2) `dplyr::left_join()` devuelve todas las filas de la primera tabla y solo las coincidencias de la segunda.

```
left_join(CONDITION_OCCURRENCE, PERSON, by="person_id")
```

| Condition_id | Person_id | Condition concept id | Condition start date | Year_of_birth | Sex_id |
|--------------|-----------|----------------------|----------------------|---------------|--------|
| 1326589      | 67        | 37311061             | 2020-11-27           | 1952          | 8532   |
| 56325853     | 365       | 37311061             | 2021-12-01           | 1994          | 8532   |
| 56325665     | 522       | 37311061             | 2022-06-29           | NA            | NA     |

## 4. Manipulación de datos con R



dplyr

3) **dplyr::right\_join()** devuelve todas las filas de la segunda tabla y solo las coincidencias de la primera.

CONDITION\_OCCURRENCE:

| Condition_id | Person_id | Condition concept id | Condition start date |
|--------------|-----------|----------------------|----------------------|
| 1326589      | 67        | 37311061             | 2020-11-27           |
| 56325853     | 365       | 37311061             | 2021-12-01           |
| 56325665     | 522       | 37311061             | 2022-06-29           |

PERSON:

| Person_id | Year_of_birth | Sex_id |
|-----------|---------------|--------|
| 67        | 1952          | 8532   |
| 98        | 1973          | 8507   |
| 134       | 1968          | 8507   |
| 235       | 2007          | 8507   |
| 365       | 1994          | 8532   |
| 487       | 1998          | 8532   |



## 4. Manipulación de datos con R



dplyr

3) **dplyr::right\_join()** devuelve todas las filas de la segunda tabla y solo las coincidencias de la primera.

```
right_join(CONDITION_OCCURRENCE, PERSON, by="person_id")
```

| Condition_id | Person_id | Condition concept id | Condition start date | Year_of_birth | Sex_id |
|--------------|-----------|----------------------|----------------------|---------------|--------|
| 1326589      | 67        | 37311061             | 2020-11-27           | 1952          | 8532   |
| NA           | 98        | NA                   | NA                   | 1973          | 8507   |
| NA           | 134       | NA                   | NA                   | 1968          | 8507   |
| NA           | 235       | NA                   | NA                   | 2007          | 8507   |
| 56325853     | 365       | 37311061             | 2021-12-01           | 1994          | 8532   |
| NA           | 487       | NA                   | NA                   | 1998          | 8532   |

## 4. Manipulación de datos con R



dplyr

4) **dplyr::full\_join()** devuelve todas las filas de ambas tablas.

CONDITION\_OCCURRENCE:

| Condition_id | Person_id | Condition concept id | Condition start date |
|--------------|-----------|----------------------|----------------------|
| 1326589      | 67        | 37311061             | 2020-11-27           |
| 56325853     | 365       | 37311061             | 2021-12-01           |
| 56325665     | 522       | 37311061             | 2022-06-29           |

PERSON:

| Person_id | Year_of_birth | Sex_id |
|-----------|---------------|--------|
| 67        | 1952          | 8532   |
| 98        | 1973          | 8507   |
| 134       | 1968          | 8507   |
| 235       | 2007          | 8507   |
| 365       | 1994          | 8532   |
| 487       | 1998          | 8532   |

## 4. Manipulación de datos con R



dplyr

4) **dplyr::full\_join()** devuelve todas las filas de ambas tablas.

**full\_join**(CONDITION\_OCCURRENCE, PERSON, by="person\_id")

| Condition_id | Person_id | Condition concept id | Condition start date | Year_of_birth | Sex_id |
|--------------|-----------|----------------------|----------------------|---------------|--------|
| 1326589      | 67        | 37311061             | 2020-11-27           | 1952          | 8532   |
| NA           | 98        | NA                   | NA                   | 1973          | 8507   |
| 56325665     | 522       | 37311061             | 2022-06-29           | NA            | NA     |
| NA           | 134       | NA                   | NA                   | 1968          | 8507   |
| NA           | 235       | NA                   | NA                   | 2007          | 8507   |
| 56325853     | 365       | 37311061             | 2021-12-01           | 1994          | 8532   |
| NA           | 487       | NA                   | NA                   | 1998          | 8532   |

## ***5. Instanciar cohortes en OMOP***

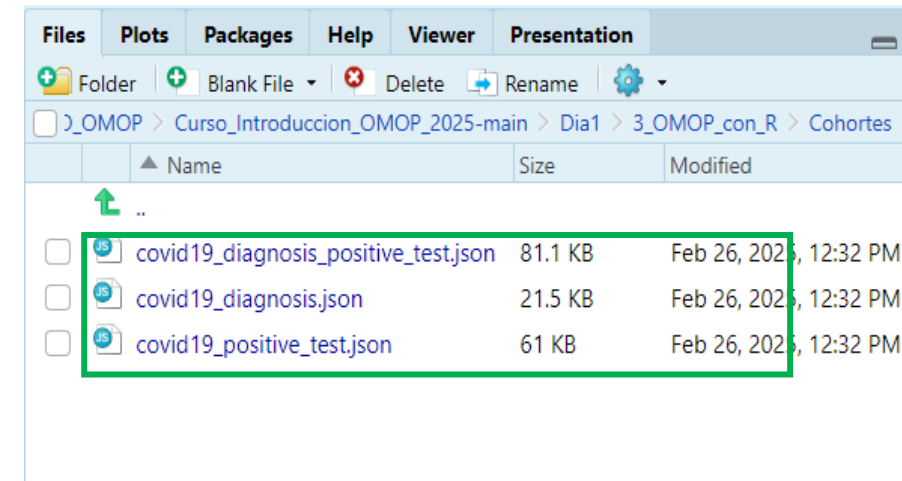
## 5. Instanciar cohortes



**CDMConnector** es una librería que nos permite crear el **cdm** y nos proporciona funciones para crear/manipular cohortes.

```
cohortSet <- readCohortSet("path/to/cohorts")
```

```
cdm <- cdm %>%  
  generateCohortSet(  
    cohortSet,  
    name = "cohort",  
    overwrite = TRUE  
  )
```





## 5. Instanciar cohortes

**CDMConnector** es una librería que nos permite crear el **cdm** y nos proporciona funciones para crear/manipular cohortes.

```

Console  Terminal x  Background Jobs x
R 4.4.0 · ~/CURSO_OMOP/Curso_Introduccion_OMOP_2025-main/Dia1/3_OMOP_con_R/
> cdm$cohort
# Source:   table<main.cohort> [?? x 4]
# Database: DuckDB v0.10.2 [agiuliodori@windows 10 x64:R 4.4.0/C:\Users\
p\RtmpGeovpG\file34981cc9494d.duckdb]
  cohort_definition_id subject_id cohort_start_date cohort_end_date
                <int>         <dbl> <date>          <date>
1                   1           620 2021-02-19      2021-03-01
2                   1          1455 2021-03-10      2021-03-20
3                   1          1850 2021-02-19      2021-03-01
4                   1          2254 2020-06-03      2020-06-13
5                   1          2307 2020-12-16      2020-12-26
6                   1          2648 2021-03-04      2021-03-14
7                   1          3762 2021-01-23      2021-02-02
8                   1          4198 2020-12-15      2020-12-25
9                   1          6418 2020-12-01      2020-12-11
10                  1          7518 2021-04-06      2021-04-16
# i more rows

```

## 5. Instanciar cohortes



**CDMConnector** es una librería que nos permite crear el **cdm** y nos proporciona funciones para crear/manipular cohortes.

`cohort_count(cdm$cohort)`

```
> cohort_count(cdm$cohort)
# A tibble: 3 × 3
  cohort_definition_id number_records number_subjects
      <int>           <int>           <int>
1             1             964             964
2             2             964             964
3             3              0              0
```

`settings(cdm$cohort)`

```
> settings(cdm$cohort)
# A tibble: 3 × 2
  cohort_definition_id cohort_name
      <int>   <chr>
1             1 covid19_diagnosis
2             2 covid19_diagnosis_positive_test
3             3 covid19_positive_test
```

`cohort_attrition(cdm$cohort)`

```
> cohort_attrition(cdm$cohort)
# A tibble: 6 × 7
  cohort_definition_id number_records number_subjects reason_id reason excluded_records excluded_subjects
      <int>           <int>           <int>   <int>   <chr>           <int>           <int>
1             1             964             964     1 Qualifying initial records             0             0
2             1             964             964     2 6 week wash-out             0             0
3             2             964             964     1 Qualifying initial records             0             0
4             2             964             964     2 6 week wash-out             0             0
5             3              0              0     1 Qualifying initial records             0             0
6             3              0              0     2 6 week wash-out             0             0
```



## ***6. PRÁCTICA DÍA 1***

### **OMOP con R**

*Introducción al uso de OMOP en R*



# *Ejercicios prácticos*

## **Objetivo principal**

Conectarse a la base de datos e instanciar una cohorte con los .JSON de COVID.

## **Objetivos secundarios**

Familiarizarse con el entorno Rstudio - SQL - OMOP.

- Entender cómo se trabaja con datos en SQL desde R.
- Familiarizarse con el formato de las tablas del schema cdm.
- Familiarizarse con la estructura de las cohortes.

## *Ejercicios prácticos*

### Buenas prácticas:

Recordar borrar tablas intermedias de la base de datos:

```
DBI::dbRemoveTable(db, "my_cohort")
```






Recordar desconectarnos:

```
CDMConnector::cdmDisconnect(cdm)
```

```
DBI::dbDisconnect(db)
```

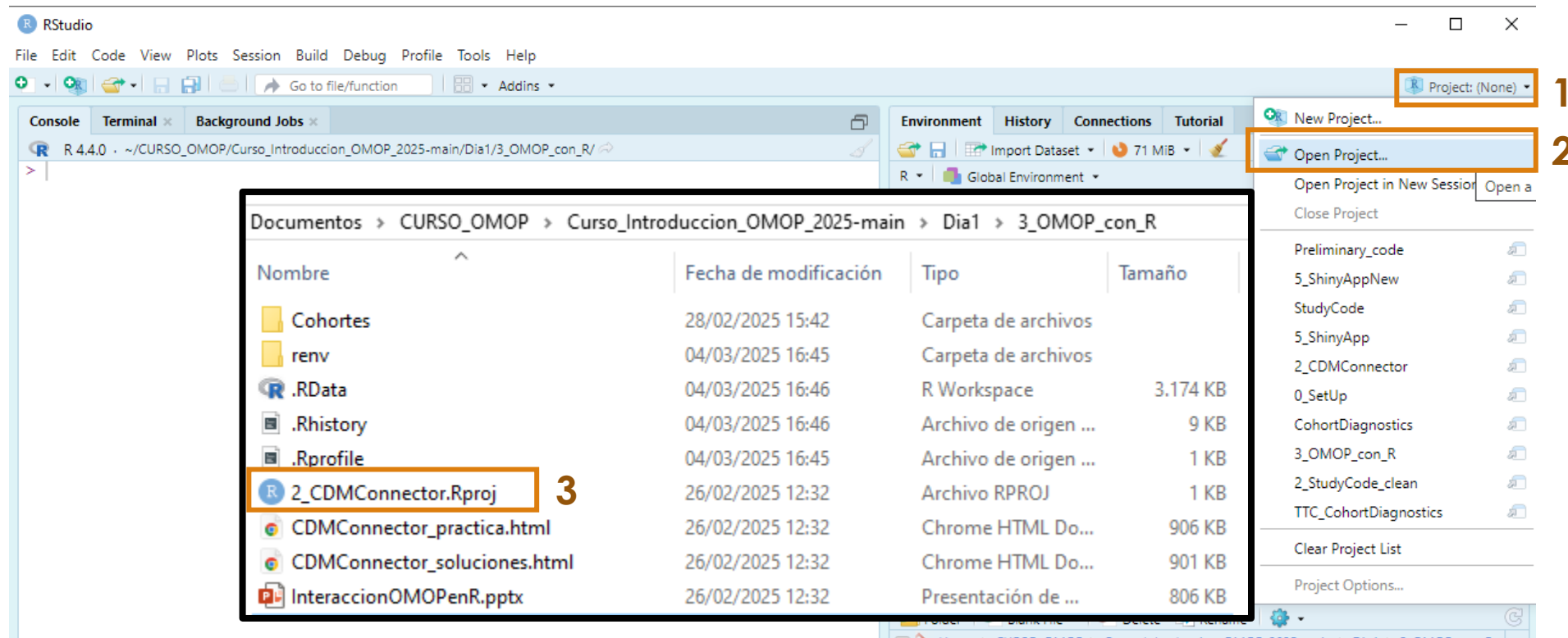
# *Ejercicios prácticos*

**Paso 1.** Acceder a la subcarpeta del Dia1 llamada “3\_OMOP\_con\_R”.

|   |                         |                                |
|---|-------------------------|--------------------------------|
|    | Cohortes                | Carpeta con las cohortes .json |
|    | 2_CDMConnector          | Proyecto R                     |
|    | CDMConnector_practica   | Enunciado práctica             |
|  | CDMConnector_soluciones | Soluciones práctica            |
|  | InteraccionOMOPenR      | Esta presentación              |

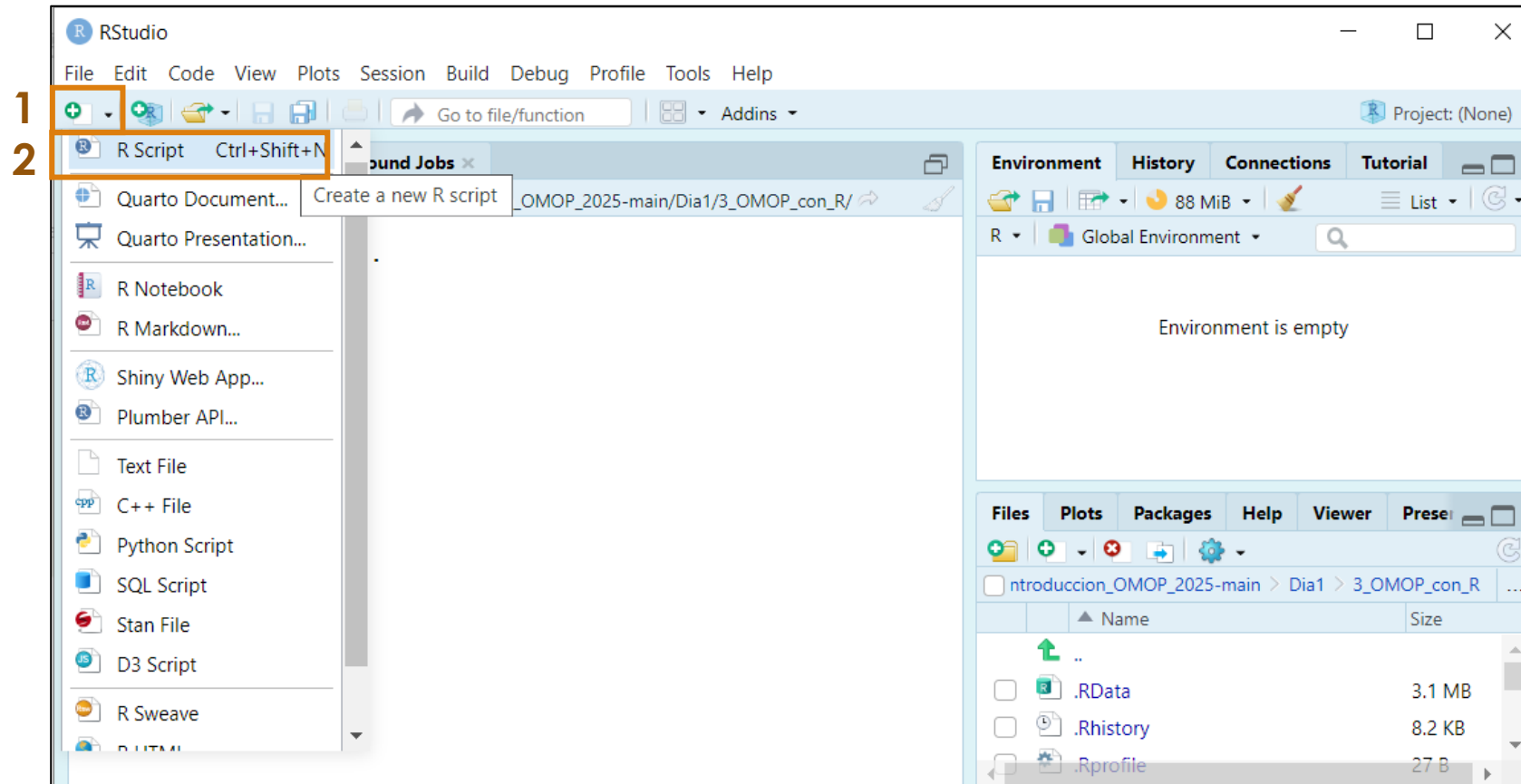
## Ejercicios prácticos

**Paso 2.** Abrir el proyecto R llamado “2\_CDMConnector.Rproj



# *Ejercicios prácticos*

**Paso 3.** Abrir un *script* en el cual desarrollaremos la práctica.



## *Ejercicios prácticos*

**Paso 4.** Abrir la práctica (CDMConnector práctica) y realizar los ejercicios.

### CDMConnector - práctica

Iniciación práctica en el análisis de datos OMOP

AUTHOR  
Real World Epidemiology

AFFILIATION  
IDIAP Jordi Gol

#### Objetivo

El objetivo de esta práctica es conectarse a una base de datos sintética en formato *OMOP*, e instanciar cohortes previamente generadas en ATLAS y guardadas en formato *.json*. Además, durante esta actividad, se destacan algunas particularidades de trabajar con bases de datos SQL mediante RStudio, y se presentarán buenas prácticas en este contexto.

#### ¿Cómo funciona?

En este fichero encontraréis una serie de ejercicios que os proponemos, acompañados de teoría y pistas para su resolución. Podéis crear un *script* R en el mismo directorio en el que está este fichero, e ir resolviéndolos en el *script*. En cada ejercicio hay pistas, puesto que el curso es abierto a muchos perfiles y habrá gente que necesitará indicaciones sobre programación en R, otras sobre dónde encontrar las cosas en OMOP, y otras con todo. Además, en este mismo directorio, hay un fichero adicional con las soluciones a los ejercicios, para poder autocorregiros la práctica. Si tenéis cualquier duda preguntad a los docentes de apoyo en las prácticas, e ¡intentad utilizar las soluciones tan solo para corregir!

#### Paquetes R

En primer lugar, añadiremos el paquete CDMConnector. Para instalarlo, deberás ejecutar:

Table of contents

[Objetivo](#)

[¿Cómo funciona?](#)

[Paquetes R](#)

[Conexión](#)

[Instanciar cohortes](#)

[SQL i R](#)

[Ejercicios extra](#)

*Gracias*

Agustina Giuliadori  
[agiuliodori@idiapjgol.org](mailto:agiuliodori@idiapjgol.org)

Real-World Epidemiology (RWEpi) Group