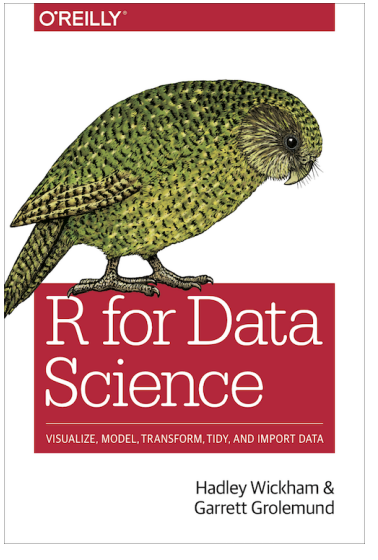
Introdução a Dados Relacionais

Benilton Carvalho, Guilherme Ludwig, Tatiana Benaglia

Dados em múltiplas tabelas

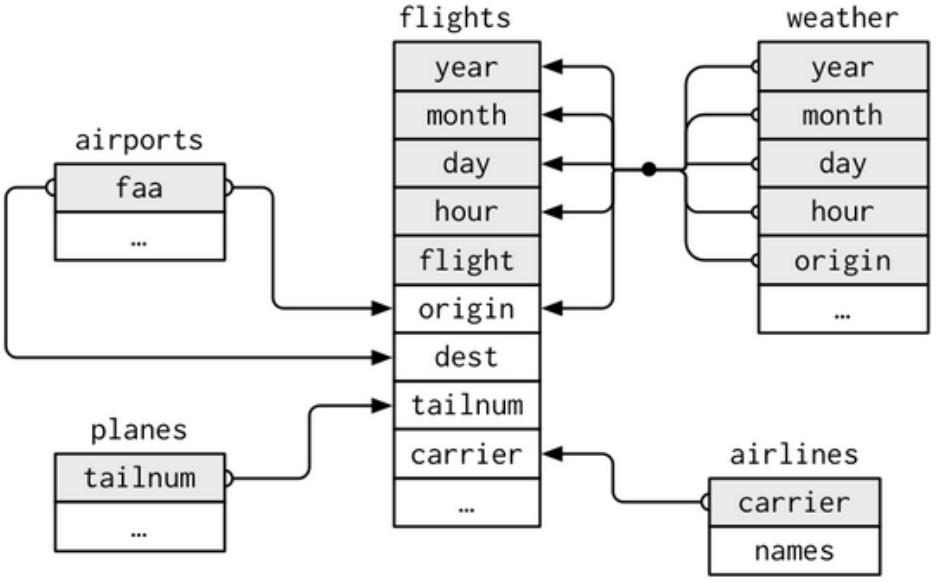
- É comum que dados estejam guardados em múltiplas tabelas. Esse modelo de banco de dados é conhecido como *Modelo Relacional* (https://en.wikipedia.org/wiki/Relational_model), em que os dados são acessados através de um *nome de tabela*, uma *chave* (*key*) e uma *coluna* (*features*).
- Se espera que, em no mínimo uma tabela, a chave identifique unicamente cada observação.
- O material da aula é baseado no capítulo 13 do livro *R for Data Science* (Wickham & Grolemund, 2017). Leiam o capítulo para verem exemplos adicionais: http://r4ds.had.co.nz/relational-data.html



Exemplo de Dados Relacionais

O pacote `nycflights13` contém 4 tabelas (`airports`, `planes`, `weather` `eairlines`) que são relacionadas com a tabela `flights`.

As relações podem ser visualizadas no seguinte diagrama:

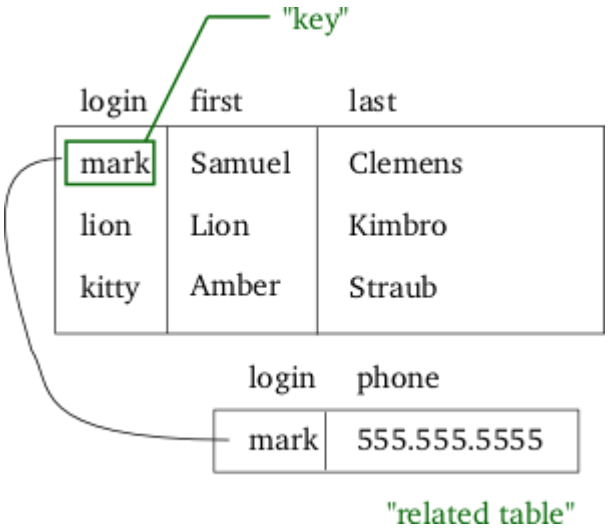


Fonte: R for Data Science, Capítulo 13. Wickham & Grolemund, 2017.

Chaves (*Keys*)

As variáveis usadas para conectar cada par de tabelas são chamadas de **chaves**.

Chave: é uma variável (ou conjunto de variáveis) que identifica unicamente uma observação.



Exemplo de base relacional: Figura de Wikipedia - Relation Model

Exemplo: Super-heróis e Editoras

Qual é a chave que conecta as duas tabelas de dados: superheroes e publishers?

superheroes

name	alignment	gender	publisher
Magneto	bad	male	Marvel
Storm	good	female	Marvel
Mystique	bad	female	Marvel
Batman	good	male	DC
Joker	bad	male	DC
Catwoman	bad	female	DC
Hellboy	good	male	Dark Horse Comics

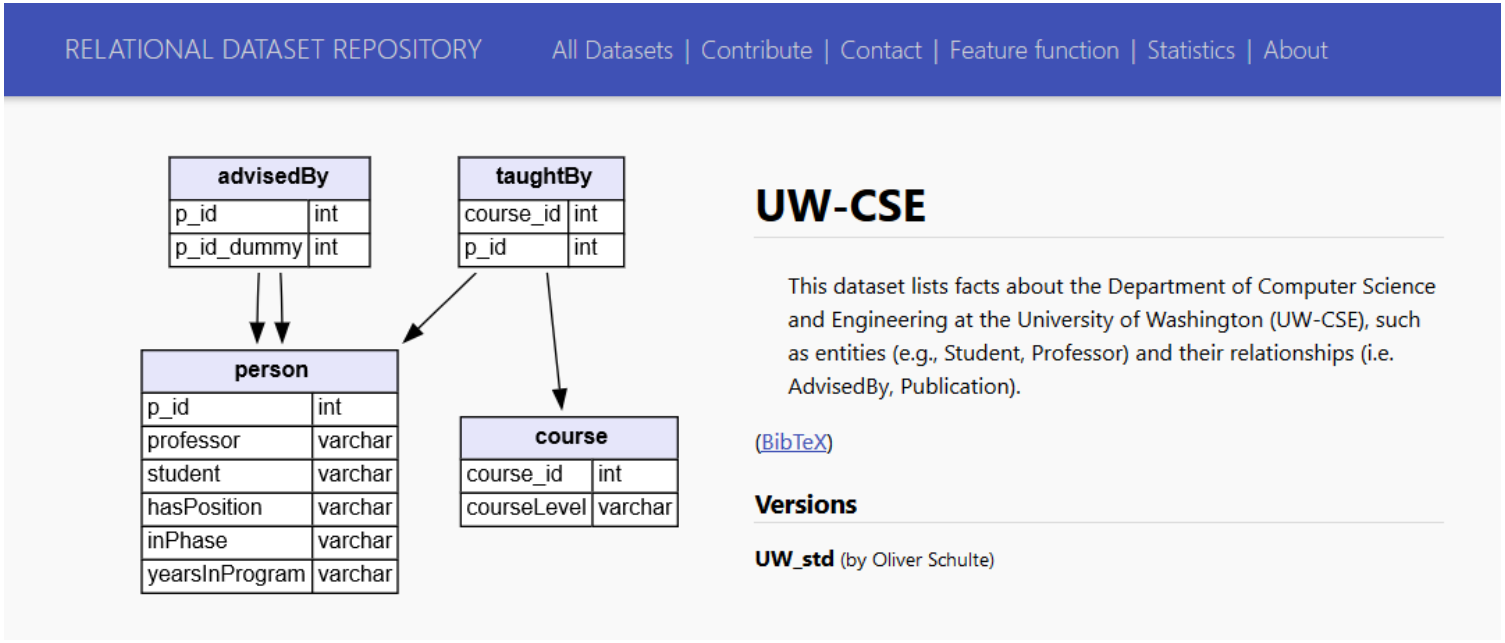
publishers

publisher	yr_founded
DC	1934
Marvel	1939
Image	1992

Consultas

- Cada tabela, separadamente, funciona como os bancos de dados com que trabalhamos até agora.
- Uma coluna em comum entre as tabelas será usada como chave, ligando a informação de cada linha. Porém, não há garantias que o valor seja único, nem sempre qual coluna servirá de chave é óbvio.
- Uma *consulta* (ou *query*) é um pedido do usuário ao *relational database management system* (RDBMS) que une informações de um grupo de indivíduos (baseados na chave) ao longo de várias tabelas.
- Nós vamos, primeiramente, examinar a operação *join*, do pacote `dplyr`, para realizar consultas em pares de tabelas.

Exemplo



Professores e alunos da University of Washington, ciência da computação.

Dados: https://relational.fit.cvut.cz/dataset/UW-CSE

Explicação: http://aiweb.cs.washington.edu/ai/mln/database.html

Recuperando dados do MySQL server

Código apenas para a reprodução do exemplo. SQL será abordado só em aulas futuras.

```
library(RMySQL)
mydb <- dbConnect(MySQL(), user='guest', password='relational',
                  dbname='UW_std', port = 3306,
                  host='relational.fit.cvut.cz')
rs <- dbSendQuery(mydb, "SELECT * FROM advisedBy")
advisedBy <- fetch(rs, n=-1)
rs <- dbSendQuery(mydb, "SELECT * FROM course")
course <- fetch(rs, n=-1)
rs <- dbSendQuery(mydb, "SELECT * FROM person")
person <- fetch(rs, n=-1)
rs <- dbSendQuery(mydb, "SELECT * FROM taughtBy")
taughtBy <- fetch(rs, n=-1)
dbDisconnect(mydb)
write.csv(advisedBy, "a03-advisedBy.csv", row.names = FALSE)
write.csv(course, "a03-course.csv", row.names = FALSE)
write.csv(person, "a03-person.csv", row.names = FALSE)
write.csv(taughtBy, "a03-taughtBy.csv", row.names = FALSE)
```

advisedBy

```
advisedBy %>% as_tibble
```

```
## # A tibble: 113 x 2
##     p_id p_id_dummy
##    <int>      <int>
##  1    96          5
##  2   118          5
##  3   183          5
##  4   263          5
##  5   362          5
##  6   266          7
##  7   272          7
##  8     6         29
##  9   242         29
## 10   303         29
## # … with 103 more rows
```

`p_id` orienta `p_id_dummy`.

course

```
course %>% as_tibble
```

```
## # A tibble: 132 x 2
##    course_id courseLevel
##        <int> <chr>
##  1         5 Level_300
##  2        11 Level_300
##  3        18 Level_300
##  4       104 Level_300
##  5       124 Level_300
##  6       146 Level_300
##  7       147 Level_300
##  8       165 Level_300
##  9         8 Level_400
## 10        20 Level_400
## # … with 122 more rows
```

`level_100` (introdução), `level_300` (graduação, segundo ano), `level_400` (graduação, avançado) e `level_500` (pós-graduação).

taughtBy

```
taughtBy %>% as_tibble
```

```
## # A tibble: 189 x 2
##    course_id  p_id
##        <int> <int>
##  1         0    40
##  2         1    40
##  3         2   180
##  4         3   279
##  5         4   107
##  6         7   415
##  7         8   297
##  8         9   235
##  9        11    52
## 10        11    57
## # … with 179 more rows
```

Qual curso em `course_id` e `p_id` de quem ensinou.

person

```
person %>% as_tibble
```

```
## # A tibble: 278 x 6
##     p_id professor student hasPosition inPhase       yearsInProgram
##    <int>     <int>   <int> <chr>       <chr>         <chr>
##  1     3         0       1 0           0             0
##  2     4         0       1 0           0             0
##  3     5         1       0 Faculty     0             0
##  4     6         0       1 0           Post_Quals    Year_2
##  5     7         1       0 Faculty_adj 0             0
##  6     9         0       1 0           Post_Generals Year_5
##  7    13         0       1 0           Post_Generals Year_7
##  8    14         0       1 0           Post_Generals Year_10
##  9    15         0       1 0           Post_Quals    Year_3
## 10    18         0       1 0           Pre_Quals     Year_3
## # … with 268 more rows
```

- `p_id` identifica indivíduos unicamente em `person`;

Tabelas não são 1-1

```
# Same course, different faculty
taughtBy %>% filter(course_id == 11)
```

```
##   course_id p_id
## 1        11   52
## 2        11   57
## 3        11  298
## 4        11  324
## 5        11  331
```

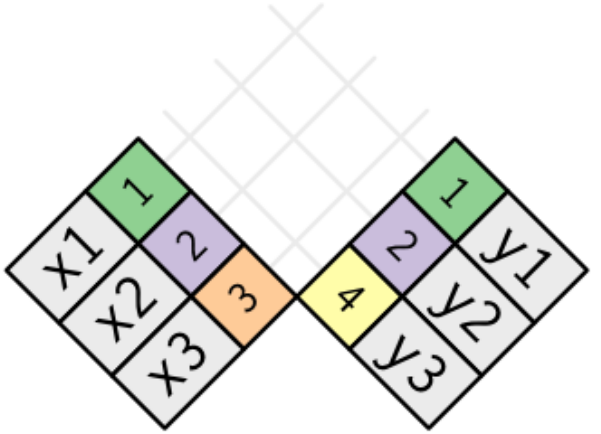
```
# Same faculty, different course
taughtBy %>% filter(p_id == 40)
```

```
##   course_id p_id
## 1         0   40
## 2         1   40
```

- `p_id` identifica indivíduos unicamente em `person`;
- `course_id` identifica cursos unicamente em `courses`.

Objetivo da aula de hoje

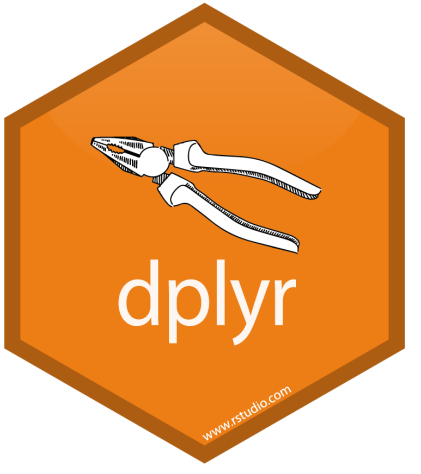
- Como relacionar informação de diferentes tabelas?
- Por exemplo, é mais comum que professores adjuntos ensinem classes de pós-graduação?
- Nós sabemos trabalhar com tabelas isoladas. Para duas ou mais tabelas, consideraremos funções do tipo **JOIN**.
- Um *join* é uma maneira de conectar cada linha de uma tabela a nenhuma, uma ou várias linhas de outra.
- Examinaremos a operação *join* do pacote `dplyr`, para realizar consultas em pares de tabelas.



Pacote `dplyr`

Para trabalhar com dados relacionais, você precisará conhecer verbos que lidem com pares de tabelas.

No pacote `dplyr`, existem três famílias de verbos que são usados para esse fim:

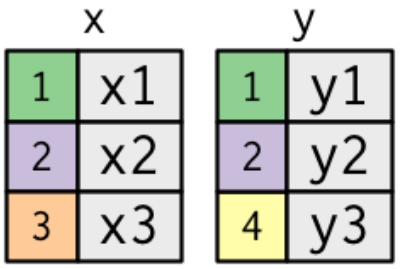


- **Mutating Joins**: adicionam novas variáveis a uma tabela de dados a partir de observações correspondentes em outra tabela.
- **Filtering Joins**: filtram observações de uma tabela baseadas no fato de corresponderem ou não a uma observação em outra tabela.
- **Set Operations**: combinam as observações das tabelas como se fossem conjuntos de elementos.

Começaremos essa aula falando dos **Mutating Joins**.

Tipos de JOIN: setup

Usando os diagramas de Wickham and Grolemund (2017), considere dados de duas tabelas:



A coluna colorida é a chave; `x` e `y` são colunas, tomando valores `x1`, `x2`, etc.

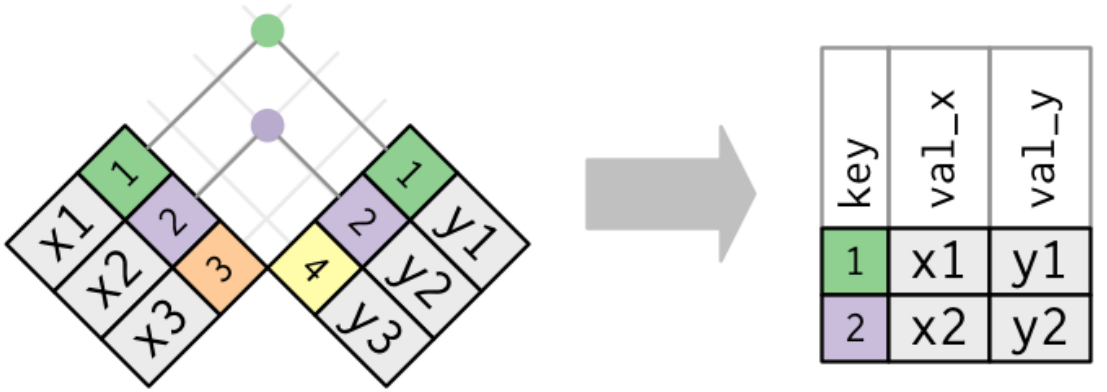
Vamos criar essas tabelas de dados no R:

```
x <- data.frame(key = c(1,2,3),
                val_x = c("x1","x2","x3"))

y <- data.frame(key = c(1,2,4),
                val_y = c("y1","y2","y4"))
```

INNER JOIN: inner_join

`inner_join(x,y)`: mantém somente as observações que estão em ambos `x` e `y`.




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

```
##   key val_x val_y
## 1   1    x1    y1
## 2   2    x2    y2
```

Exemplo: Super-heróis e Editoras

```
publishers %>% inner_join(superheroes, by = "publisher")
```

publishers

publisher	yr_founded
DC	1934
Marvel	1939
Image	1992

superheroes

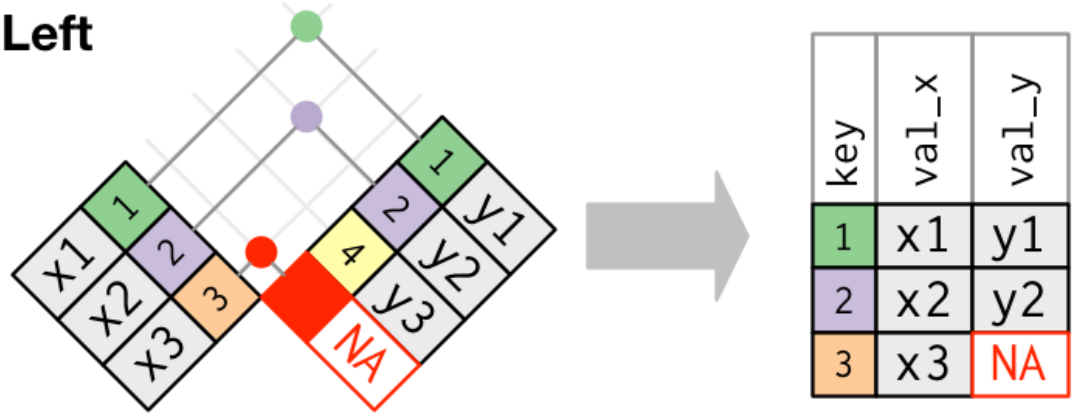
name	alignment	gender	publisher
Magneto	bad	male	Marvel
Storm	good	female	Marvel
Mystique	bad	female	Marvel
Batman	good	male	DC
Joker	bad	male	DC
Catwoman	bad	female	DC
Hellboy	good	male	Dark Horse Comics

inner_join(x = publishers, y = superheroes)

publisher	yr_founded	name	alignment	gender
DC	1934	Batman	good	male
DC	1934	Joker	bad	male
DC	1934	Catwoman	bad	female
Marvel	1939	Magneto	bad	male
Marvel	1939	Storm	good	female
Marvel	1939	Mystique	bad	female

OUTER JOIN: left_join

`left_join(x, y)`: une as linhas correspondentes de `y` em `x`.




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

```
##   key val_x val_y
## 1   1    x1    y1
## 2   2    x2    y2
## 3   3    x3  <NA>
```

Exemplo: Super-heróis e Editoras

```
publishers %>% left_join(superheroes, by = "publisher")
```

publishers

publisher	yr_founded
DC	1934
Marvel	1939
Image	1992

superheroes

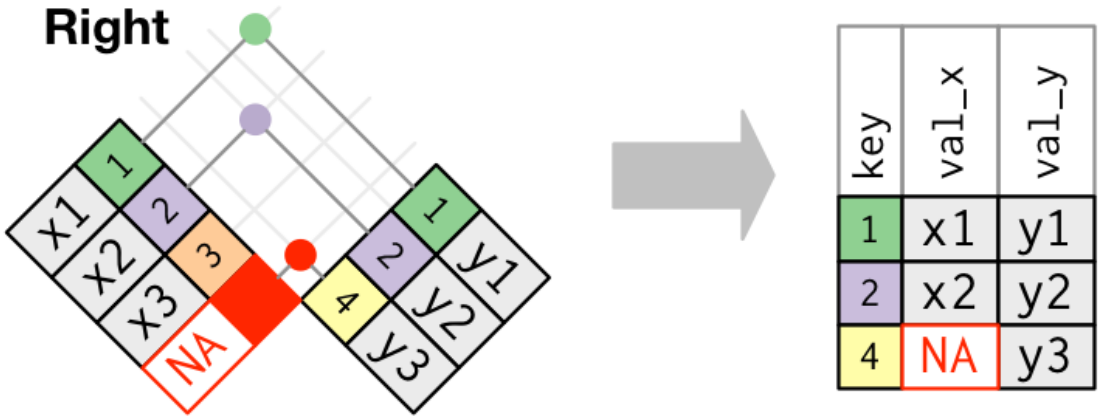
name	alignment	gender	publisher
Magneto	bad	male	Marvel
Storm	good	female	Marvel
Mystique	bad	female	Marvel
Batman	good	male	DC
Joker	bad	male	DC
Catwoman	bad	female	DC
Hellboy	good	male	Dark Horse Comics

left_join(x = publishers, y = superheroes)

publisher	yr_founded	name	alignment	gender
DC	1934	Batman	good	male
DC	1934	Joker	bad	male
DC	1934	Catwoman	bad	female
Marvel	1939	Magneto	bad	male
Marvel	1939	Storm	good	female
Marvel	1939	Mystique	bad	female
Image	1992	NA	NA	NA

OUTER JOIN: right_join

`right_join(x, y)`: une as linhas correspondentes de `x` em `y`.




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

```
##   key val_x val_y
## 1   1    x1    y1
## 2   2    x2    y2
## 3   4  <NA>    y4
```

Exemplo: Super-heróis e Editoras

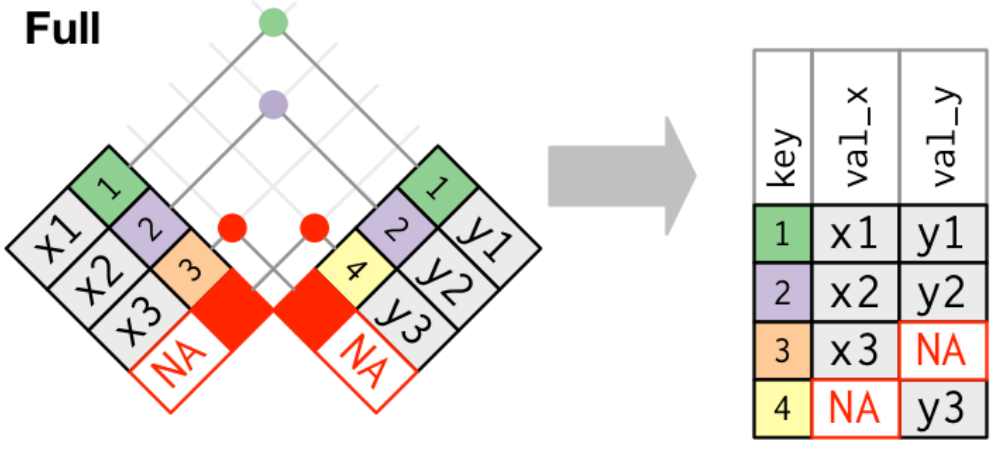
```
publishers %>% right_join(superheroes)
```

```
## Joining, by = "publisher"
```

```
## # A tibble: 7 x 5
##   publisher          yr_founded name     alignment gender
##   <chr>                   <dbl> <chr>    <chr>     <chr>
## 1 DC                       1934 Batman   good      male
## 2 DC                       1934 Joker    bad       male
## 3 DC                       1934 Catwoman bad       female
## 4 Marvel                   1939 Magneto  bad       male
## 5 Marvel                   1939 Storm    good      female
## 6 Marvel                   1939 Mystique bad       female
## 7 Dark Horse Comics          NA Hellboy  good      male
```

OUTER JOIN: full_join

`full_join(x, y)`: mantém todas as observações e valores de `x` e `y`.




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

```
##   key val_x val_y
## 1   1    x1    y1
## 2   2    x2    y2
## 3   3    x3  <NA>
## 4   4  <NA>    y4
```

Exemplo: Super-heróis e Editoras

```
superheroes %>% full_join(publishers, by = "publisher")
```

superheroes

name	alignment	gender	publisher
Magneto	bad	male	Marvel
Storm	good	female	Marvel
Mystique	bad	female	Marvel
Batman	good	male	DC
Joker	bad	male	DC
Catwoman	bad	female	DC
Hellboy	good	male	Dark Horse Comics

publishers

publisher	yr_founded
DC	1934
Marvel	1939
Image	1992

full_join(x = superheroes, y = publishers)

name	alignment	gender	publisher	yr_founded
Magneto	bad	male	Marvel	1939
Storm	good	female	Marvel	1939
Mystique	bad	female	Marvel	1939
Batman	good	male	DC	1934
Joker	bad	male	DC	1934
Catwoman	bad	female	DC	1934
Hellboy	good	male	Dark Horse Comics	NA
NA	NA	NA	Image	1992

Exemplo: que professores dão quais aulas?

Todos os professores de todos os cursos:

```
person %>%
  right_join(taughtBy, by = "p_id") %>%
  as_tibble
```

```
## # A tibble: 189 x 7
##     p_id professor student hasPosition inPhase       yearsInProgram course
##    <int>     <int>   <int> <chr>       <chr>         <chr>                <i
##  1     5         1       0 Faculty     0             0
##  2     5         1       0 Faculty     0             0
##  3     5         1       0 Faculty     0             0
##  4     9         0       1 0           Post_Generals Year_5
##  5    18         0       1 0           Pre_Quals     Year_3
##  6    22         1       0 Faculty_eme 0             0
##  7    40         1       0 Faculty     0             0
##  8    40         1       0 Faculty     0             0
##  9    46         1       0 Faculty     0             0
## 10    46         1       0 Faculty     0             0
## # … with 179 more rows
```

Exemplo: que professores dão quais aulas?

Vamos agora incluir o nível do curso.

```
person %>%
  right_join(taughtBy, by='p_id') %>%
  left_join(course, by='course_id') %>%
  as_tibble() %>%
  select(-professor, -student)
```

```
## # A tibble: 189 x 6
##     p_id hasPosition inPhase       yearsInProgram course_id courseLevel
##    <int> <chr>       <chr>         <chr>                <int> <chr>
##  1     5 Faculty     0             0                       19 Level_500
##  2     5 Faculty     0             0                       51 Level_400
##  3     5 Faculty     0             0                       71 Level_500
##  4     9 0           Post_Generals Year_5                 124 Level_300
##  5    18 0           Pre_Quals     Year_3                  51 Level_400
##  6    22 Faculty_eme 0             0                       21 Level_400
##  7    40 Faculty     0             0                        0 Level_500
##  8    40 Faculty     0             0                        1 Level_500
##  9    46 Faculty     0             0                      124 Level_300
## 10    46 Faculty     0             0                      172 Level_500
## # … with 179 more rows
```

Exemplo: que professores dão quais aulas?

Alguns estudantes ensinam classes avançadas.

```
person %>% right_join(taughtBy, by='p_id') %>%
  left_join(course, by='course_id') %>%
  filter(student == 1) %>%
  as_tibble() %>% select(-professor)
```

```
## # A tibble: 9 x 7
##    p_id student hasPosition inPhase       yearsInProgram course_id courseL
##   <int>   <int> <chr>       <chr>         <chr>                <int> <chr>
## 1     9       1 0           Post_Generals Year_5                 124 Level_3
## 2    18       1 0           Pre_Quals     Year_3                  51 Level_4
## 3    75       1 0           Post_Generals Year_6                 165 Level_3
## 4    99       1 0           Post_Quals    Year_2                  21 Level_4
## 5   141       1 0           Post_Generals Year_6                 165 Level_3
## 6   204       1 0           Post_Generals Year_6                  38 Level_4
## 7   255       1 0           Post_Generals Year_5                  38 Level_4
## 8   263       1 0           Post_Generals Year_6                  49 Level_4
## 9   278       1 0           Pre_Quals     Year_2                 144 Level_5
```

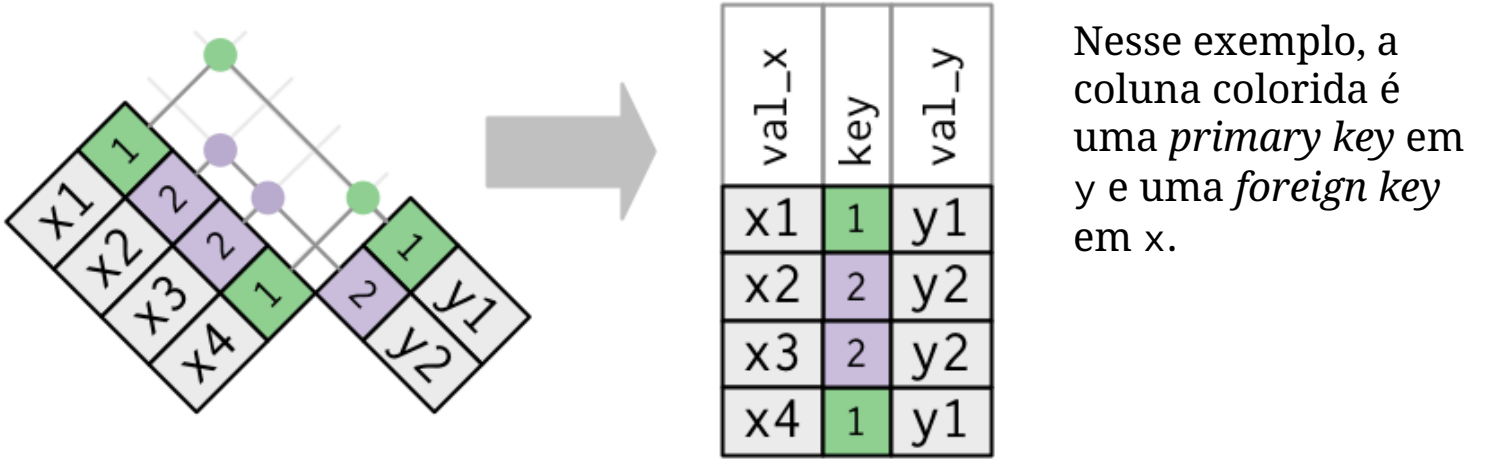
Exemplo: que professores dão quais aulas?

```
person %>% right_join(taughtBy, by='p_id') %>%
  left_join(course, by='course_id') %>%
  filter(student == 0) %>% group_by(hasPosition, courseLevel) %>%
  tally()
```

```
## # A tibble: 12 x 3
## # Groups:   hasPosition [5]
##    hasPosition courseLevel     n
##    <chr>       <chr>       <int>
##  1 0           Level_300       9
##  2 0           Level_400       7
##  3 0           Level_500       3
##  4 Faculty     Level_300      17
##  5 Faculty     Level_400      54
##  6 Faculty     Level_500      80
##  7 Faculty_adj Level_400       2
##  8 Faculty_aff Level_400       1
##  9 Faculty_aff Level_500       2
## 10 Faculty_eme Level_300       1
## 11 Faculty_eme Level_400       3
## 12 Faculty_eme Level_500       1
```

Chaves Duplicadas (*Duplicated keys*)

Como nós vimos no exemplo dos professores, chaves duplicadas em uma tabela não causam problema, desde que você escolha um join apropriado.



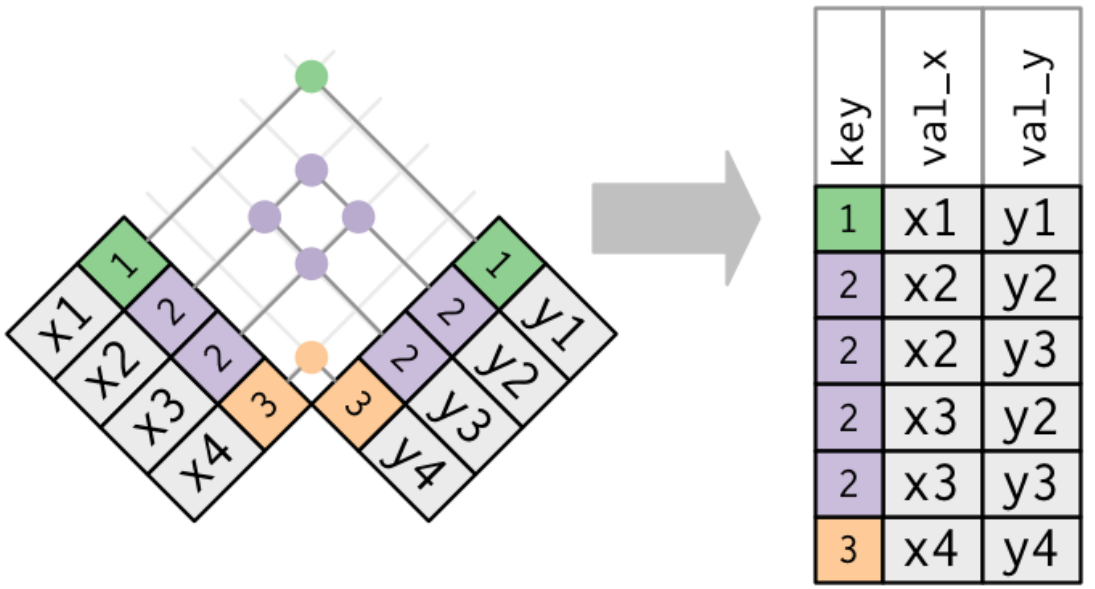
val_x	key	val_y
x1	1	y1
x2	2	y2
x3	2	y2
x4	1	y1

Nesse exemplo, a coluna colorida é uma *primary key* em `y` e uma *foreign key* em `x`.

Quando as chaves são únicas, elas são chamadas de **primary keys**; se há entradas repetidas, elas são chamadas de **foreign keys**. Os valores associados a primary keys são repetidos na tabela final.

Chaves Duplicadas

Quando ambas as tabelas têm chaves duplicadas, ao fazer um join, será executado um produto cartesiano das entradas.



key	val_x	val_y
1	x1	y1
2	x2	y2
2	x2	y3
2	x3	y2
2	x3	y3
3	x4	y4

Evite joins assim! Em tese, as bases relacionais devem ter pelo menos uma chave que unicamente determina as observações em cada tabela.

Sintaxe do argumento "by"

A ação padrão (*default*) das funções `*_join(x, y)` no `dplyr` é `by = NULL`, que realiza o join pela combinação de *todas* as colunas com nomes idênticos em x e y. Isso pode ser perigoso!

```
x$newCol <- c(1, 1, 2)
y$newCol <- c(1, 2, 2)
full_join(x, y)
```

```
## Joining, by = c("key", "newCol")
```

```
##   key val_x newCol val_y
## 1   1    x1      1    y1
## 2   2    x2      1  <NA>
## 3   3    x3      2  <NA>
## 4   2  <NA>      2    y2
## 5   4  <NA>      2    y4
```

```
x$newCol <- NULL
y$newCol <- NULL
```

Sintaxe do argumento "by"

Já `by = "colName"` une as observações pelo "colName" especificado.

```
full_join(x, y, by = "key")
```

```
##   key val_x val_y
## 1   1    x1    y1
## 2   2    x2    y2
## 3   3    x3  <NA>
## 4   4  <NA>    y4
```

Caso você queira comparar diferentes colunas, a sintaxe é `by = c("colunaX" = "colunaY")`. Note que o `R` remove `key` de `y` sem avisar!

```
x$newKey <- c(1,4,2)
full_join(x, y, by = c("newKey" = "key"))
```

```
##   key val_x newKey val_y
## 1   1    x1      1    y1
## 2   2    x2      4    y4
## 3   3    x3      2    y2
```

```
x$newKey <- NULL
```

Outras Implementações: `merge`

A função `merge()` da base do R pode executar todos os tipos de *mutating joins*:

dplyr	merge
`inner_join(x, y)`	`merge(x, y)`
`left_join(x, y)`	`merge(x, y, all.x = TRUE)`
`right_join(x, y)`	`merge(x, y, all.y = TRUE)` ,
`full_join(x, y)`	`merge(x, y, all.x = TRUE, all.y = TRUE)`

Outras Implementações: SQL

SQL serviu de inspiração para os verbos do `dplyr`:

dplyr	SQL
`inner_join(x, y, by = "z")`	`SELECT * FROM x INNER JOIN y USING (z)`
`left_join(x, y, by = "z")`	`SELECT * FROM x LEFT OUTER JOIN y USING (z)`
`right_join(x, y, by = "z")`	`SELECT * FROM x RIGHT OUTER JOIN y USING (z)`
`full_join(x, y, by = "z")`	`SELECT * FROM x FULL OUTER JOIN y USING (z)`

Filtering joins

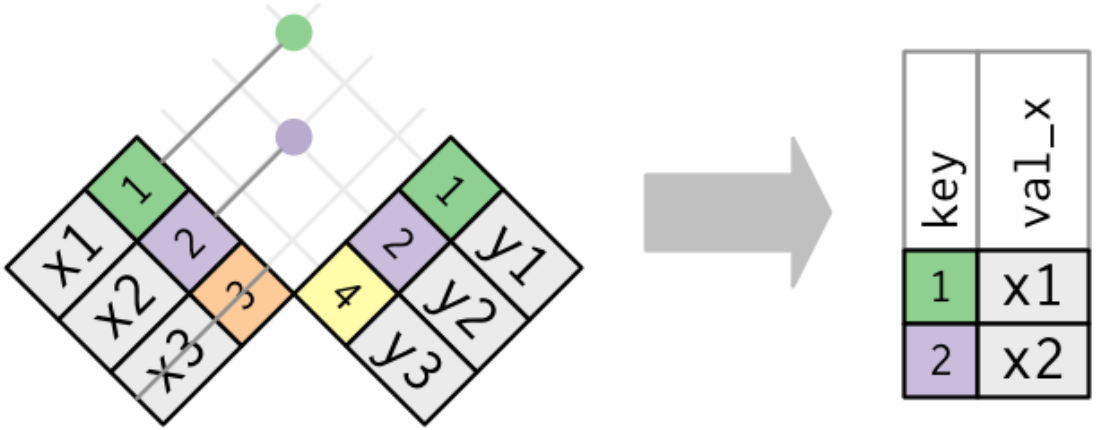
Há dois importantes filtering joins:

- `semi_join(x, y)` mantém todas as observações em `x` que estão presentes em `y`.
- `anti_join(x, y)` remove todas as observações em `x` que estão presentes em `y`.

Esses `*_join` retornam tabelas `x` filtradas, e não unem `x` e `y`.

semi_join

`semi_join(x,y)`: só retorna elementos de `x` que também estão em `y`.



Exemplo:

```
all.equal(x %>% semi_join(y,  by = "key"),
          x %>% filter(key %in% y$key))
```

```
## [1] TRUE
```

Exemplo: Super-heróis e Editoras

```
publishers %>% semi_join(superheroes, by = "publisher")
```

publishers

publisher	yr_founded
DC	1934
Marvel	1939
Image	1992

superheroes

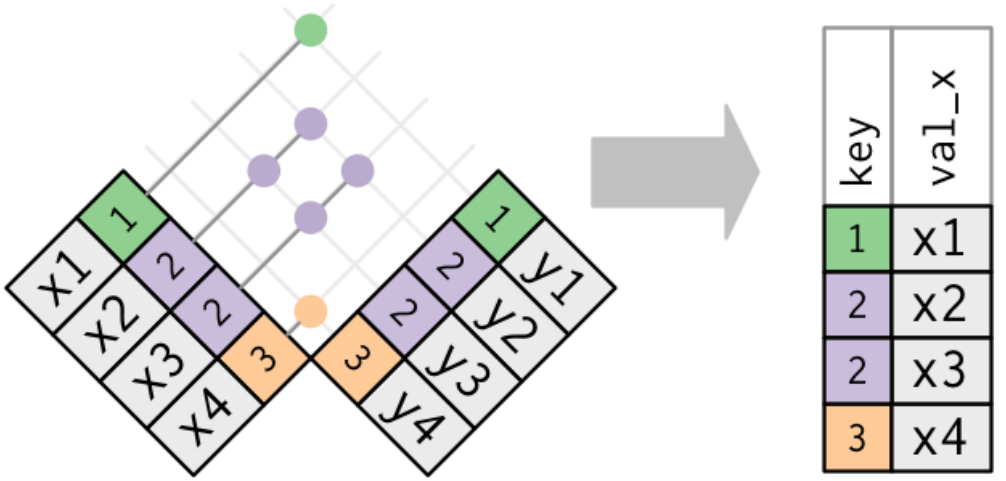
name	alignment	gender	publisher
Magneto	bad	male	Marvel
Storm	good	female	Marvel
Mystique	bad	female	Marvel
Batman	good	male	DC
Joker	bad	male	DC
Catwoman	bad	female	DC
Hellboy	good	male	Dark Horse Comics

semi-join(x = publishers, y = superheroes)

publisher	yr_founded
DC	1934
Marvel	1939

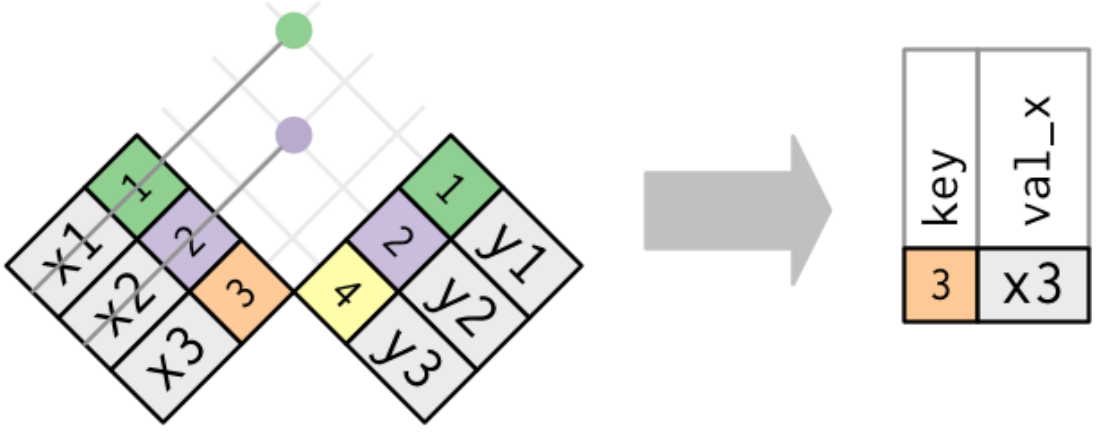
semi_join: chaves duplicadas

Não há problema se as chaves forem duplicadas para o `semi_join`, isto é, o `semi_join` não duplica as linhas.



anti_join

`anti_join(x,y)`: só retorna elementos de `x` que **não** estão em `y`.



É útil para detectar se há chaves faltantes em uma tabela.

Exemplo:

```
all.equal(x %>% anti_join(y,  by = "key"),
          x %>% filter(!(key %in% y$key)))
```

```
## [1] TRUE
```

Exemplo: Super-heróis e Editoras

```
publishers %>% anti_join(superheroes, by = "publisher")
```

publishers

publisher	yr_founded
DC	1934
Marvel	1939
Image	1992

superheroes

name	alignment	gender	publisher
Magneto	bad	male	Marvel
Storm	good	female	Marvel
Mystique	bad	female	Marvel
Batman	good	male	DC
Joker	bad	male	DC
Catwoman	bad	female	DC
Hellboy	good	male	Dark Horse Comics

anti_join(x = publishers, y = superheroes)

publisher	yr_founded
Image	1992

Set Operations

Esses verbos são usados com menor frequência, mas ocasionalmente podem ser úteis.

Aqui, espera-se que as tabelas `x` e `y` tenham as mesmas variáveis e as observações são tratados como conjuntos de elementos.

Todas essas operações usam a linha completa, comparando os valores de todas as variáveis.

Vamos usar essas duas tabelas como exemplo:

x

1	a
1	b
2	a

y

1	a
2	b

Interseção

`intersect(x, y)`: retorna apenas as observações que estão em ambos `x` e `y`.

União

`union(x, y)`: returna observações que estão em `x` ou `y`.

Diferença

`setdiff(x, y)`: retorna observações que estão em `x`, mas não em `y`.

Referência

- R for Data Science - https://r4ds.had.co.nz/