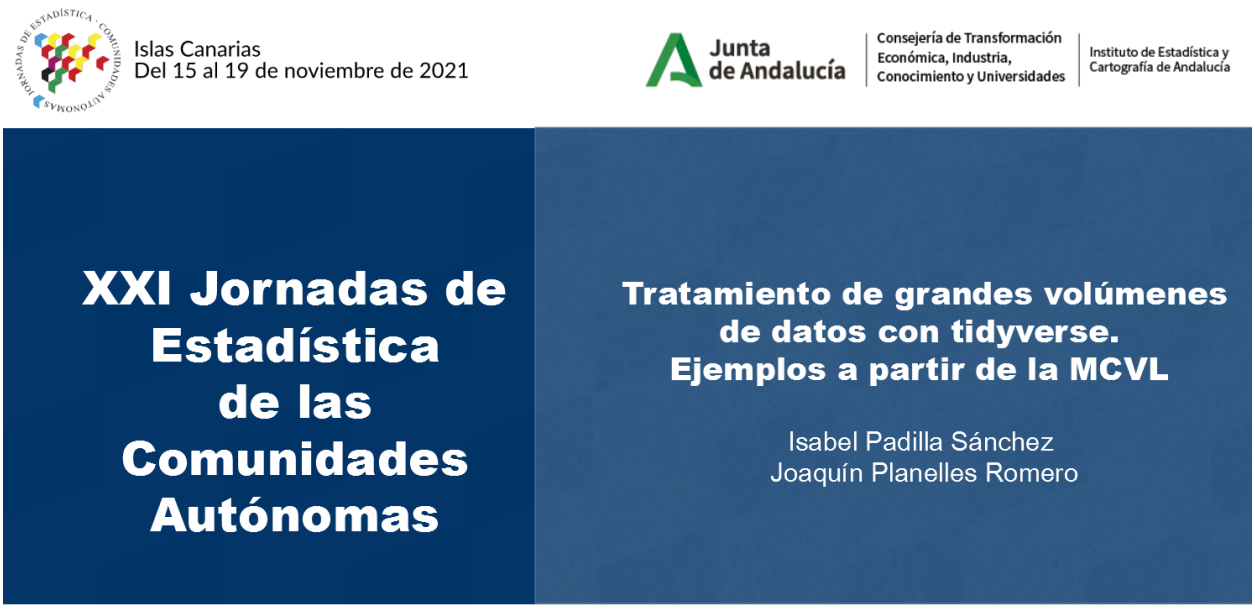
Tratamiento de grandes volúmenes de datos con tidyverse. Ejemplos a partir de la MCVL

Joaquín Planelles Romero e Isabel Padilla Sánchez

17 de noviembre de 2021

XXI Jornadas de Estadística de las Comunidades Autónomas



tidyverse

Se denomina tidyverse a un conjunto de librerías de R concebidas para el trabajo en Ciencia de Datos. En estas librerías están programadas una serie de funciones que facilitan el trabajo en las distintas fases del ciclo del dato: recogida de la información, tratamiento y difusión de resultados. Con ellas se puede hacer web-scrapping, tratamiento de textos, gráficos interactivos, informes automatizados... Se trata, además, de unas librerías de uso muy extendido, lo que facilita encontrar documentación para resolver cualquier problema concreto.

Las librerías que componen tidyverse comparten un diseño y una gramática comunes, lo que facilita la inter-operabilidad entre las distintas librerías/fases del ciclo del dato. En este sentido, podemos decir que tidyverse es una subcultura dentro del trabajo con R. El autor del concepto y de muchas de estas librerías es el científico jefe de RStudio, Hadley Wickham.

Para profundizar en tidyverse hacemos referencia a su propia página web, en la que se encuentra cada librería e información sobre sus funciones y argumentos correspondientes: https://www.tidyverse.org. Y también al libro escrito por los creadores de tidyverse: R for data science de Hadley Wickham y Garrett Grolemund, para el que también existe desde hace poco una versión completa en español: R para ciencia de datos

Gramática tidyverse

Los paquetes de tidyverse son interoperables y tienen una misma 'filosofía de trabajo'. He aquí dos de esos principios compartidos por los paquetes de tidyverse: uso de tuberías y de datos ordenados

1. Uso de tuberías

Las tuberías son un operador que sirve para realizar varias operaciones de forma secuencial sin recurrir a paréntesis anidados o a realizar una operación en cada línea de código (con lo que eso supone en términos de extensión del script y de crear/sobrescribir multitud de objetos). Se representan mediante el símbolo `%>%`.

Para ver como funciona esto, supongamos que tenemos un vector x, compuesto por los números pares entre el 2 y el 10. Para ellos queremos realizar algunas transformaciones: primero, obtener el logaritmo de cada elemento. Una vez hecha esa transformación queremos obtener la raíz cuadrada. Finalmente, queremos calcular el promedio de los distintos elementos que constituyen el vector transformado y mostrarlo redondeado al primer decimal. Para realizar esto lo convencional consistiría en anidar las operaciones, del siguiente modo:

```r
x <- c(2, 4, 6, 8, 10)
y <- round(mean(sqrt(log(x))), 1)
```

En cambio, si se utilizan tuberías, es mucho más fácil de interpretar a primera vista ya que se lee de izquierda a derecha y no de adentro hacia afuera:

```r
y <- x %>% log() %>% sqrt() %>% mean() %>% round(1)
```

2. Uso de datos ordenados o 'datos tidy'

Es importante estructurar los datos bien para que la respuesta sea eficiente (rápida). Tanto es así que 'tidy' significa 'ordenado'. Es decir, `tidyverse` significa literalmente, 'el universo ordenado'. R es un lenguaje muy eficiente al trabajar con vectores. Para aprovechar esa potencialidad, `tidyverse` necesita trabajar con datos rectangulares, organizados por filas y columnas (filas contiguas y del mismo tamaño, columnas contiguas y del mismo tamaño).

Los objetos rectangulares de uso más común en R son data.frames y tibbles. Los tibbles son algo así como unos data.frames evolucionados y son la estructura de datos preferida por los paquetes de 'tidyverse'.

El núcleo de tidyverse

El núcleo de tidyverse lo componen 8 paquetes o librerías, que son los de uso más frecuente. Estas 8 librerías se pueden instalar y cargar por separado o todas juntas. Para instalarlas todas juntas habrá que utilizar el comando `install.packages('tidyverse')`. Una vez instaladas, para cargarlas en la sesión de trabajo, habrá que utilizar, como si de una única librería se tratara, el comando `library(tidyverse)`.

En la imagen siguiente se presentan los iconos de cada una de estas librerías y su cometido principal. Si queréis profundizar en las funcionalidades de cada una, os recomendamos que os dirijáis directamente a la propia página de tidyverse. En este curso veremos con cierto detalle 3 de estas librerías: `dplyr`, `tidyr` y `tibble`.

Otros paquetes de tidyverse

`tidyverse` incluye muchos otros paquetes, de uso más especializado, por lo que no forman parte del núcleo. Por ello, estos paquetes no se cargan con `library('tidyverse')`, sino que habrá que cargar cada uno de ellos por separado cuando queramos acceder a las funciones más específicas que tienen implementadas. He aquí **algunos** de estos paquetes:

Estas librerías permiten hacer distintas cosas:

- Unas facilitan tratar con un determinado tipo de datos. Por ejemplo, el paquete `lubridate`, está diseñado para tratar con fechas y duraciones.
- Otros paquetes funcionan como traductores. Desde RStudio están construyendo una especie de 'piedra rosetta' que permite transformar el código `dplyr` que hayas escrito en código optimizado para otros lenguajes de programación. Es un proceso prácticamente ciego para el usuario: tú escribes tu código `dplyr` en la consola de `RStudio` y por detrás se transforma en sentencias de otro lenguaje que se corren y te devuelven el objeto deseado.

  Podemos hacer referencia a dos de estos paquetes: `dbplyr` permite transformar tu código `dplyr` en un código SQL. Esto te permite, por ejemplo, conectarte desde R a una base de datos externa y pedirle que realice determinadas acciones mediante sentencias SQL. Como veremos, esta puede ser una estrategia muy potente de cara a evadir las limitaciones de R con la memoria volátil (RAM). La librería `dtplyr` permite transformar el código dplyr en código data.table.
- Finalmente, mencionar que hay un conjunto amplio de paquetes de `tidyverse` especialmente diseñados para construir modelos y para programar.

Problemas de memoria

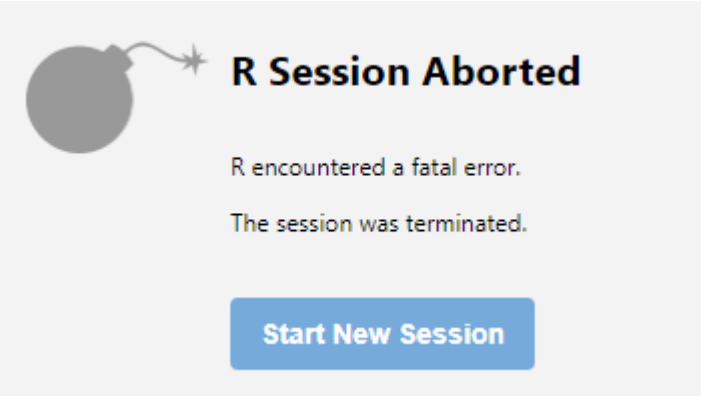
Cuando trabajamos con gran cantidad de datos desde R, resultas familiares mensajes como el siguiente en el que el programa te advierte de que el tamaño del set de datos le supera. El problema es que R tiene limitaciones al trabajar con grandes volúmenes de datos, ya que consume recursos de la memoria volátil

(RAM) del ordenador. Incluso, cuando insistes en pedirle más de lo que puede ofrecerte, te puede llegar a aparecer un mensaje muy gráfico que te invita a salir y a reiniciar la sesión de R.

Una situación no tan extrema como las anteriores, pero mucho más frecuente, se da cuando el volumen de datos no llega a colapsar el ordenador, pero sí afecta a su rendimiento. ¿Qué hacer? Propondremos distintas estrategias de acción, aunque pondremos el énfasis en la conexión a bases de datos y en el uso de la librería dbplyr. Se trata de una solución elegante y muy potente, que te permite trabajar desde R con cientos de millones de registros, si es necesario.

Mensajes de R

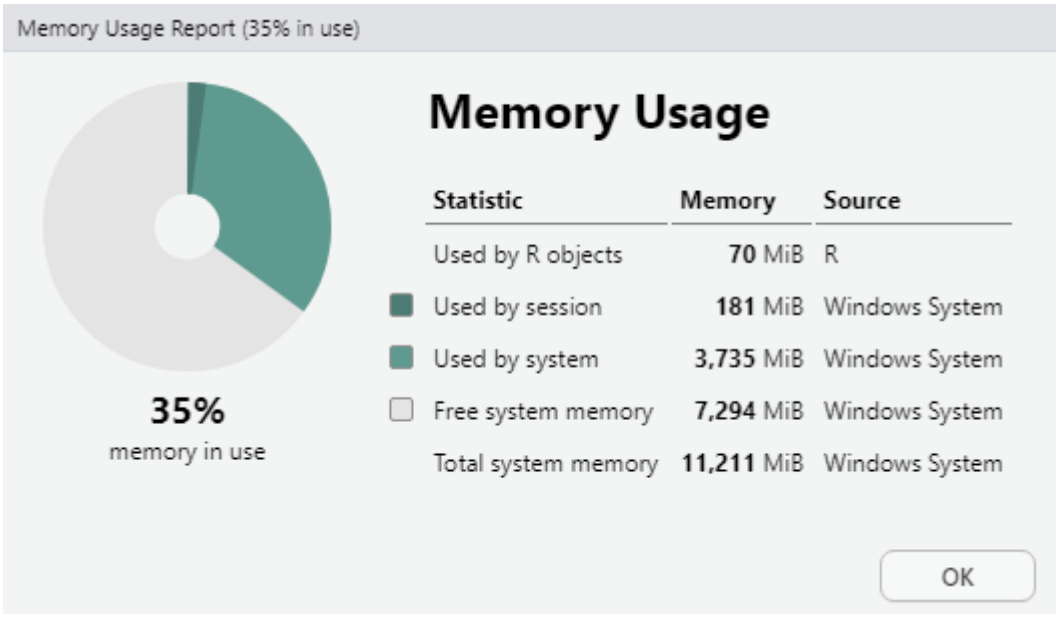
```
> arquivo_atual_validado<-inner_join(arquivo_atual,arquivo_antigo_fi
ltrado,by="codigo_estado")
Error: cannot allocate vector of size 1.4 Gb
>
```



Posibles soluciones

Opción 1. Extensión de la memoria RAM del equipo

Actualmente (año 2021) en los ordenadores personales disponibles en el mercado encontrarás opciones entre los 8 GB de un equipo básico hasta los 32 GB de un equipo premium/gamer. También hay ordenadores con mayor capacidad, pensados para trabajo profesional. No obstante, es muy importante tener en cuenta una cosa. Frecuentemente lo que limita no es tanto la memoria RAM del equipo, sino la cantidad de esa memoria RAM que el equipo reserva a la sesión de R. Cuando trabajas en tu equipo y abres distintos programas, java asigna una determinada cantidad de memoria a cada uno. De ese modo, asignará a la sesión de R una pequeña parte de la memoria total, frecuentemente 256/512 MB. Esto puedes cambiarlo en cada sesión de trabajo. El comando para hacerlo, es el siguiente: options(java.parameters = "-Xmx4g") donde el final '4g' está indicando que quieres reservar 4 gigas. Puedes indicar la cantidad que te sea más útil, teniendo en cuenta la capacidad de tu ordenador. Para que esto tenga el efecto deseado hay que indicarlo al inicio de la sesión de R, antes de cargar las librerías. Por ello, cuando nosotros creamos un script en el que hay un gran volumen de datos implicados, ubicamos el comando como el primer elemento del script,



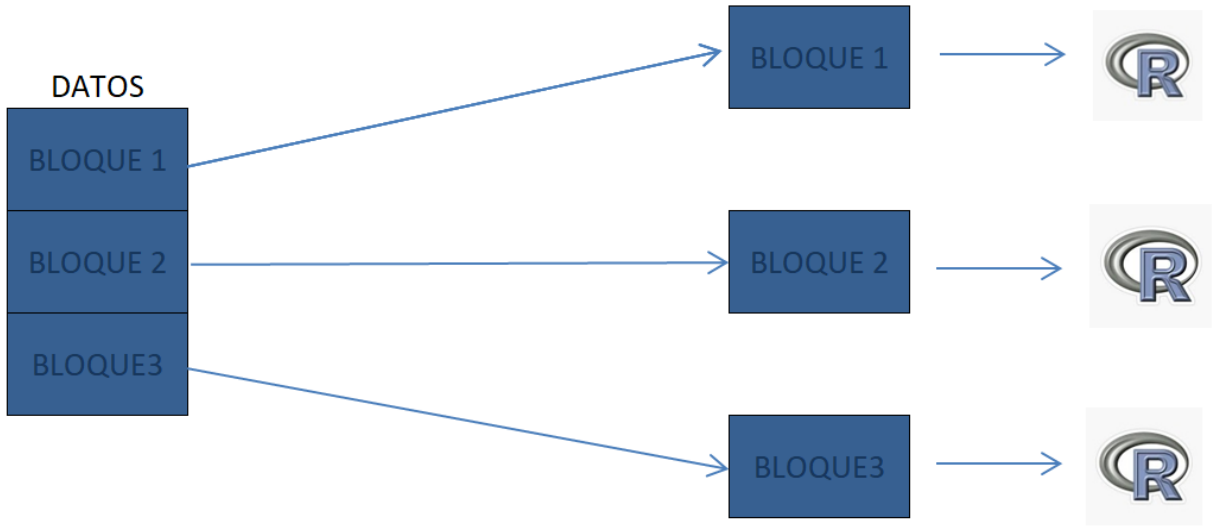

```
options(java.parameters = "-Xmx4g")

#Cargamos las librerías necesarias
library(tidyverse)
library(readxl)

...
```

Opción 2. Particionar los datos

Consiste en particionar esos datos que te resultan demasiado grandes, y enfrentarte a cada bloque (chunk) de información por separado. Si no puedes procesar de una vez los datos de toda España, pero sí puedes procesar los datos de cada provincia, quizás te resulta posible comenzar realizando una partición de tus ficheros y trabajar con cada sub-fichero por separado. De hecho, también hay paquetes de R pensados específicamente para implementar este tipo de soluciones, como los paquetes LaF y chunked. Esta idea no sirve únicamente para leer ficheros planos, también vale para otros grandes conjuntos de datos a los que accedemos a través de una API, o para comunicarse de forma rudimentaria con una base de datos.

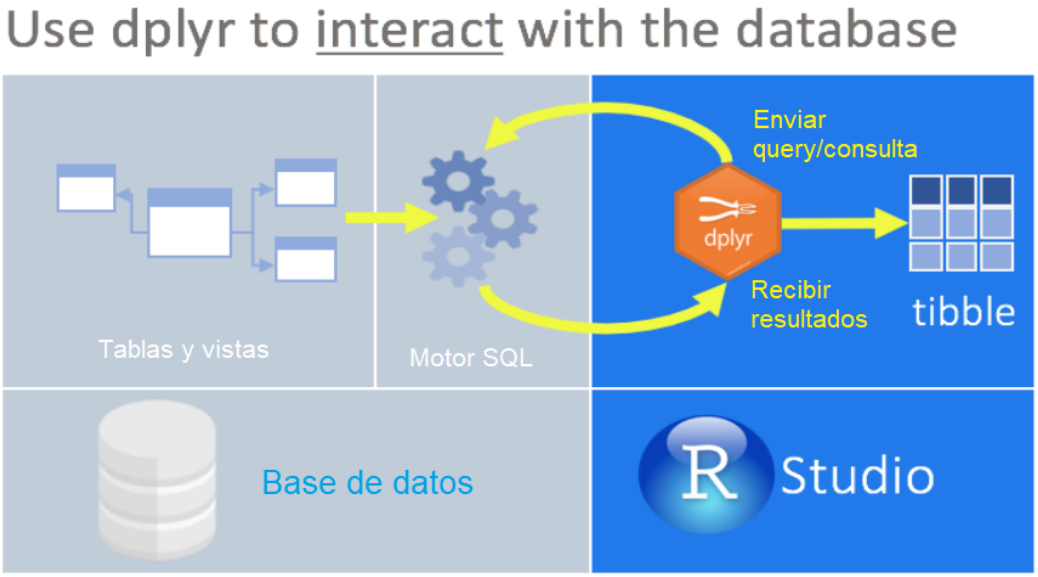


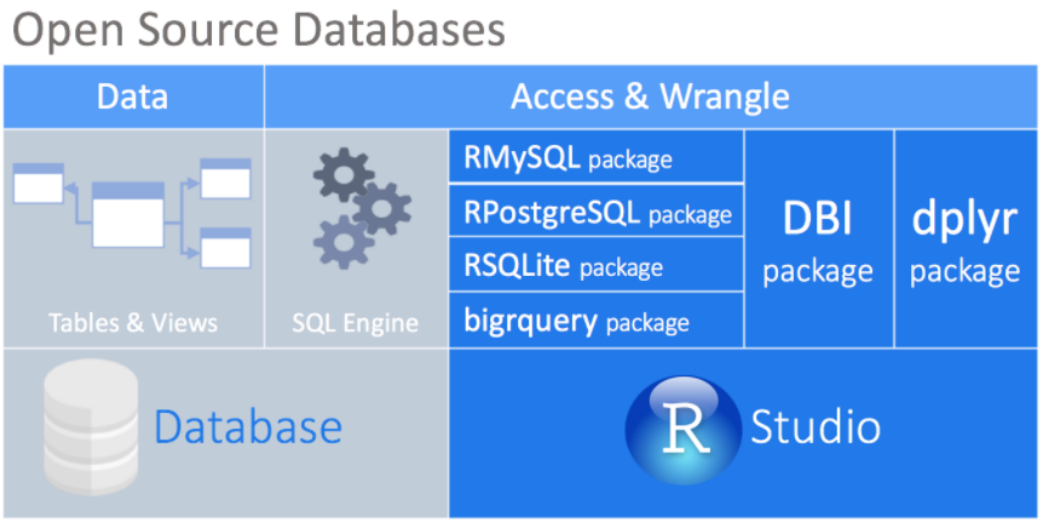
Opción 3. Conectarse a bases de datos relacionales

Dejamos que sea el motor de la propia base de datos el que soporte la mayor parte del trabajo. En nuestro trabajo como productores de información estadística es muy frecuente recibir la información en un texto plano que unas veces te interesará leer directamente desde R, pero en otras ocasiones te interesará empezar por almacenar en una base de datos relacional. Los motivos que típicamente te llevan a ello son la seguridad (control de accesos, diferenciación de roles) y el tamaño (porque tu conjunto de datos sea más grande que la memoria RAM de tu ordenador, o lo suficientemente grande como para comprometer el rendimiento).

Conexión a BD y dbplyr

Interacción entre R y la base de datos.





Formas de interacción con la base de datos

1. librería DBI. Puedes escribir trocitos de código en SQL insertados en medio de tu código R y lanzarlo a la base de datos.

```
AFI<- dbSendQuery(con,
      paste0("CREATE TABLE AFI AS
              SELECT pk_id,IDPF,GRUPCOT,
               CASE
                WHEN  GRUPCOT not in ('01','02','03','04','05','06','07','08','09','10','11','12') THEN '00'
                ELSE GRUPCOT
               END AS GRUPCOT_R
              FROM MCVL_AFILIACION
              WHERE LOTE=",ANOREF," AND substr(FALTA, 1, 4)<", ANOREF+1," AND substr(FBAJA, 1, 4)>",ANOREF-1))

AFI<- tbl(con, "AFI")
```

2. librería dbplyr. Puedes escribir tu código íntegramente en R y esta librería se encarga de hacer la traducción a SQL y comunicarse con la base de datos.

```
AFI<- tbl(con, "MCVL_AFILIACION") %>%
      filter(LOTE==ANOREF) %>%
      filter(substr(FALTA, 1, 4)< ANOREF+1 & substr(FBAJA, 1, 4)>ANOREF-1) %>%
      mutate(GRUPCOT_R=
             if_else(!GRUPCOT%in%c('01','02','03','04','05','06','07','08','09','10','11','12'),'00',GRUPCOT))%>%
      select(PK_ID,IDPF,GRUPCOT,GRUPCOT_R)
```

Normalmente la traducción a SQL es ciega para el usuario. No obstante, en ocasiones te puede interesar ver qué traducción está haciendo. El comando `show_query()` sirve precisamente para eso.

Muestra continua de vidas laborales

Los datos de la Muestra Continua de Vidas Laborales proceden de una selección aleatoria de los afiliados y/o pensionistas de la Seguridad Social en el año de referencia. Para ellos, la Seguridad Social incorpora información relativa a cada episodio de cotización y de cobro de prestaciones. A estos datos se les añaden otros procedentes del Padrón Continuo Municipal (INE) y del resumen anual de retenciones e ingresos a cuenta del IRPF de la Agencia Tributaria.

El proceso de selección da lugar a una muestra de 1,2 millones de personas para España y de unas 200 mil para Andalucía, con la particularidad de que para cada una de ellas se incorpora todo su historial de relaciones con la Seguridad Social, no sólo las del periodo de referencia, lo que permite analizar la situación del mercado de trabajo en un momento del tiempo, así como las trayectorias laborales previas.

La estructura de la MCVL

Anualmente la Seguridad Social elabora dos versiones de la MCVL: la primera, que difunde en torno al mes de julio del año siguiente al de referencia, se denomina versión "sin datos fiscales". La elabora la Seguridad Social, incluyendo para las personas seleccionadas información procedente de los registros de la Seguridad Social y del Padrón Continuo.

La segunda versión o "versión con datos fiscales" se difunde unos meses más tarde. La elabora la Agencia Tributaria a partir de las tablas sin datos fiscales. La AEAT cambia los identificadores anonimizados de los individuos seleccionados y añade información fiscal que procedente del Modelo 190 (resumen anual de retenciones e ingresos a cuenta). Esta última versión, que está disponible desde el año 2006, es la que se explota en el IECA.

La información se organiza en 6 tablas, que se pueden cruzar entre ellas utilizando el identificador anonimizado de la persona (que aparece en todas ellas) y, de ser necesario, otros campos comunes.

Tablas con un único registro por persona:

1 - Tabla Personas: contiene información relativa a las personas seleccionadas que procede de la Seguridad Social y del Padrón Municipal, como fecha y lugar de nacimiento, lugar de residencia, sexo, nivel de estudios...

2 - Tabla Convivientes: contiene para cada persona seleccionada el sexo y la fecha de nacimiento de las personas registradas con ella en la hoja padronal.

Tablas con uno o más registros por persona:

3 - Tabla Afiliación: muestra los datos esenciales de cada episodio de cotización a la Seguridad Social que el individuo ha mantenido a lo largo de su vida. Para cada episodio se dispone de la fecha de alta y la fecha de baja e información sobre características del mismo: tipo de contrato, grupo de cotización, coeficiente de parcialidad, etc.

4 - Tabla Pensiones: muestra las características fundamentales de las pensiones contributivas que perciben, o han percibido, en el pasado los individuos seleccionados.

5 - Tabla Bases de Cotización: refleja el importe mensual de las bases de cotización de las personas en la MCVL.

6 - Tabla Datos Fiscales: Los datos proceden del Modelo 190 y contiene información individualizada sobre las retribuciones satisfechas y las retenciones practicadas por IRPF a las personas incluidas en la MCVL durante el año de referencia, así como algunos datos relativos a su situación familiar cuando son necesarios para la aplicación de reducciones o beneficios fiscales.

El proceso de selección de la muestra y la información que se incorpora en las tablas anteriores para cada individuo seleccionado, presenta ciertas particularidades que proporcionan a la MCVL una gran ductilidad en cuanto a los distintos tipos de análisis que se pueden realizar con ella, en concreto en lo relativo a la perspectiva temporal.

En la siguiente tabla se muestra para cada fichero de la MCVL, el número de registros que almacena así como el número de columnas de cada tabla. En la base de datos tenemos almacenados más de 24.000 millones de datos. Teniendo en cuenta que en la base de datos disponemos de 14 ediciones, en la tabla de personas hay casi 17 millones de registros mientras que en la de afiliación 284 millones. Estas son las tablas que vamos a utiliza en el ejemplo que vamos a ver.

Muchos de los registros son redundantes ya que en cada edición y para cada persona se almacena todo su historial de relaciones con la seguridad social. Se podría hacer un esfuerzo por optimizar este aspecto pero presentaría un inconveniente ya que complicaría diseño de la base de datos y por tanto, los accesos para extraer información de ella no sería tan intuitivos. Por este motivo se ha optado por cargar los datos recibidos de la Seguridad Social tal cual llegan.

FICHEROS	N.º Columnas	N.º Registros 2019	Total registros	N.º Datos (millones)
Personas	12	1.268.856	16.885.455	203
Afiliación	37	27.319.556	284.229.997	10.517
Bases de cotización (cuenta ajena)	24	27.167.657	332.472.894	7.979
Bases de cotización (cuenta propia)	22	5.670.023	67.818.163	1.492
Pensiones	47	5.228.133	57.449.781	2.700
Convivientes	23	1.224.460	16.062.613	369
Fiscales	51	2.103.999	27.389.875	1.397
				24.657

Ejemplo de código con la Base de Datos de la MCVL

```
#Cargamos las librerías
options(java.parameters = "-Xmx8g")
library(tidyverse)
library(RJDBC)
library(DBI)
library(dbplyr)

#Establecemos la conexión a la base de datos (en nuestro caso ORACLE)
source("./conectaBD.R")

#driver <- JDBC("oracle.jdbc.OracleDriver", classPath="./ojdbc6.jar")
#Err<-try(con<- dbConnect(driver, "jdbc:oracle:thin:@miservidor01:1521:MIBD","BASEDATOS","CLAVE"),silent=FALSE)

#Traductor necesario para Oracle
sql_translation.JDBCConnection <- dbplyr:::sql_translation.Oracle
sql_select.JDBCConnection <- dbplyr:::sql_query_select.Oracle
sql_subquery.JDBCConnection <- dbplyr:::sql_query_wrap.Oracle

ANOREF=2019

#Consulta del fichero de afiliación donde obtenemos aquellos episodios que cumplen las siguientes condiciones:
#   -Pertenecen al lote 2019
#   -Han estado activos en algún momento del año 2019
#Se recodifica la variable grupo de cotización según una recomendación de la guía de la MCVL de la
#Seguridad Social
#Por último, seleccionamos las variables que nos interesan
afi<- tbl(con, "MCVL_AFILIACION") %>%
      filter(LOTE==ANOREF) %>%
      filter(substr(FALTA, 1, 4)< ANOREF+1 & substr(FBAJA, 1, 4)>ANOREF-1) %>%
      mutate(GRUPCOT_R=if_else(!GRUPCOT%in%c('01','02','03','04','05','06','07','08','09','10','11','12'),'00',
                               GRUPCOT))%>%
      select(PK_ID,IDPF,GRUPCOT,GRUPCOT_R)
afi<-modificaID(afi)

head(afi)


#Del fichero de personas:
#   -Filtramos aquellas personas que forman parte de la edición de 2019
```

```
#   -Seleccionamos las columnas que nos interesan para unirlas al fichero de afiliación
personas <- tbl(con, "MCVL_PERSONAS")%>%
                 filter(LOTE==ANOREF) %>%
                 select(IDPF, SEXO)
personas<-modificaID(personas)


head(personas)


#Cruzamos el fichero de afiliación y de personas
afi_ampliado <- inner_join(afi,personas,by='ID')

head(afi_ampliado)


#Operamos y traemos el fichero obtenido a la consola de datos
episodios <- afi_ampliado %>% group_by(GRUPCOT_R,SEXO) %>% summarise(n=n()) %>% collect()

head(episodios)
#Podemos transformar el resultado para que se entienda mejor
episodios<-episodios%>% mutate(SEXO=recode(SEXO,'1'='Hombres','2'='Mujeres')) %>%
                        pivot_wider(names_from=SEXO, values_from=n) %>%
                        mutate(Total=Mujeres+Hombres) %>%
                        rename(Grupo_cot=GRUPCOT_R) %>%
                        arrange(Grupo_cot)


head(episodios)
```

```
 user  system elapsed
 0.34    0.10   36.94
```

Código compacto

```
options(java.parameters = "-Xmx8g")
#Cargamos las librerías
library(tidyverse)
library(RJDBC)
library(DBI)
library(dbplyr)
ANOREF=2019

#Establecemos la conexión con ORACLE:
driver <- JDBC("oracle.jdbc.OracleDriver", classPath="./ojdbc6.jar")
Err<-try(con<- dbConnect(driver, "jdbc:oracle:thin:@miservidor01:1521:MIBD", "BASEDATOS", "CLAVE"),silent=FALSE)

#La conexión JDBC actual apunta a la traducción de Oracle dentro dbplyr:
sql_translation.JDBCConnection <- dbplyr:::sql_translation.Oracle
sql_select.JDBCConnection <- dbplyr:::sql_query_select.Oracle
sql_subquery.JDBCConnection <- dbplyr:::sql_query_wrap.Oracle
```

```
afi <- tbl(con, "MCVL_AFILIACION") %>%
       filter(LOTE==ANOREF) %>%
       filter(substr(FALTA, 1, 4)< ANOREF+1 & substr(FBAJA, 1, 4)>ANOREF-1) %>%
       mutate(GRUPCOT_R = if_else(!GRUPCOT %in% c('01','02','03','04','05','06','07','08','09','10','11','12'),'00',GRUPCOT)) %>%
       select(PK_ID,IDPF,GRUPCOT,GRUPCOT_R)

personas <- tbl(con, "MCVL_PERSONAS") %>%
            filter(LOTE==ANOREF) %>%
            select(IDPF, SEXO)

afi_ampliado <- inner_join(afi,personas,by='IDPF')

episodios <- afi_ampliado %>% group_by(GRUPCOT_R,SEXO) %>% summarise(n=n()) %>% collect()
```

dbplyr

```
episodios <- episodios %>% mutate(SEXO=recode(SEXO,'1'='Hombres','2'='Mujeres')) %>%
                           pivot_wider(names_from=SEXO, values_from=n) %>%
                           mutate(Total=Mujeres+Hombres) %>%
                           rename(Grupo_cot=GRUPCOT_R) %>% arrange(Grupo_cot)
```

dplyr