# Datos Relacionales

Gabriel Illanes

12 de mayo de 2021

- Usualmente, manejaremos muchas bases de datos con información de interés. Como conjunto, son llamadas *bases de datos relacionales:* importa tanto la información que contienen, como la relación existente entre ellas.

- Usualmente, manejaremos muchas bases de datos con información de interés. Como conjunto, son llamadas *bases de datos relacionales:* importa tanto la información que contienen, como la relación existente entre ellas.
- En el fondo, las relaciones entre muchas tablas se reduce a las relaciones entre pares de tablas.

- Usualmente, manejaremos muchas bases de datos con información de interés. Como conjunto, son llamadas *bases de datos relacionales:* importa tanto la información que contienen, como la relación existente entre ellas.
- En el fondo, las relaciones entre muchas tablas se reduce a las relaciones entre pares de tablas.

## Verbos para datos relacionales

- ***Uniones de transformación (Mutating joins):*** *Agregan nuevas variables a un data frame a partir de las observaciones coincidentes en otra tabla*
- ***Uniones de filtro (Filtering joins):*** *Filtran observaciones con base en si coinciden o no con una observación en otra tabla.*
- ***Operaciones de conjuntos (Set operations):*** *Tratan a las observaciones como elementos de un conjunto.*

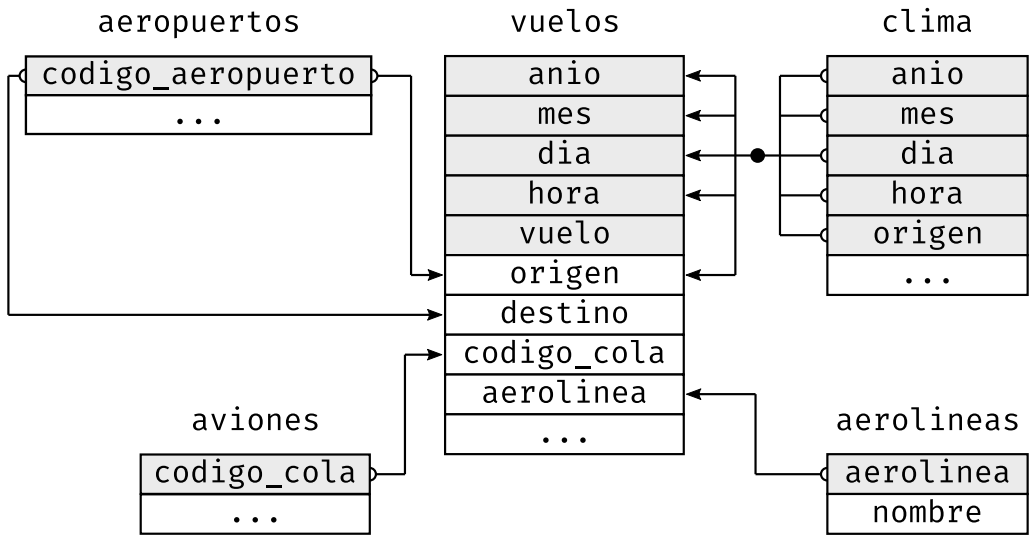
El setting es el usual.

```
library(tidyverse)
library(datos)
```

Y manejaremos los siguientes tibbles.

```
aerolineas
aeropuertos
aviones
clima
```

# Relaciones existentes



# Claves (Keys)

- `vuelos` se conecta con `aviones` a través de la variable `codigo_cola`.
- `vuelos` se conecta con `aerolineas` a través de la variable `codigo_carrier`.
- `vuelos` se conecta con `aeropuertos` a través de las variables `origen` y `destino`.
- `vuelos` se conecta con `clima` a través de la variables `origen`, `mes`, `dia` y `hora`.

# Claves (Keys)

- `vuelos` se conecta con `aviones` a través de la variable `codigo_cola`.
- `vuelos` se conecta con `aerolineas` a través de la variable `codigo_carrier`.
- `vuelos` se conecta con `aeropuertos` a través de las variables `origen` y `destino`.
- `vuelos` se conecta con `clima` a través de la variables `origen`, `mes`, `dia` y `hora`.

- *Una **clave primaria** identifica únicamente una observación en su propia tabla (`aviones$codigo_cola`).*
- *Una **clave foránea** únicamente identifica una observación en otra tabla, referenciando una clave primaria (`vuelos$codigo_cola`).*
- *Una clave primaria y su correspondiente clave foránea en otra tabla forman una **relación** (tipicamente uno-a-muchos).*

Verificamos que una clave es primaria...

```
aviones %>%
  count(codigo_cola) %>%
  filter(n > 1)

## # A tibble: 0 x 2
## # ... with 2 variables: codigo_cola <chr>, n <int>

clima %>%
  count(anio, mes, dia, hora, origen) %>%
  filter(n > 1)

## # A tibble: 3 x 6
##    anio   mes   dia  hora origen     n
##   <int> <int> <int> <int> <chr>  <int>
## 1  2013    11     3     1 EWR        2
## 2  2013    11     3     1 JFK        2
## 3  2013    11     3     1 LGA        2
```

Podemos generar claves primarias artificiales, por ejemplo, número de fila de la observación (claves subrogadas).

# Uniones de transformación (Mutating joins)

```
vuelos2 <- vuelos %>% # Data frame reducido para más comodidad
  select(anio:dia, hora, origen, destino, codigo_cola, aerolinea)
vuelos2
```

```
## # A tibble: 336,776 x 8
##     anio   mes   dia  hora origen destino codigo_cola aerolinea
##    <int> <int> <int> <dbl> <chr>  <chr>   <chr>       <chr>
##  1  2013     1     1     5 EWR    IAH     N14228      UA
##  2  2013     1     1     5 LGA    IAH     N24211      UA
##  3  2013     1     1     5 JFK    MIA     N619AA      AA
##  4  2013     1     1     5 JFK    BQN     N804JB      B6
##  5  2013     1     1     6 LGA    ATL     N668DN      DL
##  6  2013     1     1     5 EWR    ORD     N39463      UA
##  7  2013     1     1     6 EWR    FLL     N516JB      B6
##  8  2013     1     1     6 LGA    IAD     N829AS      EV
##  9  2013     1     1     6 JFK    MCO     N593JB      B6
## 10  2013     1     1     6 LGA    ORD     N3ALAA      AA
## # ... with 336,766 more rows
```

¿Cómo podemos incluir el nombre completo de la aerolínea en `vuelos2`?

```
vuelos2 %>%
  left_join(aerolineas, by = "aerolinea")

## # A tibble: 336,776 x 9
##     anio   mes   dia  hora origen destino codigo_cola aerolinea nombre
##    <int> <int> <int> <dbl> <chr>  <chr>   <chr>       <chr>     <chr>
##  1  2013     1     1     5 EWR    IAH     N14228      UA        United Air Line~
##  2  2013     1     1     5 LGA    IAH     N24211      UA        United Air Line~
##  3  2013     1     1     5 JFK    MIA     N619AA      AA        American Airlin~
##  4  2013     1     1     5 JFK    BQN     N804JB      B6        JetBlue Airways
##  5  2013     1     1     6 LGA    ATL     N668DN      DL        Delta Air Lines~
##  6  2013     1     1     5 EWR    ORD     N39463      UA        United Air Line~
##  7  2013     1     1     6 EWR    FLL     N516JB      B6        JetBlue Airways
##  8  2013     1     1     6 LGA    IAD     N829AS      EV        ExpressJet Airl~
##  9  2013     1     1     6 JFK    MCO     N593JB      B6        JetBlue Airways
## 10  2013     1     1     6 LGA    ORD     N3ALAA      AA        American Airlin~
## # ... with 336,766 more rows
```

Terminamos *uniendo* la columna `nombre` a `vuelos2`.

Trabajaremos con un ejemplo de juguete para entender mejor cómo funcionan las uniones.

```
x <- tribble(
  ~key, ~val_x,
  1, "x1",
  2, "x2",
  3, "x3"
)
y <- tribble(
  ~key, ~val_y,
  1, "y1",
  2, "y2",
  4, "y3"
)
```

Una unión es una forma de conectar cada fila en `x` con cero, una o más filas en `y`.

# Unión interior

Una unión interior une pares de observaciones, siempre que sus claves sean iguales. El resto de las observaciones no se incluyen en el resultado (hay que tener cuidado con esto!).

```
x %>%
  inner_join(y, by = "key")

## # A tibble: 2 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
```

# Unión exterior

Una unión exterior mantiene las observaciones que aparecen en, al menos, una de las tablas.

- *Una **unión izquierda** mantiene todas las observaciones en $x$.*
- *Una **unión derecha** mantiene todas las observaciones en $y$.*
- *Una **unión completa** mantiene todas las observaciones en $x$ e $y$.*

Si faltan valores, se rellena con `NA`.

```r
x %>%
  left_join(y, by = "key")
```

```
## # A tibble: 3 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     3 x3    <NA>
```

```r
x %>%
  right_join(y, by = "key")
```

```
## # A tibble: 3 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     4 <NA>  y3
```

```r
x %>%
  full_join(y, by = "key")
```

```
## # A tibble: 4 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     3 x3    <NA>
## 4     4 <NA>  y3
```

# Claves duplicadas

La cosa se complica cuando las claves no son únicas. Existen dos posibilidades:

1. Una tabla tiene claves duplicadas.

```
x <- tribble(
 ~key, ~val_x,
 1, "x1",
 2, "x2",
 2, "x3",
 1, "x4"
)
y <- tribble(
 ~key, ~val_y,
 1, "y1",
 2, "y2"
)
left_join(x, y, by = "key")

## # A tibble: 4 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     2 x3    y2
## 4     1 x4    y1
```

2 Ambas tablas tienen claves duplicadas.

```
x <- tribble(
 ~key, ~val_x,
 1, "x1",
 2, "x2",
 2, "x3",
 3, "x4"
  )
  y <- tribble(
 ~key, ~val_y,
 1, "y1",
 2, "y2",
 2, "y3",
 3, "y4"
  )
left_join(x, y, by = "key")

## # A tibble: 6 x 3
##     key val_x val_y
##   <dbl> <chr> <chr>
## 1     1 x1    y1
## 2     2 x2    y2
## 3     2 x2    y3
## 4     2 x3    y2
## 5     2 x3    y3
## 6     3 x4    y4
```

En ambos casos, obtenemos todas las posibles combinaciones dentro de cada clave.

# Definiendo las columnas clave

Hasta ahora solo unimos bases de datos a partir de una única variable, y esa variable tiene el mismo nombre en ambas tablas. En las funciones que estamos estudiando, esto se maneja con el argumento `by`.

- Por defecto `by = NULL`, usa todas las variables que aparecen en ambas tablas con el mismo nombre. Esta unión se conoce como **unión natural**.

```
vuelos2 %>%
 left_join(clima)

## Joining, by = c("anio", "mes", "dia", "hora", "origen")

## # A tibble: 336,776 x 18
##     anio   mes   dia  hora origen destino codigo_cola aerolinea temperatura
##    <int> <int> <int> <dbl> <chr>  <chr>   <chr>       <chr>           <dbl>
##  1  2013     1     1     5 EWR    IAH     N14228      UA               39.0
##  2  2013     1     1     5 LGA    IAH     N24211      UA               39.9
##  3  2013     1     1     5 JFK    MIA     N619AA      AA               39.0
##  4  2013     1     1     5 JFK    BQN     N804JB      B6               39.0
##  5  2013     1     1     6 LGA    ATL     N668DN      DL               39.9
##  6  2013     1     1     5 EWR    ORD     N39463      UA               39.0
##  7  2013     1     1     6 EWR    FLL     N516JB      B6               37.9
##  8  2013     1     1     6 LGA    IAD     N829AS      EV               39.9
##  9  2013     1     1     6 JFK    MCO     N593JB      B6               37.9
## 10  2013     1     1     6 LGA    ORD     N3ALAA      AA               39.9
## # ... with 336,766 more rows, and 9 more variables: punto_rocio <dbl>,
## #   humedad <dbl>, direccion_viento <dbl>, velocidad_viento <dbl>,
## #   velocidad_rafaga <dbl>, precipitacion <dbl>, presion <dbl>,
## #   visibilidad <dbl>, fecha_hora <dttm>
```

- Un vector de caracteres `by = "x"`. Con esto podemos usar solo algunas de las variables comunes, Por ejemplo, `vuelos` y `aviones` tienen la variable `anio`, pero significa distintas cosas en cada tabla, por lo cual unimos por `codigo_cola`.

```
vuelos2 %>%
 left_join(aviones, by = "codigo_cola")

## # A tibble: 336,776 x 16
##    anio.x   mes   dia  hora origen destino codigo_cola aerolinea anio.y tipo
##     <int> <int> <int> <dbl> <chr>  <chr>   <chr>       <chr>      <int> <chr>
##  1   2013     1     1     5 EWR    IAH     N14228      UA          1999 Ala fij~
##  2   2013     1     1     5 LGA    IAH     N24211      UA          1998 Ala fij~
##  3   2013     1     1     5 JFK    MIA     N619AA      AA          1990 Ala fij~
##  4   2013     1     1     5 JFK    BQN     N804JB      B6          2012 Ala fij~
##  5   2013     1     1     6 LGA    ATL     N668DN      DL          1991 Ala fij~
##  6   2013     1     1     5 EWR    ORD     N39463      UA          2012 Ala fij~
##  7   2013     1     1     6 EWR    FLL     N516JB      B6          2000 Ala fij~
##  8   2013     1     1     6 LGA    IAD     N829AS      EV          1998 Ala fij~
##  9   2013     1     1     6 JFK    MCO     N593JB      B6          2004 Ala fij~
## 10   2013     1     1     6 LGA    ORD     N3ALAA      AA            NA <NA>
## # ... with 336,766 more rows, and 6 more variables: fabricante <chr>,
## #   modelo <chr>, motores <int>, asientos <int>, velocidad <int>,
## #   tipo_motor <chr>
```

Las variables con el mismo nombre en las dos tablas que no son usadas para la unión, son desambiguadas con un sufijo (ver `anio`).

- Un vector de caracteres con nombres: `by = c("a" = "b")`. Esto va a unir la variable `a` en la tabla `x` con la variable `b` en la tabla `y`. Las variables de `x` se usarán en el output.

```
# Podemos unir el código de aeropuerto con el origen o destino
# de los vuelos.
# Si elegimos el origen:
vuelos2 %>%
 left_join(aeropuertos, c("origen" = "codigo_aeropuerto"))

## # A tibble: 336,776 x 15
##     anio   mes   dia  hora origen destino codigo_cola aerolinea nombre   latitud
##    <int> <int> <int> <dbl> <chr>  <chr>   <chr>       <chr>     <chr>      <dbl>
##  1  2013     1     1     5 EWR    IAH     N14228      UA        Newark ~    40.7
##  2  2013     1     1     5 LGA    IAH     N24211      UA        La Guar~    40.8
##  3  2013     1     1     5 JFK    MIA     N619AA      AA        John F ~    40.6
##  4  2013     1     1     5 JFK    BQN     N804JB      B6        John F ~    40.6
##  5  2013     1     1     6 LGA    ATL     N668DN      DL        La Guar~    40.8
##  6  2013     1     1     5 EWR    ORD     N39463      UA        Newark ~    40.7
##  7  2013     1     1     6 EWR    FLL     N516JB      B6        Newark ~    40.7
##  8  2013     1     1     6 LGA    IAD     N829AS      EV        La Guar~    40.8
##  9  2013     1     1     6 JFK    MCO     N593JB      B6        John F ~    40.6
## 10  2013     1     1     6 LGA    ORD     N3ALAA      AA        La Guar~    40.8
## # ... with 336,766 more rows, and 5 more variables: longitud <dbl>,
## #   altura <dbl>, zona_horaria <dbl>, horario_verano <chr>,
## #   zona_horaria_iana <chr>
```

# Uniones de filtro

Las uniones de filtro unen observaciones de a misma forma que las uniones de transformación, pero afectan a las obesrvaciones en vez de a las variables.

- *`semi_join(x, y)` **mantiene** todas las observaciones en $x$ con coincidencias en $y$.*
- *`anti_join(x, y)` **descarta** todas las observaciones en $x$ con coincidencias en $y$.*

Son útiles para unir tablas previamente filtradas.

```r
# Estos son los destinos más populares
destinos_populares <- vuelos %>%
  count(destino, sort = TRUE) %>%
  head(10)
destinos_populares
```

```
## # A tibble: 10 x 2
##    destino     n
##    <chr>   <int>
##  1 ORD     17283
##  2 ATL     17215
##  3 LAX     16174
##  4 BOS     15508
##  5 MCO     14082
##  6 CLT     14064
##  7 SFO     13331
##  8 FLL     12055
##  9 MIA     11728
## 10 DCA      9705
```

```r
# Buscamos vuelos a esos lugares
vuelos %>%
  filter(destino %in% destinos_populares$destino)
```

```
## # A tibble: 141,145 x 19
##     anio   mes   dia horario_salida salida_programada atraso_salida
##    <int> <int> <int>          <int>              <int>         <dbl>
##  1  2013     1     1            542                540             2
##  2  2013     1     1            554                600            -6
##  3  2013     1     1            554                558            -4
##  4  2013     1     1            555                600            -5
##  5  2013     1     1            557                600            -3
##  6  2013     1     1            558                600            -2
##  7  2013     1     1            558                600            -2
##  8  2013     1     1            558                600            -2
##  9  2013     1     1            559                559             0
## 10  2013     1     1            600                600             0
## # ... with 141,135 more rows, and 13 more variables: horario_llegada <int>,
## #   llegada_programada <int>, atraso_llegada <dbl>, aerolinea <chr>,
## #   vuelo <int>, codigo_cola <chr>, origen <chr>, destino <chr>,
## #   tiempo_vuelo <dbl>, distancia <dbl>, hora <dbl>, minuto <dbl>,
## #   fecha_hora <dttm>
```

```r
# Pero es más fácil y generalizable usar semi_join
# Busca las variables comunes, y preserva coincidencias
vuelos %>%
  semi_join(destinos_populares)

## Joining, by = "destino"

## # A tibble: 141,145 x 19
##     anio   mes   dia horario_salida salida_programada atraso_salida
##    <int> <int> <int>          <int>              <int>         <dbl>
##  1  2013     1     1            542                540             2
##  2  2013     1     1            554                600            -6
##  3  2013     1     1            554                558            -4
##  4  2013     1     1            555                600            -5
##  5  2013     1     1            557                600            -3
##  6  2013     1     1            558                600            -2
##  7  2013     1     1            558                600            -2
##  8  2013     1     1            558                600            -2
##  9  2013     1     1            559                559             0
## 10  2013     1     1            600                600             0
## # ... with 141,135 more rows, and 13 more variables: horario_llegada <int>,
## #   llegada_programada <int>, atraso_llegada <dbl>, aerolinea <chr>,
## #   vuelo <int>, codigo_cola <chr>, origen <chr>, destino <chr>,
## #   tiempo_vuelo <dbl>, distancia <dbl>, hora <dbl>, minuto <dbl>,
## #   fecha_hora <dttm>
```

Las anti uniones son útiles para encontrar desajustes.

```
# Hay vuelos con código de cola que
# no estén en la base de datos "aviones"?
vuelos %>%
  anti_join(aviones, by = "codigo_cola") %>%
  count(codigo_cola, sort = TRUE)
```

```
## # A tibble: 722 x 2
##    codigo_cola     n
##    <chr>       <int>
##  1 <NA>         2512
##  2 N725MQ        575
##  3 N722MQ        513
##  4 N723MQ        507
##  5 N713MQ        483
##  6 N735MQ        396
##  7 NOEGMQ        371
##  8 N534MQ        364
##  9 N542MQ        363
## 10 N531MQ        349
## # ... with 712 more rows
```

Los datos que estamos manejando en esta presentación son super amigables. En general, los datos son más complicados de manejar. Algunas cosas a tener en cuenta son:

1. Identificar claves primarias en cada tabla, teniendo en cuenta su significado e interpretación.
2. Verificar que no haya datos faltantes en las claves primarias.
3. Verificar la coincidencia de claves foráneas con claves primarias (es buena idea usar `anti_join`!).
4. Verificar cantidad de filas antes y después de unir suele no ser una buena práctica.

# Operaciones de conjuntos

También podemos tratar a las bases de datos como conjuntos de observaciones, y usar operaciones de conjuntos. Estas funciones no son tan utilizadas, pero pueden ser útiles.

- *intersect(x, y): devuelve las observaciones comunes en x e y.*
- *union(x, y): devuelve las observaciones únicas en x e y.*
- *setdiff(x, y): devuelve las observaciones en x que no están en y.*

Trabajemos de vuelta con datos simplificados

```
df1 <- tribble(
  ~x, ~y,
  1, 1,
  2, 1
)
df2 <- tribble(
  ~x, ~y,
  1, 1,
  1, 2
)
```

```
intersect(df1, df2)
```

```
## # A tibble: 1 x 2
##       x     y
##   <dbl> <dbl>
## 1     1     1
```

```
union(df1, df2)
```

```
## # A tibble: 3 x 2
##       x     y
##   <dbl> <dbl>
## 1     1     1
## 2     2     1
## 3     1     2
```

```
setdiff(df1, df2)
```

```
## # A tibble: 1 x 2
##       x     y
##   <dbl> <dbl>
## 1     2     1
```