

(<https://databricks.com>)

## Etapa 2 - Criação do Schema estrela:

Para a criação do Schema estrela, o desafio será usar `R` e o `SparkR`. Como os dados originais estão salvos em *parquet*, a primeira leitura precisa ser realizada com o `SparkR` e depois transformada para o `R` para a criação do *schema*

### Leitura e separação inicial da base:

```
#bibliotecas
library(dplyr)
library(SparkR)
```

The following object is masked by\_ ‘.GlobalEnv’:

```
setLocalProperty
```

The following objects are masked from ‘package:dplyr’:

```
arrange, between, coalesce, collect, contains, count, cume_dist,
dense_rank, desc, distinct, explain, expr, filter, first, group_by,
intersect, lag, last, lead, mutate, n, n_distinct, ntile,
percent_rank, rename, row_number, sample_frac, select, slice, sql,
summarize, union
```

The following objects are masked from ‘package:stats’:

```
cov, filter, lag, na.omit, predict, sd, var, window
```

The following objects are masked from ‘package:base’:

```
as.data.frame, colnames, colnames<-, drop, endsWith, intersect,
rank, rbind, sample, startsWith, subset, summary, transform, union
```

```
# Leitura do parquet amazon
amazon <- read.df("/FileStore/tables/amazon.csv", source = "csv", header="true", inferSchema = "true")
amazon <- drop(amazon, c("review_content","about_product")) #necessario para trazer o DF para DF em R e não em SparkR
head(amazon)
```

	user_name	review_id
1	Manav,Adarsh gupta,Sundeep,S.Sayeed Ahmed,jaspreet singh,Khaja moin,Anand,S.ARUMUGAM	
2	ArdKn,Nirbhay kumar,Sagar Viswanathan,Asp,Placeholder,BharanI,sonia,Niam	
3	Kunal,Himanshu,viswanath,sai niharka,saqib malik,Aashiq,Ramu Challa,Sanjay gupta	
4	Omkar dhale,JD,HEMALATHA,Ajwadh a.,amar singh chouhan,Ravi Siddan,Himanshu Goel,Udaykumar	
5	rahuls6099,Swasat Borah,Ajay Wadke,Pranali,RVK,Bhargav,Durai Vignesh,Amazon Customer	
6	Jayesh,Rajesh k.,Soopy,amazon customer,Aman,Shankar,dinesh,Chitra,Ajaybabu.O.M	

	review_title
1	Satisfied,Charging is really fast,Value for money,Product review,Good quality,Go
od product,Good Product,As of now seems good	
2	A Good Braided Cable for Your Type C Device,Good quality product from ambrane,Super cable,As,Good quality,Good product,its good,Good qual
ity for the price but one issue with my unit	
3	Good speed for earlier versions,Good Product,Working good,Good for the price,Good,Worth for m

Obstáculo: A transformação do DF de sparkR para R, não contempla células grandes. Portanto, foi necessário retirar as colunas "review\_content" e "about\_product" para fazer essa transformação.

```
# O DF amazon realmente está em SparkDF em RSpark
class(amazon)
```

```
[1] "SparkDataFrame"
attr(,"package")
[1] "SparkR"
```

Analisando alguns *product\_id* da base (como o da célula a seguir), percebi um problema na transferência de `SparkR` para `R`: algumas linhas se misturam na própria leitura do DF, ficam "deslocadas" após o *product\_name*. neste caso, vemos que o product name "entra dentro" das categorias. e fica tudo bagunçado. Portanto, além da perda das duas colunas *review\_content* e *about\_product*, teremos perdas em algumas linhas (alguns produtos) em que isso acontece.

```
head(amazon[amazon$product_id=='B0BNVBjW2S',])
```

product_id	product_name						
1 B0BNVBjW2S	"boAt Newly Launched Wave Electra with 1.81"" HD Display						
	category	discounted_price					
1	Smart Calling Ultra-Seamless BT Calling Chip	20	Built-in Watch Faces				
	actual_price	discount_percentage					rating
1	100 + Sports Modes	Menu Personalization	in-Built Games(Cherry Blossom)"				
	rating_count	user_id	user_name	review_id			
1	Electronics WearableTechnology SmartWatches	₹7,990	69%	4.1			
	review_title						
1	154						
img_link							
1	AEYLB6L333GKGRGR5N6NDB335TQ,AEUZYVUGRR6URWHTQQR3NCGWN46A,AHYWG4RZCXWYBUPMUCNYX76JWF4Q,AHKCYSBVKKLZ6TZEUYSMS7JK703A,AHOLDR6WNL5GVEDVEX7HEK7KGA2A,AEVCDJRYLA3LTJCNTFYX53MAHAGA,AHM52LICMSWL734Q5OL4BUM7YWLA,AHFK5JSZGYMOMOE36LRSR2HC3V3Q						
	product_link						
1	App,Dr. Rinchin,Priyanka Devaraj,Yasinlove,Anand Patel,bhaskar,MUHAMMAD ALAU DIN,Mantu kumar Sahoo						

```
# transformação do DF Spark para R
amaz <- as.data.frame(amazon)
class(amaz)
```

```
[1] "data.frame"
```

Agora que o DF está no formato do R, podemos começar com o processo de construção do schema. Como visto na leitura inicial dos dados, as colunas *user\_id*, *user\_name*, *review\_id* e *review\_title* possuem 8 elementos em cada. Ou seja, cada produto tem 8 reviews escritos por 8 usuários diferentes. Aqui, o objetivo é criar essas bases separadas, para usuários, produtos e reviews.

```
#criando a df_users, em que separamos usuarios, reviews e categorias de dentro da base original
```

```
i=1
df_users = data.frame(user_id = strsplit(as.character(amaz$user_id[i]),',')[[1]],
                      user_name = strsplit(as.character(amaz$user_name[i]),',')[[1]],
                      review_id = strsplit(as.character(amaz$review_id[i]),',')[[1]],
                      review_title = strsplit(as.character(amaz$review_title[i]),',')[[1]],
                      product_id = amaz$product_id[i],
                      category = strsplit(as.character(amaz$category[i]), '\\|')[[1]][1])

for(i in 2:nrow(amaz)){
  tryCatch({
    dfc = data.frame(user_id = strsplit(as.character(amaz$user_id[i]),',')[[1]],
                      user_name = strsplit(as.character(amaz$user_name[i]),',')[[1]],
                      review_id = strsplit(as.character(amaz$review_id[i]),',')[[1]],
                      review_title = strsplit( gsub("(, [a-zA-Z]+)|(-)|([0-9])|(, à)", "~\\1", as.character(amaz$review_title[i])), "~" )[[1]],
                      product_id = amaz$product_id[i],
                      category = strsplit(as.character(amaz$category[i]), '\\|')[[1]][1])

    # print(i)
    df_users = rbind(df_users,dfc)
  },error=function(e){cat("ERROR :",conditionMessage(e),i, "\n")})
}
```

```
ERROR : arguments imply differing number of rows: 8, 7, 1 1259
ERROR : arguments imply differing number of rows: 8, 9, 1 1261
ERROR : arguments imply differing number of rows: 8, 7, 1 1262
ERROR : arguments imply differing number of rows: 8, 9, 1 1279
ERROR : arguments imply differing number of rows: 8, 9, 1 1281
ERROR : arguments imply differing number of rows: 8, 9, 1 1295
ERROR : arguments imply differing number of rows: 8, 7, 1 1308
ERROR : arguments imply differing number of rows: 8, 9, 1 1328
ERROR : arguments imply differing number of rows: 8, 7, 1 1332
ERROR : arguments imply differing number of rows: 8, 7, 1 1340
ERROR : arguments imply differing number of rows: 8, 7, 1 1349
ERROR : arguments imply differing number of rows: 8, 9, 1 1351
ERROR : arguments imply differing number of rows: 8, 7, 1 1372
ERROR : arguments imply differing number of rows: 8, 7, 1 1380
ERROR : arguments imply differing number of rows: 8, 7, 1 1382
ERROR : arguments imply differing number of rows: 8, 7, 1 1383
ERROR : arguments imply differing number of rows: 8, 7, 1 1393
ERROR : arguments imply differing number of rows: 8, 9, 1 1418
ERROR : arguments imply differing number of rows: 8, 7, 1 1434
ERROR : arguments imply differing number of rows: 8, 9, 1 1440
ERROR : arguments imply differing number of rows: 8, 7, 1 1465
```

```
df_users <- df_users %>% dplyr::distinct()
```

obstáculo: Para a separação do *review\_title*, é bem difícil definir aonde fica a separação dos textos. Alguns separam facilmente por vírgulas. Porém, quando algum título de produto único possui vírgulas, essa separação não fica clara. O comando em REGEX, auxilia nessa separação e cria algumas exceções. O comando `tryCatch` vai executar o looping e pular, quando ocorrer o erro. Portanto, aqui existe mais perda de informação devido a esta extração.

## Criando as Tabelas FATO, PRODUCTS, USERS e REVIEWS

Observando alguns displays, foi possível observar que alguns produtos, com `product_id` diferentes, possuem os mesmos reviews e usuários. (Uma comparação de exemplo pode ser vista nas imagens 2 e 3 no readme)

Neste caso, vamos criar um ranking dessas observações e manter apenas uma delas. de forma aleatória.

```
# checando os produtos iguais, mas product_id diferentes. Neste caso, o review_id e o user_id são os mesmos, porem o product_id é diferente.
prod_diff <- df_users %>%
dplyr::group_by(user_id,user_name,review_id) %>%
dplyr::mutate(id_order = rank(review_id,ties.method= "random"))
```

```
#podemos ver claramente que o produto é essencialmente o mesmo.
display(prod_diff[prod_diff$review_id=="RGIQEG07R9HS2",])
```

Table

	A <sup>B</sup> <sub>C</sub> user_id	A <sup>B</sup> <sub>C</sub> user_name	A <sup>B</sup> <sub>C</sub> review_id	A <sup>B</sup> <sub>C</sub> review_title	A <sup>B</sup> <sub>C</sub> product_id	A <sup>B</sup> <sub>C</sub> category	1 <sup>2</sup> <sub>3</sub> id_order
1	AECPFYFQVRUWC3KGNLJIOREFP5...	ArdKn	RGIQEG07R9HS2	A Good Braided Cable for Your Type C Device	B098NS6PVG	Computers&Accessori...	
2	AECPFYFQVRUWC3KGNLJIOREFP5...	ArdKn	RGIQEG07R9HS2	A Good Braided Cable for Your Type C Device	B082LZGK39	Computers&Accessori...	
3	AECPFYFQVRUWC3KGNLJIOREFP5...	ArdKn	RGIQEG07R9HS2	A Good Braided Cable for Your Type C Device	B082LSVT4B	Computers&Accessori...	

3 rows

```
prod_diff2 <- prod_diff %>% dplyr::filter(id_order==1) %>% dplyr::select(-c(id_order))
# confirmando se temos agora produtos unicos com usuarios e reviews unicos:
n_before <- nrow(prod_diff2)
n_after <- nrow(prod_diff2 %>% dplyr::group_by(user_id,review_id,product_id) %>% dplyr::count())

df_users2 <- prod_diff2
print(paste(n_before,n_after)) # existe ainda uma pequena duplicação de 11 observações. Mas está suficientemente aceitavel
```

[1] "8125 8116"

```
# criando as tabelas do schema
fato = df_users2[,c('user_id','review_id','product_id')]
users = df_users2[,c('user_id','user_name')]
reviews = df_users2[,c('review_id','review_title')]
products0 = amaz[(names(amaz) %in% c("product_id", "product_name", "discounted_price", "actual_price","discount_percentage", "rating","rating_count"))]
```

```
categories <- df_users2 %>% dplyr::group_by(product_id,category) %>% dplyr::count() %>% dplyr::select(-c(n))
```

```
products <- merge(products0,categories,by="product_id")
```

```
products <- products %>% dplyr::filter(product_id %in% df_users2$product_id)
```

```
fato <- dplyr::distinct(fato)
users <- dplyr::distinct(users)
reviews <- dplyr::distinct(reviews)
products <- dplyr::distinct(products)
```

```
products$actual_price <- gsub('₹','',products$actual_price)
products$discounted_price <- gsub('₹','',products$discounted_price)
products$rating_count <- as.numeric(gsub(',', '',products$rating_count))
```




```
#Rupias Indianas Para Reais: (valor em 01/07/2024), imagem no github
products$actual_price <-round(as.numeric(gsub(",","",products$actual_price))*0.067,2)
products$discounted_price <-round(as.numeric(gsub(",","",products$discounted_price))*0.067,2)
```

Warning in eval(parse(text = DATABRICKS\_CURRENT\_TEMP\_CMD\_\_)) :  
NAs introduced by coercion

```
Warning in eval(parse(text = DATABRICKS_CURRENT_TEMP_CMD__)) :
  NAs introduced by coercion
Warning in eval(parse(text = DATABRICKS_CURRENT_TEMP_CMD__)) :
  NAs introduced by coercion
```

Aqui, por conta de alguma "bagunça" que ocorre na transformação do CSV em *PARQUET*, algumas linhas de produtos perderam os seus valores, por isso o Warning acima ocorreu. A decisão é de seguir com a base desta mesma forma, com mais esta perda de informação. No comando à seguir é possível ver um exemplo de como ficou esta bagunça, na leitura do parquet original 'amazon'

display(amazon[amazon\$product\_id=='B0BNVBJW2S',])

Table									
	product_id	product_name	category	discounted_price	actual_price	discount			
1	B0BNVBJW2S	"boAt Newly Launched Wave Electra with 1.81"" HD Display	Smart Calling Ultra-Seamless BT Calling Chip	20 Built-in Watch Faces	100 + Sports Modes	Menu Pers			

1 row

Resultado das bases do schema estrela:

```
head(fato)

# A tibble: 6 x 3
# Groups:   user_id, review_id [6]
  user_id      review_id      product_id
  <chr>      <chr>      <chr>
1 AHCTC6ULH4XB6YHDY6PCH2R772LQ R6AQJGUP6P86 B07JW9H4J1
2 AGYHHIERNXA6P5T7CZLXKVPT7IQ R1KD19VHEDV00R B07JW9H4J1
3 AFPHD2CRPDZMWMBL7WXRSVYWS5JA RYGG50M09S3KY B098NS6PVG
4 AGU3BBQ2V2DDAMOAKGFAWDDQ6QHA R3J3EQQ9TZISZJ B096MSW6CT
5 AESFLDV2PT363T2AQLWQOWZ4N3OA R3E7WBGK7ID0KV B096MSW6CT
6 AHTPQRIMGUD4BYR5YIHBH3CCGEFQ RWU79XKQ6I1QF B096MSW6CT
```

```
head(users)

# A tibble: 6 x 2
# Groups:   user_id, user_name [6]
  user_id      user_name
  <chr>      <chr>
1 AHCTC6ULH4XB6YHDY6PCH2R772LQ Sundeep
2 AGYHHIERNXA6P5T7CZLXKVPT7IQ S.Sayeed Ahmed
3 AFPHD2CRPDZMWMBL7WXRSVYWS5JA Asp
4 AGU3BBQ2V2DDAMOAKGFAWDDQ6QHA Kunal
5 AESFLDV2PT363T2AQLWQOWZ4N3OA Himanshu
6 AHTPQRIMGUD4BYR5YIHBH3CCGEFQ viswanath
```

```
head(reviews)

# A tibble: 6 x 2
# Groups:   review_id [6]
  review_id      review_title
  <chr>      <chr>
1 R6AQJGUP6P86 Value for money
2 R1KD19VHEDV00R Product review
3 RYGG50M09S3KY ,As
4 R3J3EQQ9TZISZJ Good speed for earlier versions
5 R3E7WBGK7ID0KV ,Good Product
6 RWU79XKQ6I1QF ,Working good
```

```
head(products)

product_name
```

Flash Drive

BlueRigger Digital Optical Audio Toslink Cable (3.3 Feet / 1 Meter) With 8 Channel (7.1) Audio Support (for Home Theatre, Xbox, Playstation etc.)

	discounted_price	actual_price	discount_percentage	rating	rating_count
1	33.97	80.94	58%	4.1	8131
2	50.18	89.71	44%	4.2	179692
3	26.73	33.43	20%	4.3	27201
4	18.69	25.12	26%	4.3	31534
5	19.36	43.55	56%	4.3	253105
6	27.87	40.13	31%	4.2	30023

## Escrevendo as bases em parquet no databricks

```
# transformando de R para RSpark
fato.spark <- as.DataFrame(fato)
users.spark <- as.DataFrame(users)
reviews.spark <- as.DataFrame(reviews)
products.spark <- as.DataFrame(products)
```

```
write.parquet(fato.spark, "amazon_star_schema/fato.parquet", "overwrite")
write.parquet(users.spark, "amazon_star_schema/users.parquet", "overwrite")
write.parquet(reviews.spark, "amazon_star_schema/reviews.parquet", "overwrite")
write.parquet(products.spark, "amazon_star_schema/products.parquet", "overwrite")
#
```

Testando a leitura:

```
fato <- read.df("/fato.parquet", "parquet")
```

	user_id	review_id	product_id
1	AG3D604STAQKAY2UVGEUV46KN35Q	R3HXWT0LRP0NMF	B07JW9H4J1
2	AHMY5CWJMMK5BJRBBSNLYT3ONILA	R2AJM3LFTLZHFO	B07JW9H4J1
3	AHCTC6ULH4XB6YHDY6PCH2R772LQ	R6AQJGUP6P86	B07JW9H4J1
4	AGYHHIERNXKA6P5T7CZLXKVP7TIQ	R1KD19VHEDV00R	B07JW9H4J1
5	AG40G0FWXJZTQ2HKYIOCOY3KXF2Q	R3C02RMQMK6FC	B07JW9H4J1
6	AENGUS23SXMOS7JPDTW52PNNVWGQ	R39GQRVBUZBWGY	B07JW9H4J1