Pontifícia Universidade Católica do Rio de Janeiro

Pós-Graduação em Ciência de Dados e Analytics

Sprint de Engenharia de Dados

Tomás Cavalcante Valadão

MVP - Criação de um Pipeline de Dados utilizando o Databricks para fazer o processo ETL e algumas análises do catálogo de produtos da plataforma Steam

Projeto disponível em https://github.com/TomasValadao/DataEngineering-PUC-Rio/

Rio de Janeiro,
Julho de 2024

Resumo

Este projeto foi criado para aplicar os conhecimentos adquiridos no módulo de Engenharia de Dados do curso de Especialização em Ciência de Dados e Analytics da PUC-Rio. O objetivo é desenvolver um MVP (produto mínimo viável) de um pipeline de dados, abrangendo as etapas de busca, coleta, modelagem, carga e análise de dados, utilizando as tecnologias da plataforma Databricks.

Os dados para este projeto foram obtidos a partir da pesquisa "Steam Games Dataset 2024", que contém informações relevantes sobre aproximadamente 83.000 títulos na plataforma Steam. O estudo inicial foi realizado por Artemiy Ermilov, Arina Nevolina, Assol Kubaeva e Артем Поспелов, e foi publicado na plataforma Kaggle, estando disponível para consulta pública em Kaggle.

Objetivo

O objetivo desse projeto é entender algumas características dos produtos contidos na maior plataforma de distribuição de jogos online do mundo, a Steam. O intuito desse projeto é realizar um processo de ponta a ponta que vai desde a coleta até a análise das informações dos jogos contidos na plataforma para que se tenha uma ideia do que o público gosta e se quantidade é qualidade.

Esse objetivo pretende ser atingido com as seguintes perguntas:

1. Qual o preço médio por ano dos jogos na Steam?
2. Quais são os 5 jogos com mais DLC (Downloadable Content)?
3. Qual é o jogo com o maior pico de jogadores na Steam?
4. Qual ano teve a maior quantidade de lançamentos de jogos na Steam?
5. Qual jogo tem o maior número de conquistas?

Desenvolvimento do Projeto

1. Coleta de Dados

A coleta dos dados necessários se deu através da plataforma Kaggle, onde Artemiy Ermilov, Arina Nevolina, Assol Kubaeva e Артем Поспелов compilaram os dados que eles extraíram da plataforma em maio de 2024 referente a aproximadamente 83.000 títulos. Essa base de dados então foi disponibilizada na plataforma para que outras pessoas pudessem extrair algum tipo de informação desses dados, como é o caso desse projeto.

A coleta desses dados se deu de forma manual devido a necessidade de ter uma API Key para automatizar a coleta desses dados. Desse modo, a API Key não conseguiria ser armazenada de forma segura nesse projeto, fazendo com que essas credenciais tivessem sujeitas a exploit e eventualmente a geração de problemas ao dono.

2. Carga

Após a coleta de dados dentro da plataforma Kaggle e a exportação manual desses dados, a opção de armazenamento dessa base foi persistida dentro do Databricks File System, DBFS, para que o projeto pudesse fazer uso da arquitetura medallion. No entanto, antes de que essa modelagem fosse possível, o upload das informações no DBFS foi feita através da UI do Databricks como representada nas imagens a seguir:

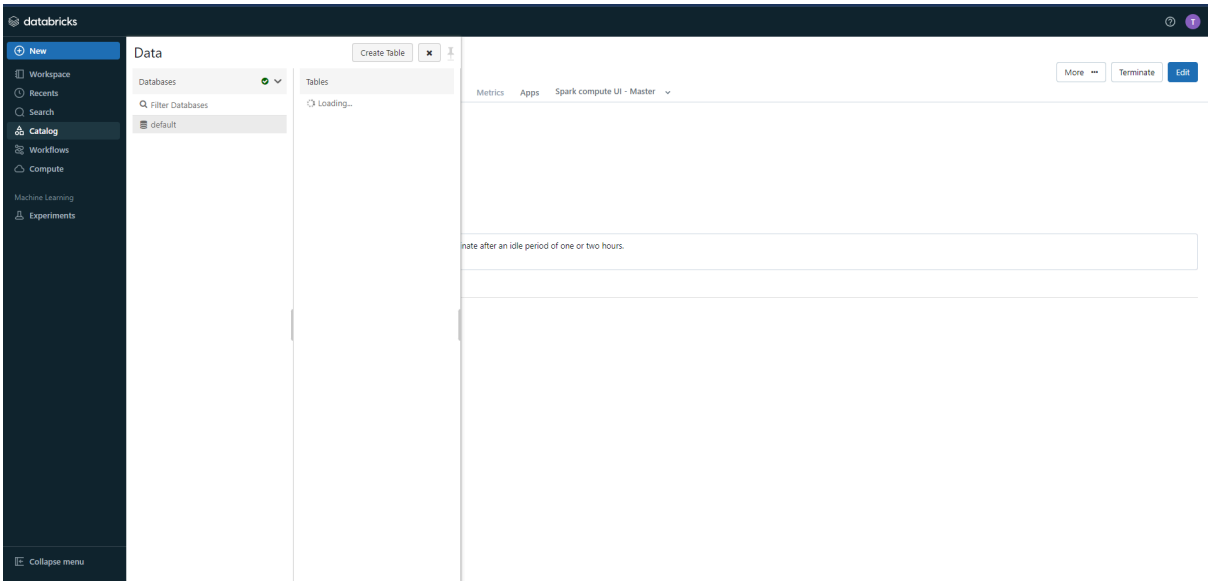


Imagem 1: Identificação do Catálogo do Databricks.

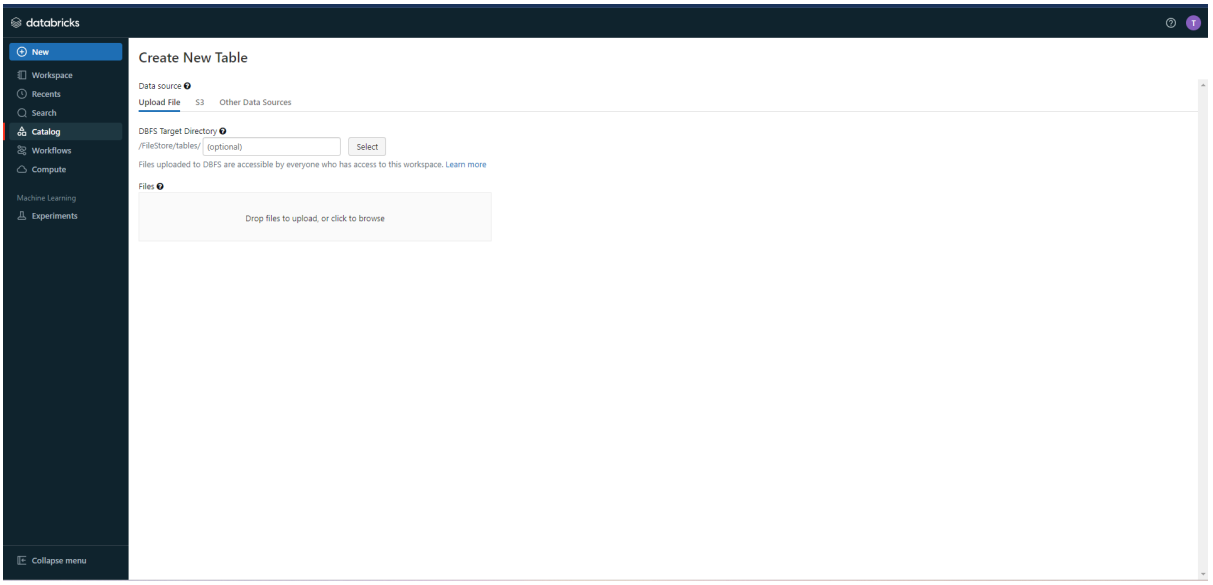


Imagem 2: Interface de upload de arquivo pro DBFS.

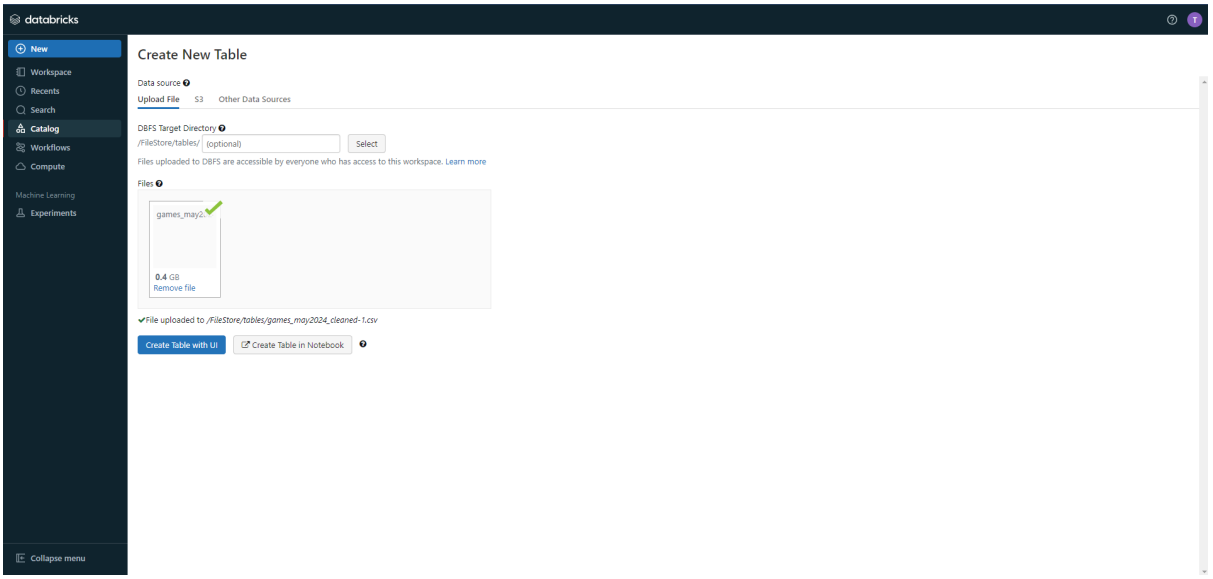


Imagem 3: Upload do csv extraído para o DBFS.

Como próximo passo desse pipeline de dados, o processo necessitava de uma geração dos dados, pois neste momento não havíamos chegado na arquitetura medallion. Seguindo a metodologia da arquitetura medallion, era necessário seguir o modelo bronze, prata e ouro.

A camada bronze é a camada onde os dados crus são obtidos e armazenados sem qualquer limpeza neles. Seguindo essa definição, essa parte da arquitetura ainda não foi entregue. Para isso, é necessário que esse dado seja transformado num arquivo Parquet e armazenado em um DataTable utilizando o PySpark. A próxima imagem mostrará como esse passo foi feito e como o cluster foi ligado. As duas fases faltantes vão ser mencionadas ao longo deste relatório.

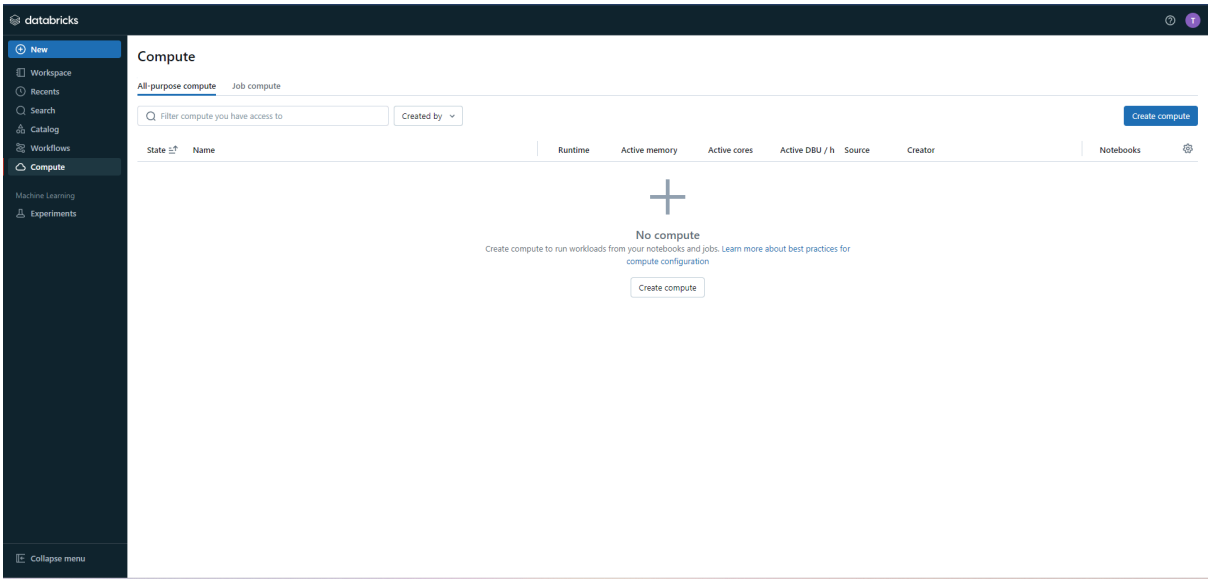


Imagem 4: Interface da aba Compute no Databricks

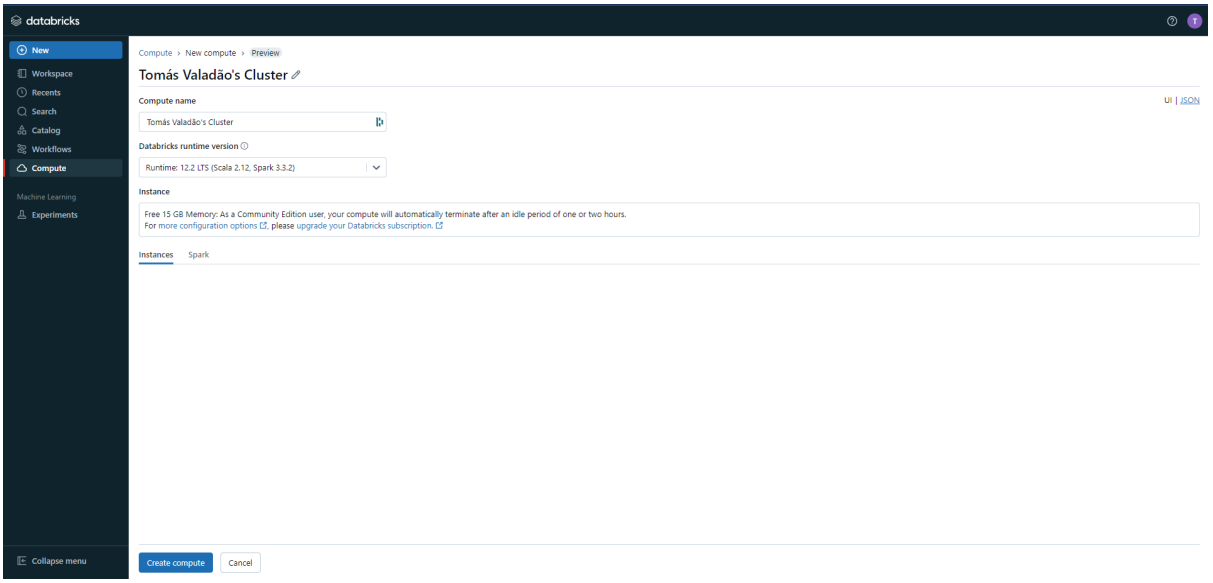


Imagem 5: Criação do Cluster utilizado para a criação do pipeline de dados.

Steam Bronze Layer

Esse notebook vai persistir os dados que foram inseridos via o DBFS na área do Catalog. Após a persistencia dos dados crus, será feita um próximo passo no qual haverá uma análise da corretude e filtração dos dados para que eles possam atingir o objetivo final. O código das células a seguir está em python.

```
# Localização do arquivo
file_location = "/FileStore/tables/games_may2024_cleaned-1.csv"
file_type = "csv"

# Opções do CSV
infer_schema = "false"
first_row_is_header = "true"
delimiter = ","

# Criação do DataFrame utilizando o PySpark
df = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)
```

(1) Spark Jobs

Imagem 6: Criação de um DataFrame Spark

```
%py

# Configuração para habilitar o Overwrite quando o Cluster for desligado
spark.conf.set("spark.sql.legacy.allowCreatingManagedTableUsingNonemptyLocation", "true")
```

```
# Criação da tabela persistida usando o CREATE OR ALTER do PySpark

permanent_table_name = "steam_bronze_layer_table"

df.write.mode("overwrite").format("parquet").saveAsTable(permanent_table_name)
```

(1) Spark Jobs

Imagem 7: Persistência da Tabela na camada bronze.

```
%sql

/* Checagem para ver se os dados inseridos podem ser acessados usando o SparkSQL. */

select count(*) from `steam_bronze_layer_table`
```

(2) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [count(1): long]

Table

	count(1)
1	83646

1 row | 1.40 seconds runtime — Refreshed 3 days ago

SQL cell result stored as PySpark data frame `_sqldf`. Learn more

Imagem 8: Garantido que os dados são acessíveis via SparkSQL.

3. Limpeza de Dados

Tendo passado do passo de carga dos dados e tendo a camada bronze preparada, agora é necessário garantir a qualidade dos dados para que os mesmos não atrapalhem a análise que visa atingir o objetivo desse projeto. Para que isso seja possível, é necessário adentrar a camada prata e identificar todos os potenciais problemas nos dados que podem vir a afetar a análise e eventualmente direcionar o entendimento numa direção incorreta. Nessa camada será feita toda a checagem de qualidade de dados e a persistência final dos dados.

Para que a limpeza seja sempre efetiva quando novos dados forem incluídos, é necessário que exista um workflow para que as camadas executem em ordem. No entanto, na versão do Databricks Community, esse workflow não está habilitado, então foi feito um contorno para que a análise seja sempre feita após a limpeza e a limpeza sempre após a extração.

Durante a limpeza de dados foi constatada algumas colunas desnecessárias que só consumiam espaço e outras que estavam com valores incorretos. Desse modo, tudo que pudessem interferir foi removido, além do catálogo de dados ao final, como apresentado nas imagens a seguir:

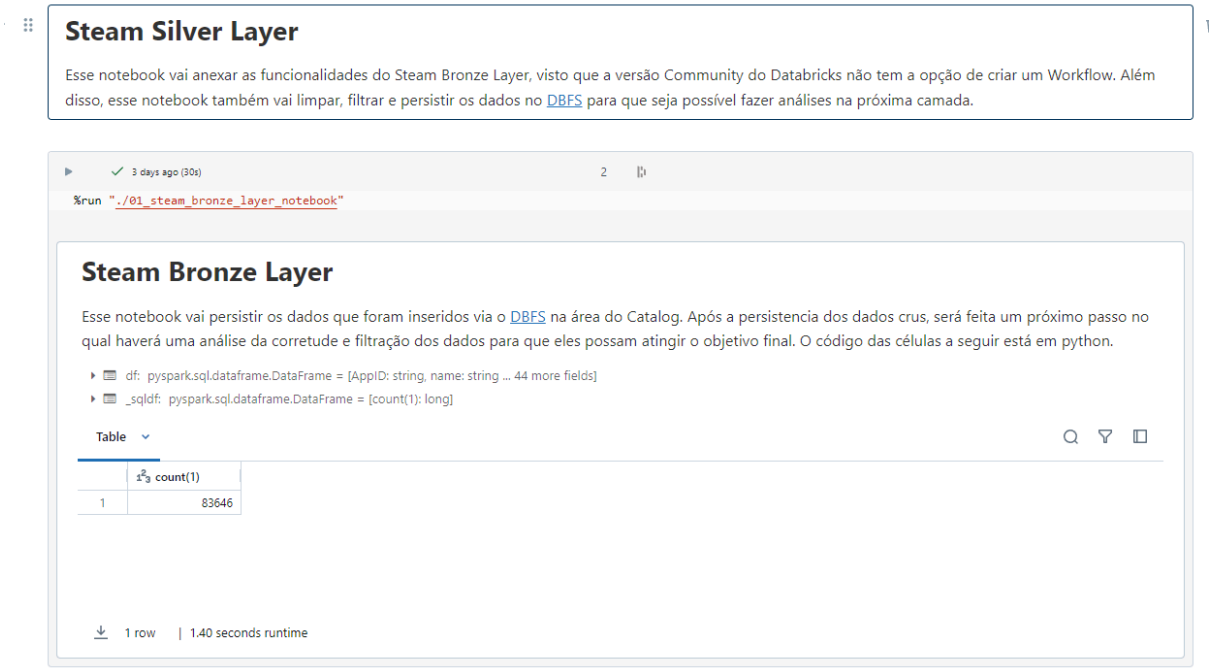
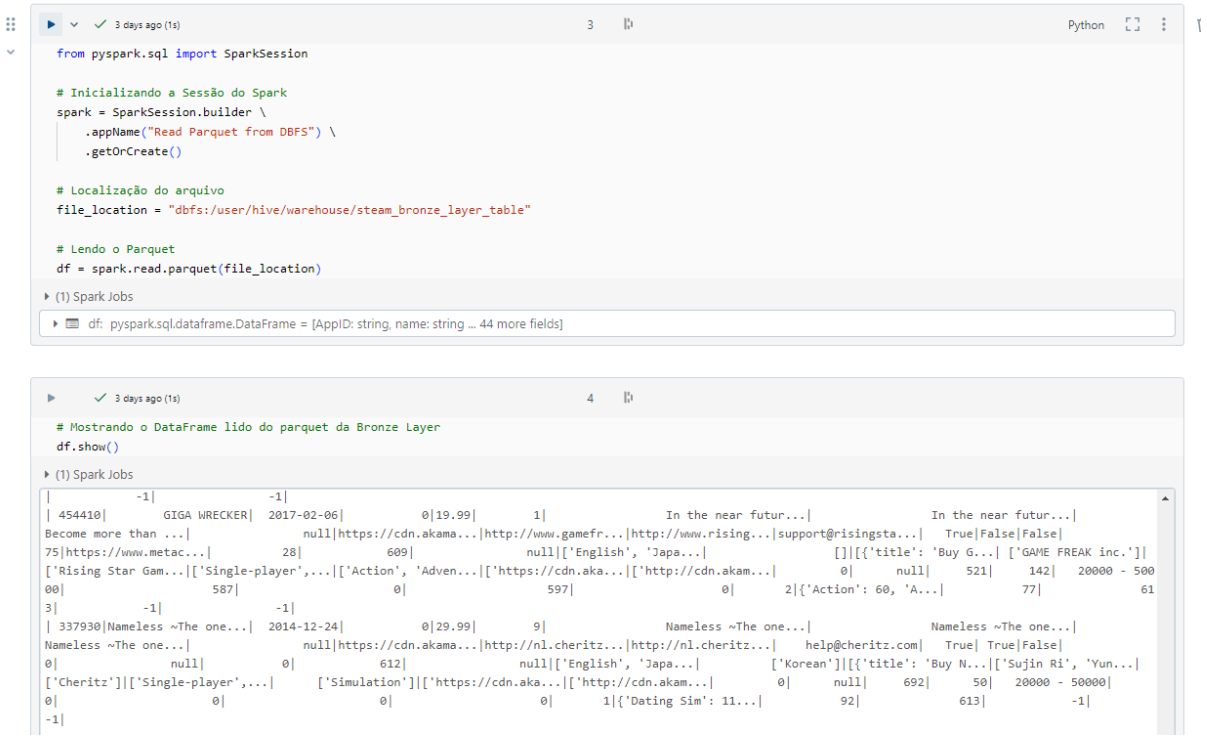


Imagem 9: Garantia de conexão entre a camada bronze e a prata.



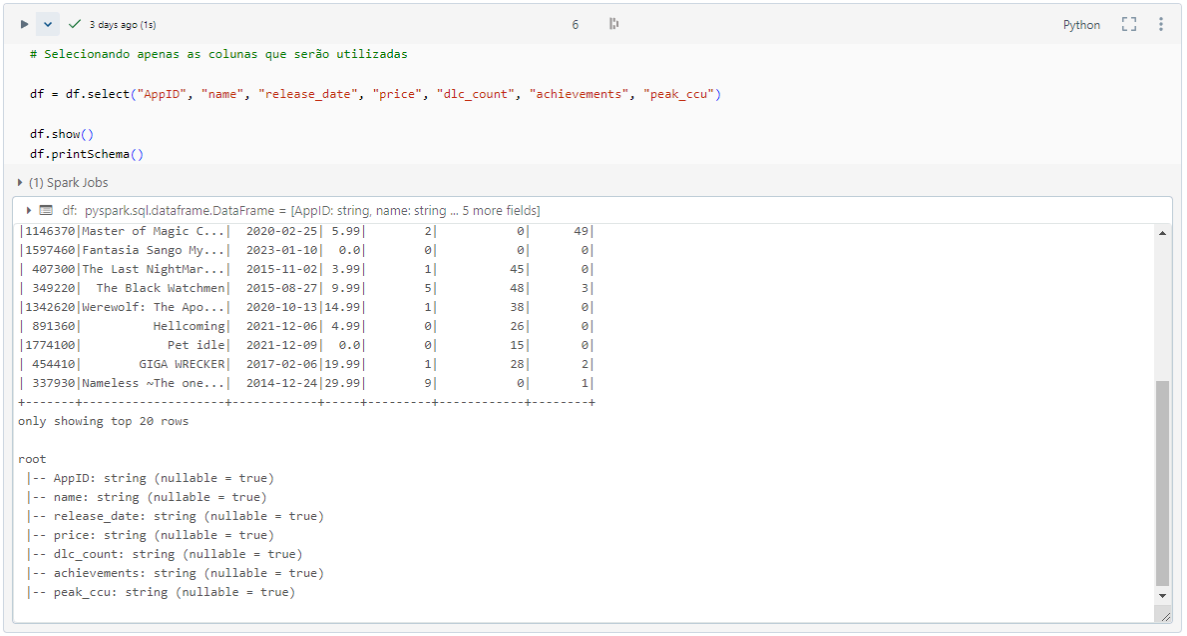
```
3 days ago (1s)  3  Python

from pyspark.sql import SparkSession

# Inicializando a Sessão do Spark
spark = SparkSession.builder \
    .appName("Read Parquet from DBFS") \
    .getOrCreate()

# Localização do arquivo
file_location = "dbfs:/user/hive/warehouse/steam_bronze_layer_table"

# Lendo o Parquet
df = spark.read.parquet(file_location)

(1) Spark Jobs
df: pyspark.sql.dataframe.DataFrame = [AppID: string, name: string ... 44 more fields]
```

```
3 days ago (1s)  4

# Mostrando o DataFrame lido do parquet da Bronze Layer
df.show()

(1) Spark Jobs
|           -1|               -1|
| 454410|       GIGA WRECKER| 2017-02-06|          0|19.99|       1|               In the near futur...|               In the near futur...|
Become more than ...|               null|https://cdn.akama...|http://www.gamefr...|http://www.rising...|support@risingsta...|   True|False|False|
75|https://www.metac...|          28|          609|               null|['English', 'Japa...|                []|[{'title': 'Buy G...| ['GAME FREAK inc.']|
['Rising Star Gam...|['Single-player',...|['Action', 'Adven...|['https://cdn.aka...|['http://cdn.akam...|        0|      null|     521|     142|   20000 - 500
00|             587|                  0|                597|                  0|       2|{'Action': 60, 'A...|          77|              61
3|           -1|               -1|
| 337930|Nameless ~The one...| 2014-12-24|          0|29.99|       9|               Nameless ~The one...|               Nameless ~The one...|
Nameless ~The one...|               null|https://cdn.akama...|http://nl.cheritz...|http://nl.cheritz...|    help@cheritz.com|   True| True|False|
0|              null|          0|          612|               null|['English', 'Japa...|         ['Korean']|[{'title': 'Buy N...|['Sujin Ri', 'Yun...|
['Cheritz']|['Single-player',...|      ['Simulation']|['https://cdn.aka...|['http://cdn.akam...|        0|      null|     692|      50|   20000 - 50000|
0|                    0|                  0|                  0|       1|{'Dating Sim': 11...|          92|             613|           -1|
-1|
```

Imagem 10: Acesso a camada bronze via PySpark

```
3 days ago (<1s)  5  Python

# Print the schema
df.printSchema()

|-- publishers: string (nullable = true)
|-- categories: string (nullable = true)
|-- genres: string (nullable = true)
|-- screenshots: string (nullable = true)
|-- movies: string (nullable = true)
|-- user_score: string (nullable = true)
|-- score_rank: string (nullable = true)
|-- positive: string (nullable = true)
|-- negative: string (nullable = true)
|-- estimated_owners: string (nullable = true)
|-- average_playtime_forever: string (nullable = true)
|-- average_playtime_2weeks: string (nullable = true)
|-- median_playtime_forever: string (nullable = true)
|-- median_playtime_2weeks: string (nullable = true)
|-- peak_ccu: string (nullable = true)
|-- tags: string (nullable = true)
|-- pct_pos_total: string (nullable = true)
|-- num_reviews_total: string (nullable = true)
|-- pct_pos_recent: string (nullable = true)
|-- num_reviews_recent: string (nullable = true)
```

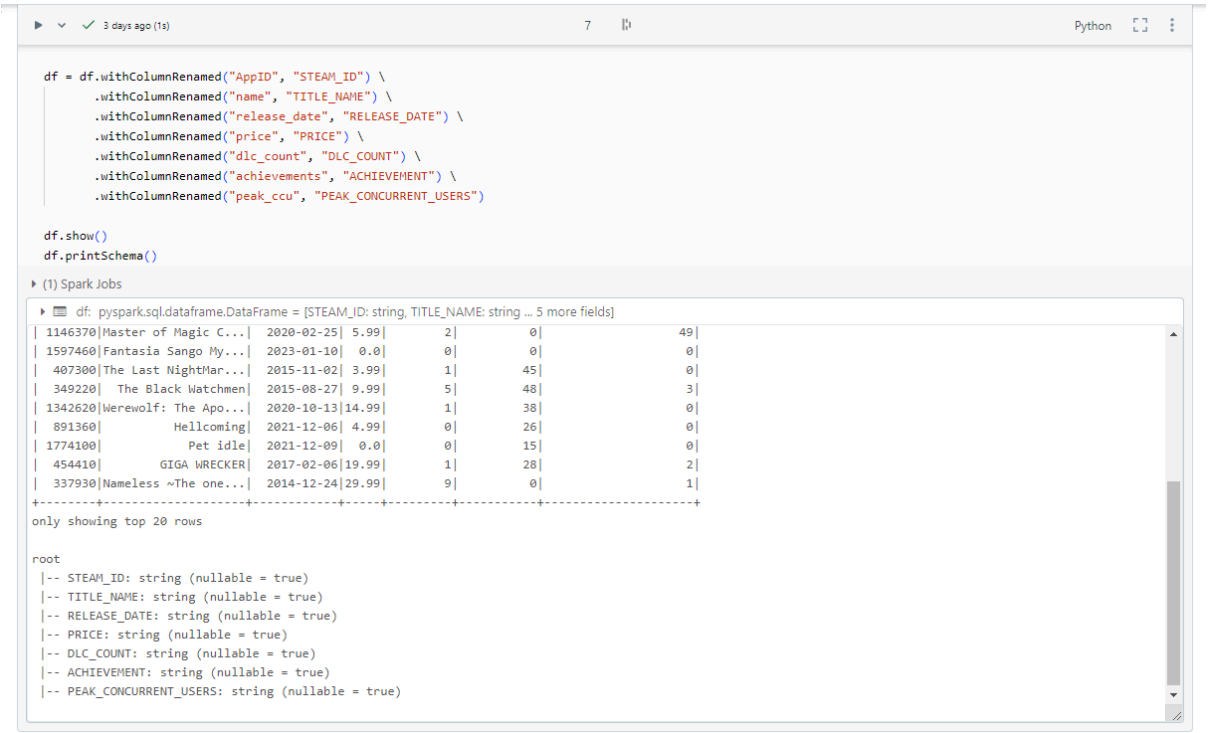
Imagem 11: Identificação do esquema de dados na camada bronze.



```
3 days ago (1s)  6  Python

# Selecionando apenas as colunas que serão utilizadas

df = df.select("AppID", "name", "release_date", "price", "dlc_count", "achievements", "peak_ccu")

df.show()
df.printSchema()
```

```
(1) Spark Jobs
df: pyspark.sql.dataframe.DataFrame = [AppID: string, name: string ... 5 more fields]
|1146370|Master of Magic C...|  2020-02-25| 5.99|        2|           0|      49|
|1597460|Fantasia Sango My...|  2023-01-10|  0.0|        0|           0|       0|
| 407300|The Last NightMar...|  2015-11-02| 3.99|        1|          45|       0|
| 349220|  The Black Watchmen|  2015-08-27| 9.99|        5|          48|       3|
|1342620|Werewolf: The Apo...|  2020-10-13|14.99|        1|          38|       0|
| 891360|          Hellcoming|  2021-12-06| 4.99|        0|          26|       0|
|1774100|            Pet idle|  2021-12-09|  0.0|        0|          15|       0|
| 454410|        GIGA WRECKER|  2017-02-06|19.99|        1|          28|       2|
| 337930|Nameless ~The one...|  2014-12-24|29.99|        9|           0|       1|
+-------+--------------------+------------+-----+---------+------------+--------+
only showing top 20 rows

root
 |-- AppID: string (nullable = true)
 |-- name: string (nullable = true)
 |-- release_date: string (nullable = true)
 |-- price: string (nullable = true)
 |-- dlc_count: string (nullable = true)
 |-- achievements: string (nullable = true)
 |-- peak_ccu: string (nullable = true)
```

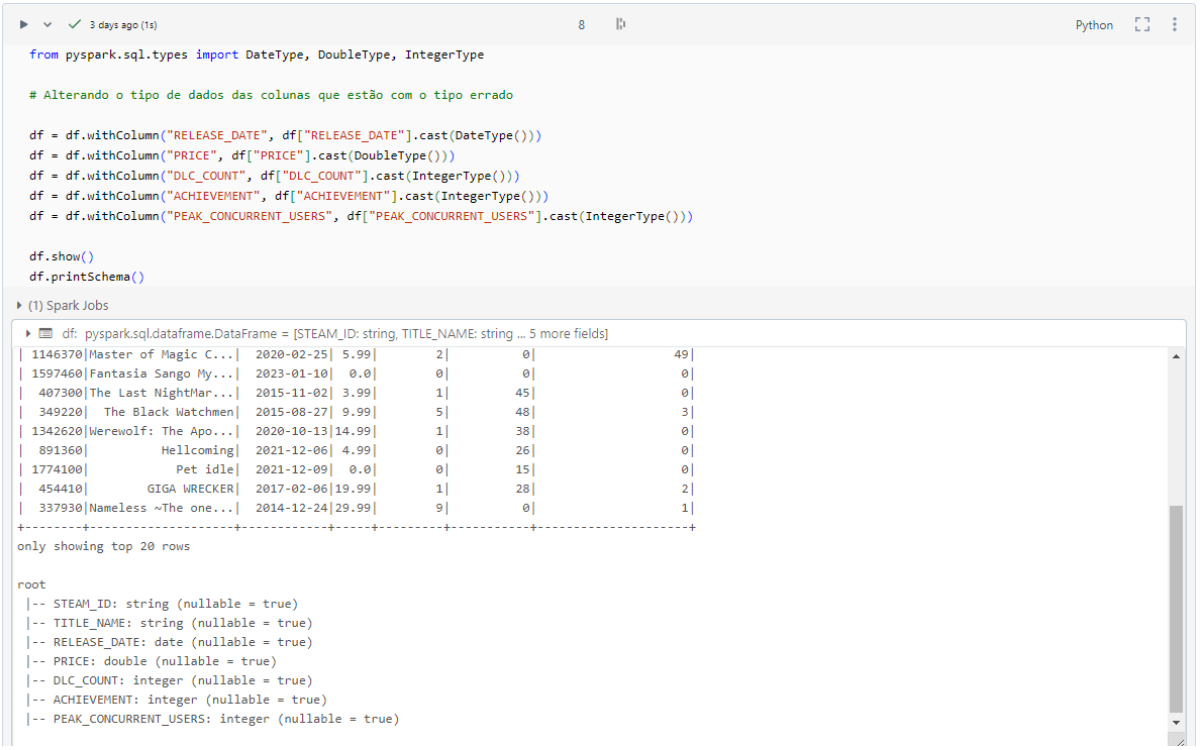
Imagem 12: Remoção das colunas que são desnecessárias para a camada ouro.



```
3 days ago (1s)  7  Python

df = df.withColumnRenamed("AppID", "STEAM_ID") \
       .withColumnRenamed("name", "TITLE_NAME") \
       .withColumnRenamed("release_date", "RELEASE_DATE") \
       .withColumnRenamed("price", "PRICE") \
       .withColumnRenamed("dlc_count", "DLC_COUNT") \
       .withColumnRenamed("achievements", "ACHIEVEMENT") \
       .withColumnRenamed("peak_ccu", "PEAK_CONCURRENT_USERS")

df.show()
df.printSchema()
```

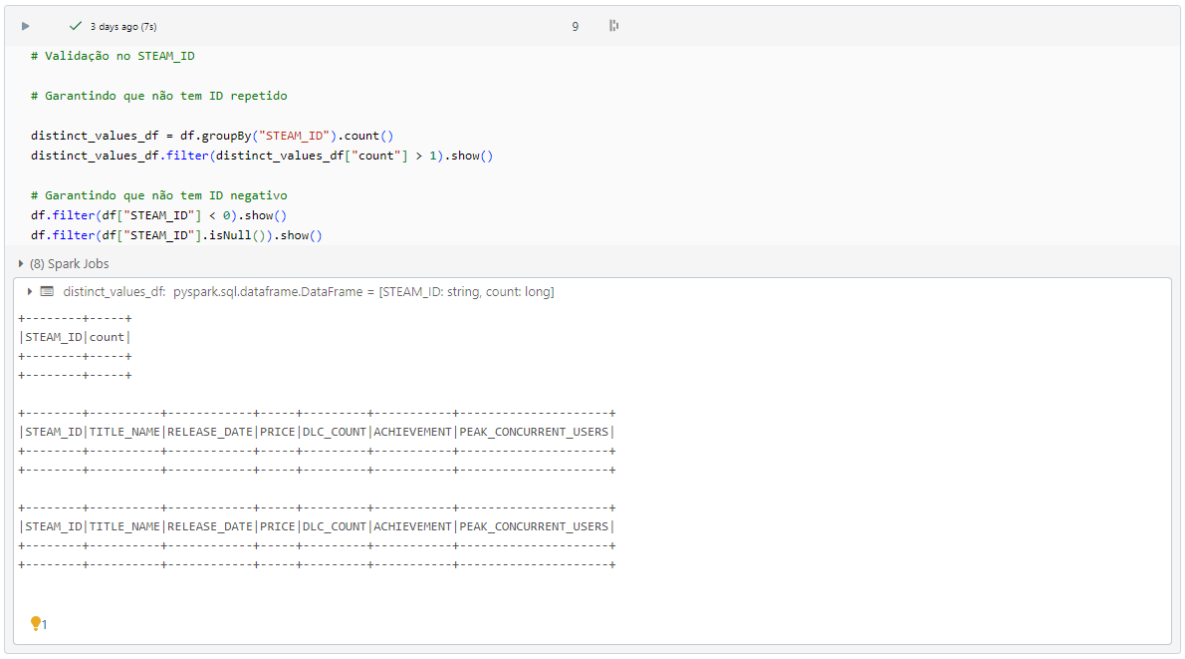
```
(1) Spark Jobs
df: pyspark.sql.dataframe.DataFrame = [STEAM_ID: string, TITLE_NAME: string ... 5 more fields]
| 1146370|Master of Magic C...|  2020-02-25| 5.99|        2|           0|                  49|
| 1597460|Fantasia Sango My...|  2023-01-10|  0.0|        0|           0|                   0|
|  407300|The Last NightMar...|  2015-11-02| 3.99|        1|          45|                   0|
|  349220|  The Black Watchmen|  2015-08-27| 9.99|        5|          48|                   3|
| 1342620|Werewolf: The Apo...|  2020-10-13|14.99|        1|          38|                   0|
|  891360|          Hellcoming|  2021-12-06| 4.99|        0|          26|                   0|
| 1774100|            Pet idle|  2021-12-09|  0.0|        0|          15|                   0|
|  454410|        GIGA WRECKER|  2017-02-06|19.99|        1|          28|                   2|
|  337930|Nameless ~The one...|  2014-12-24|29.99|        9|           0|                   1|
+--------+--------------------+------------+-----+---------+------------+--------------------+
only showing top 20 rows

root
 |-- STEAM_ID: string (nullable = true)
 |-- TITLE_NAME: string (nullable = true)
 |-- RELEASE_DATE: string (nullable = true)
 |-- PRICE: string (nullable = true)
 |-- DLC_COUNT: string (nullable = true)
 |-- ACHIEVEMENT: string (nullable = true)
 |-- PEAK_CONCURRENT_USERS: string (nullable = true)
```

Imagem 13: Padronização das colunas.



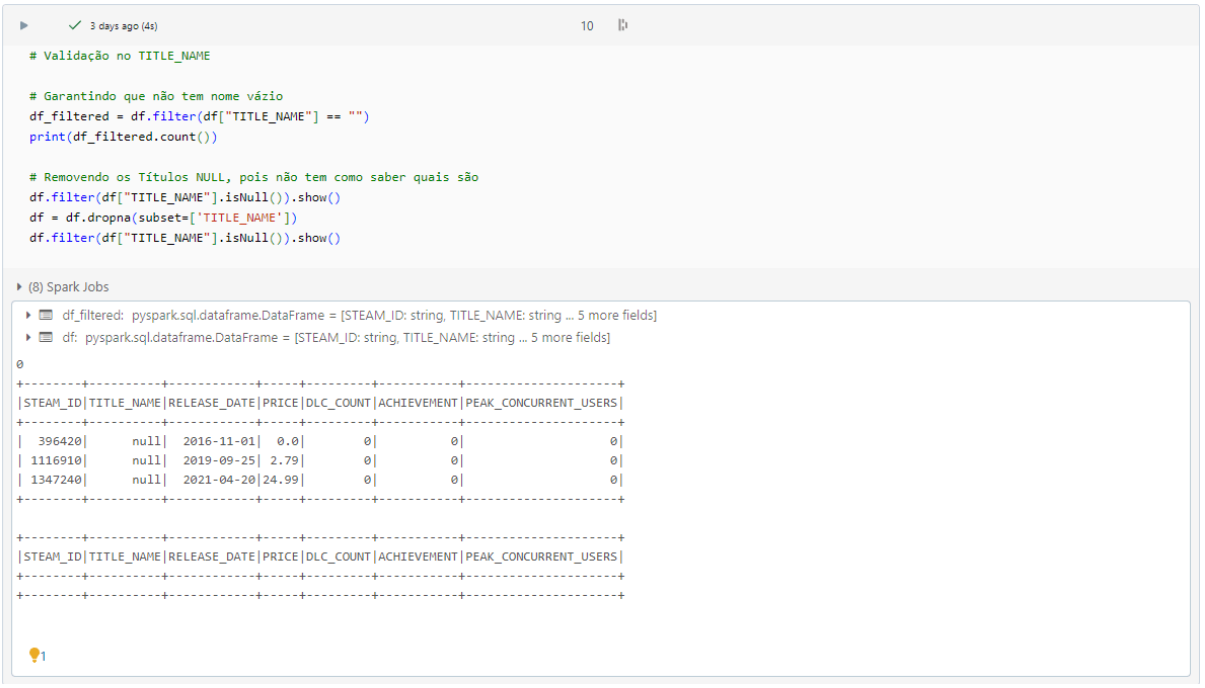
```
3 days ago (1s)                                                    8                                                    Python

from pyspark.sql.types import DateType, DoubleType, IntegerType

# Alterando o tipo de dados das colunas que estão com o tipo errado

df = df.withColumn("RELEASE_DATE", df["RELEASE_DATE"].cast(DateType()))
df = df.withColumn("PRICE", df["PRICE"].cast(DoubleType()))
df = df.withColumn("DLC_COUNT", df["DLC_COUNT"].cast(IntegerType()))
df = df.withColumn("ACHIEVEMENT", df["ACHIEVEMENT"].cast(IntegerType()))
df = df.withColumn("PEAK_CONCURRENT_USERS", df["PEAK_CONCURRENT_USERS"].cast(IntegerType()))

df.show()
df.printSchema()

(1) Spark Jobs
df: pyspark.sql.dataframe.DataFrame = [STEAM_ID: string, TITLE_NAME: string ... 5 more fields]
| 1146370|Master of Magic C...|  2020-02-25| 5.99|        2|          0|                  49|
| 1597460|Fantasia Sango My...|  2023-01-10|  0.0|        0|          0|                   0|
|  407300|The Last NightMar...|  2015-11-02| 3.99|        1|         45|                   0|
|  349220|  The Black Watchmen|  2015-08-27| 9.99|        5|         48|                   3|
| 1342620|Werewolf: The Apo...|  2020-10-13|14.99|        1|         38|                   0|
|  891360|          Hellcoming|  2021-12-06| 4.99|        0|         26|                   0|
| 1774100|            Pet idle|  2021-12-09|  0.0|        0|         15|                   0|
|  454410|        GIGA WRECKER|  2017-02-06|19.99|        1|         28|                   2|
|  337930|Nameless ~The one...|  2014-12-24|29.99|        9|          0|                   1|
+--------+--------------------+------------+-----+---------+-----------+--------------------+
only showing top 20 rows

root
 |-- STEAM_ID: string (nullable = true)
 |-- TITLE_NAME: string (nullable = true)
 |-- RELEASE_DATE: date (nullable = true)
 |-- PRICE: double (nullable = true)
 |-- DLC_COUNT: integer (nullable = true)
 |-- ACHIEVEMENT: integer (nullable = true)
 |-- PEAK_CONCURRENT_USERS: integer (nullable = true)
```

Imagem 14: Correção no tipo de dado das colunas.



```
3 days ago (7s)                                                    9

# Validação no STEAM_ID

# Garantindo que não tem ID repetido

distinct_values_df = df.groupBy("STEAM_ID").count()
distinct_values_df.filter(distinct_values_df["count"] > 1).show()

# Garantindo que não tem ID negativo
df.filter(df["STEAM_ID"] < 0).show()
df.filter(df["STEAM_ID"].isNull()).show()

(8) Spark Jobs
distinct_values_df: pyspark.sql.dataframe.DataFrame = [STEAM_ID: string, count: long]
+--------+-----+
|STEAM_ID|count|
+--------+-----+
+--------+-----+

+--------+----------+------------+-----+---------+-----------+--------------------+
|STEAM_ID|TITLE_NAME|RELEASE_DATE|PRICE|DLC_COUNT|ACHIEVEMENT|PEAK_CONCURRENT_USERS|
+--------+----------+------------+-----+---------+-----------+--------------------+
+--------+----------+------------+-----+---------+-----------+--------------------+

+--------+----------+------------+-----+---------+-----------+--------------------+
|STEAM_ID|TITLE_NAME|RELEASE_DATE|PRICE|DLC_COUNT|ACHIEVEMENT|PEAK_CONCURRENT_USERS|
+--------+----------+------------+-----+---------+-----------+--------------------+
+--------+----------+------------+-----+---------+-----------+--------------------+

1
```

Imagem 15: Validação na coluna STEAM_ID.



```
3 days ago (4s)                                                    10

# Validação no TITLE_NAME

# Garantindo que não tem nome vázio
df_filtered = df.filter(df["TITLE_NAME"] == "")
print(df_filtered.count())

# Removendo os Títulos NULL, pois não tem como saber quais são
df.filter(df["TITLE_NAME"].isNull()).show()
df = df.dropna(subset=['TITLE_NAME'])
df.filter(df["TITLE_NAME"].isNull()).show()
```

(8) Spark Jobs

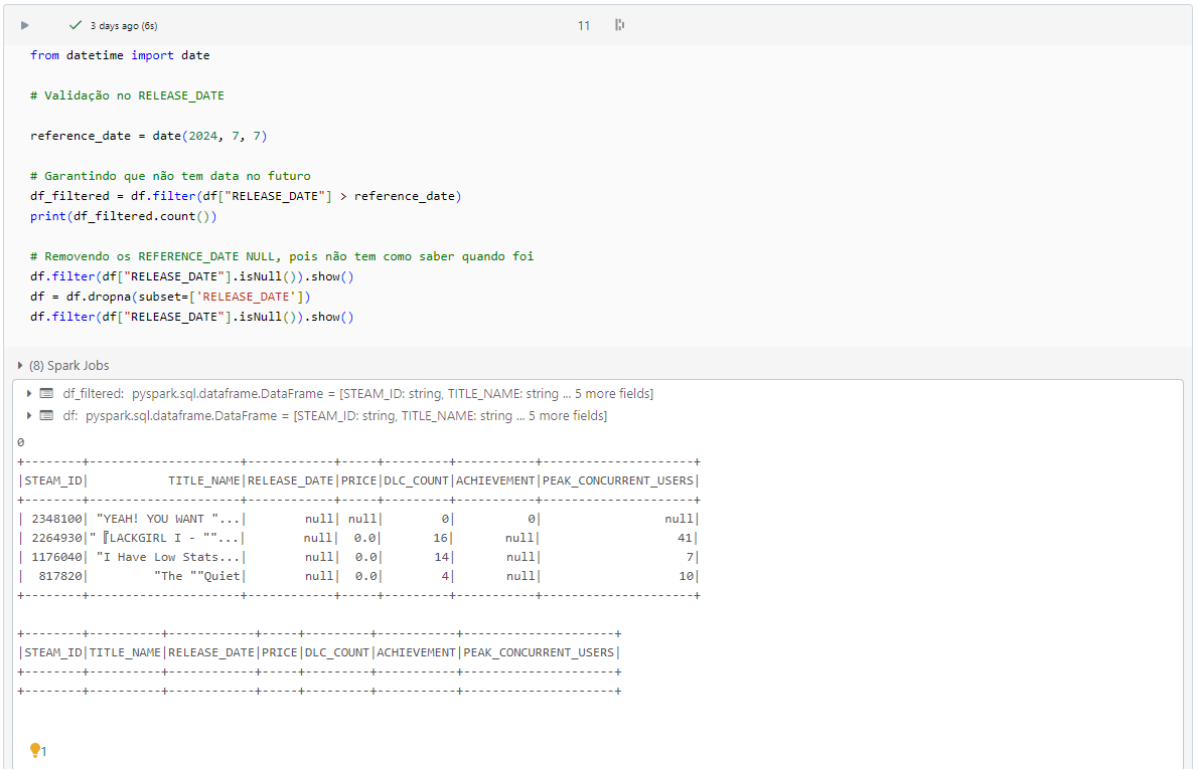
df_filtered: pyspark.sql.dataframe.DataFrame = [STEAM_ID: string, TITLE_NAME: string ... 5 more fields]

df: pyspark.sql.dataframe.DataFrame = [STEAM_ID: string, TITLE_NAME: string ... 5 more fields]

```
0
+--------+----------+------------+-----+---------+-----------+--------------------+
|STEAM_ID|TITLE_NAME|RELEASE_DATE|PRICE|DLC_COUNT|ACHIEVEMENT|PEAK_CONCURRENT_USERS|
+--------+----------+------------+-----+---------+-----------+--------------------+
|  396420|      null|  2016-11-01|  0.0|        0|          0|                   0|
| 1116910|      null|  2019-09-25| 2.79|        0|          0|                   0|
| 1347240|      null|  2021-04-20|24.99|        0|          0|                   0|
+--------+----------+------------+-----+---------+-----------+--------------------+

+--------+----------+------------+-----+---------+-----------+--------------------+
|STEAM_ID|TITLE_NAME|RELEASE_DATE|PRICE|DLC_COUNT|ACHIEVEMENT|PEAK_CONCURRENT_USERS|
+--------+----------+------------+-----+---------+-----------+--------------------+
+--------+----------+------------+-----+---------+-----------+--------------------+
```

1

Imagem 16: Validação na coluna TITLE_NAME.



```
3 days ago (6s)                                                    11

from datetime import date

# Validação no RELEASE_DATE

reference_date = date(2024, 7, 7)

# Garantindo que não tem data no futuro
df_filtered = df.filter(df["RELEASE_DATE"] > reference_date)
print(df_filtered.count())

# Removendo os REFERENCE_DATE NULL, pois não tem como saber quando foi
df.filter(df["RELEASE_DATE"].isNull()).show()
df = df.dropna(subset=['RELEASE_DATE'])
df.filter(df["RELEASE_DATE"].isNull()).show()
```

(8) Spark Jobs

df_filtered: pyspark.sql.dataframe.DataFrame = [STEAM_ID: string, TITLE_NAME: string ... 5 more fields]

df: pyspark.sql.dataframe.DataFrame = [STEAM_ID: string, TITLE_NAME: string ... 5 more fields]

```
0
+--------+--------------------+------------+-----+---------+-----------+--------------------+
|STEAM_ID|          TITLE_NAME|RELEASE_DATE|PRICE|DLC_COUNT|ACHIEVEMENT|PEAK_CONCURRENT_USERS|
+--------+--------------------+------------+-----+---------+-----------+--------------------+
| 2348100| "YEAH! YOU WANT "...|        null| null|        0|          0|                null|
| 2264930|" ▛LACKGIRL I - ""...|        null|  0.0|       16|       null|                  41|
| 1176040| "I Have Low Stats...|        null|  0.0|       14|       null|                   7|
|  817820|         "The ""Quiet|        null|  0.0|        4|       null|                  10|
+--------+--------------------+------------+-----+---------+-----------+--------------------+

+--------+----------+------------+-----+---------+-----------+--------------------+
|STEAM_ID|TITLE_NAME|RELEASE_DATE|PRICE|DLC_COUNT|ACHIEVEMENT|PEAK_CONCURRENT_USERS|
+--------+----------+------------+-----+---------+-----------+--------------------+
+--------+----------+------------+-----+---------+-----------+--------------------+
```

1

Imagem 17: Validação na coluna RELEASE_DATE.

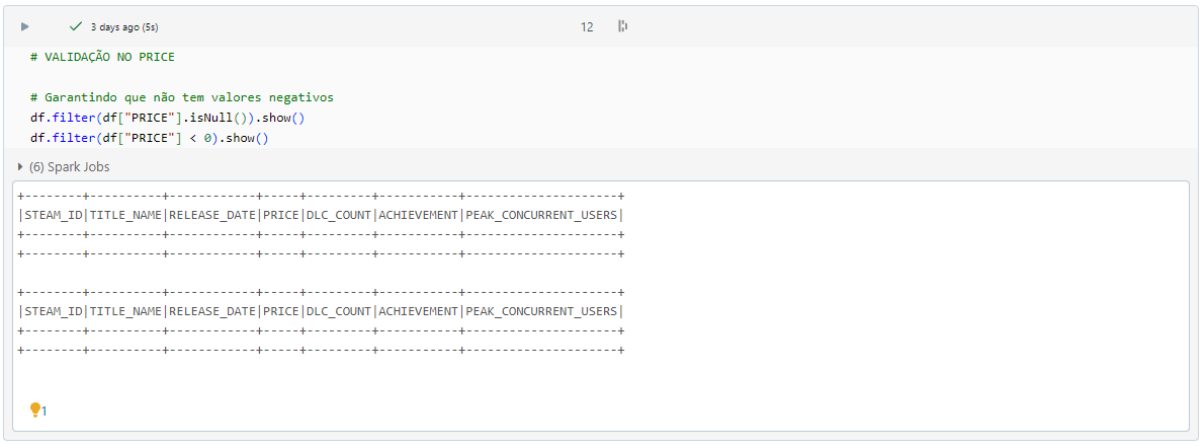


Imagem 18: Validação na coluna PRICE.

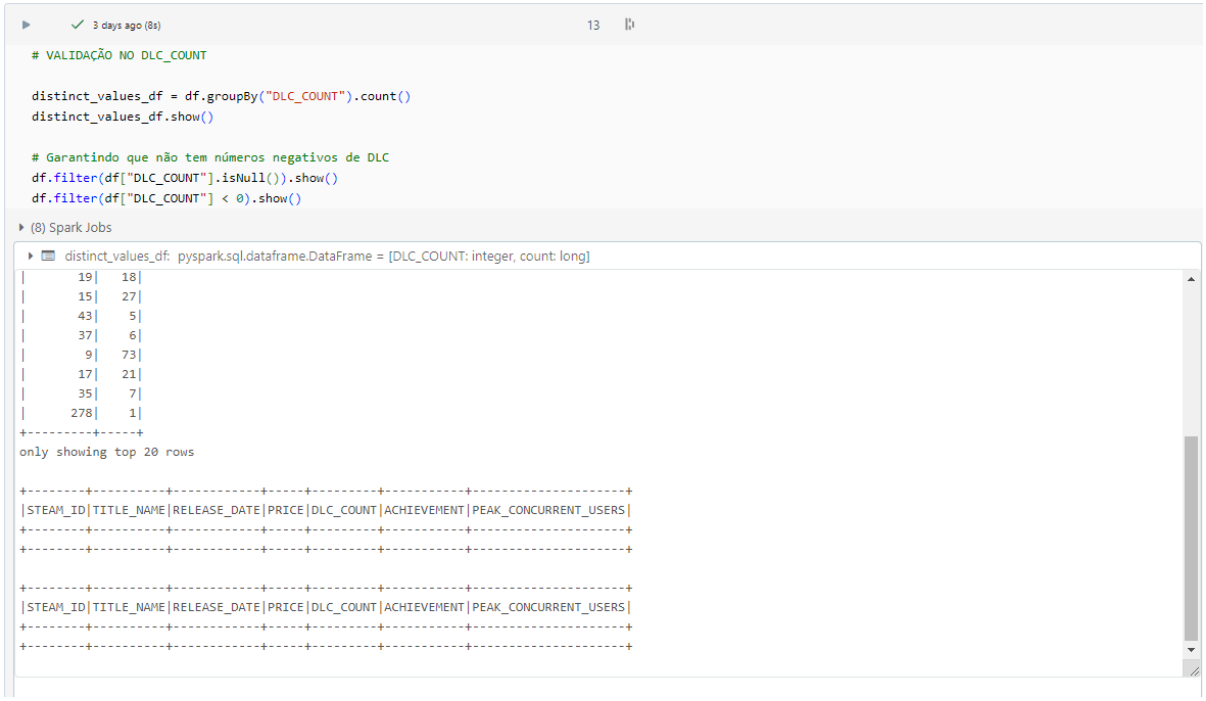


Imagem 19: Validação na coluna DLC_COUNT.

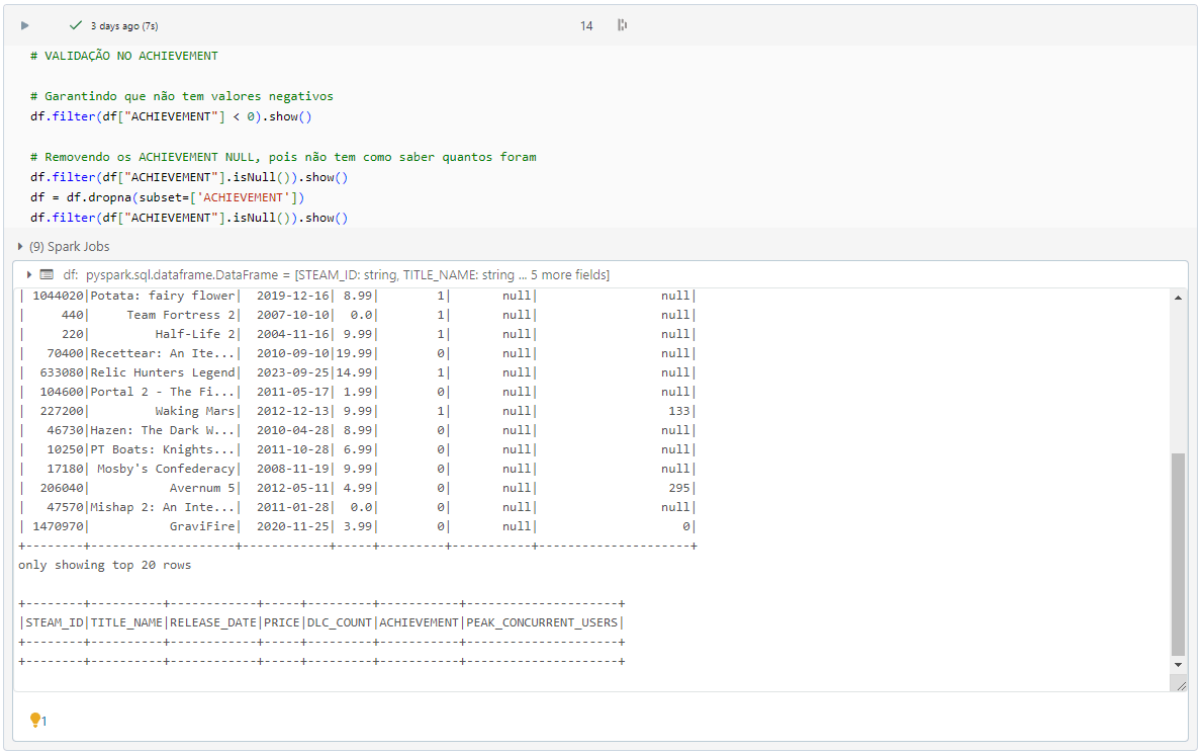


Imagem 20: Validação na coluna ACHIEVEMENT.

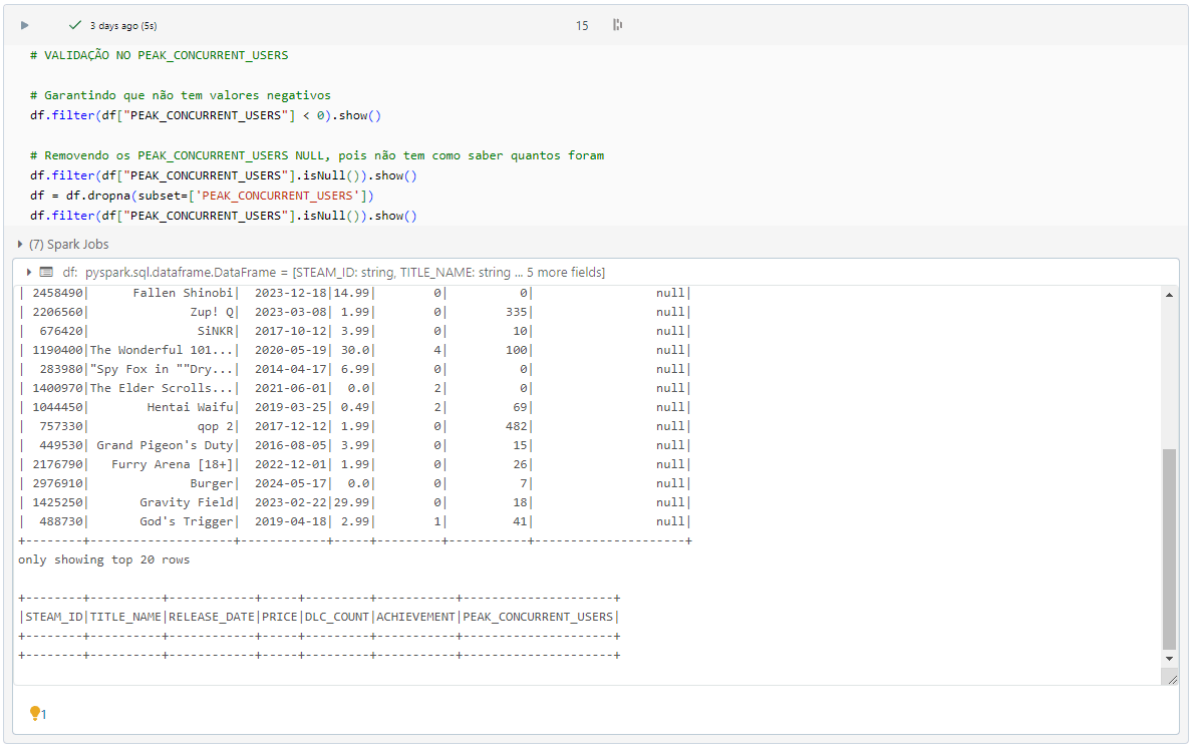


Imagem 21: Validação na coluna PEAK_CONCURRENT_USERS.

Imagem 22: Persistência dos dados na camada prata.

Catálogo de Dados

NOME DA COLUNA	TIPO DE DADO	DESCRIÇÃO	VALORES ACEITÁVEIS
STEAM_ID	INT	IDENTIFICADOR ÚNICO DO TÍTULO DENTRO DA PLATAFORMA STEAM	VALOR INTEIRO COMEÇANDO NO 0 E INDO ATÉ O TAMANHO DO CATÁLOGO DE PRODUTOS DA PLATAFORMA STEAM, SEM QUE HAJA 2 OU MAIS TÍTULOS REFERENCIANDO O MESMO IDENTIFICADOR.
TITLE_NAME	STRING	NOME DO TÍTULO	VALOR TEXTO QUE NÃO SEJA VAZIO.
RELEASE_DATE	DATETIME	DATA DE LANÇAMENTO DO TÍTULO	VALOR DO TIPO DATA QUE NÃO TENHA DATAS NO FUTURO.
PRICE	DOUBLE	PREÇO PARA A AQUISIÇÃO DO TÍTULO	VALOR NÚMERICO EM DÓLARES (USD) QUE NÃO PODE SER NEGATIVO.
DLC_COUNT	INT	NÚMERO DE DLCS DO TÍTULO	VALOR INTEIRO QUE NÃO PODE SER NEGATIVO.
ACHIEVEMENT	INT	NÚMERO DE CONQUISTAS QUE PODEM SER OBTIDAS NO TÍTULO	VALOR INTEIRO QUE NÃO PODE SER NEGATIVO.
PEAK_CONCURRENT_USERS	INT	NÚMERO MÁXIMO DE JOGADORES CONCORRENTES	VALOR INTEIRO QUE NÃO PODE SER NEGATIVO.

Imagem 23: Catálogo de Dados

4. Análise

Após todo esse passo a passo e a obtenção dos dados prontos para serem analisados, passa a ser possível gerar algumas análises e identificar se é possível responder a todas as perguntas contidas no objetivo. Antes da análise poder ser executada, devido ao fato de estarmos utilizando a versão Community do Databricks, foi necessário fazer um contorno para que a camada prata fosse executada antes da camada ouro.

Seguindo com o workflow artificial, agora é possível garantir que todo o processo de extração, transformação e carga (ETL) seja executado e que o usuário final possa extrair a informação necessária para responder às perguntas. Nas imagens a seguir, a camada ouro será apresentada referenciado as resposta das perguntas com o SparkSQL.

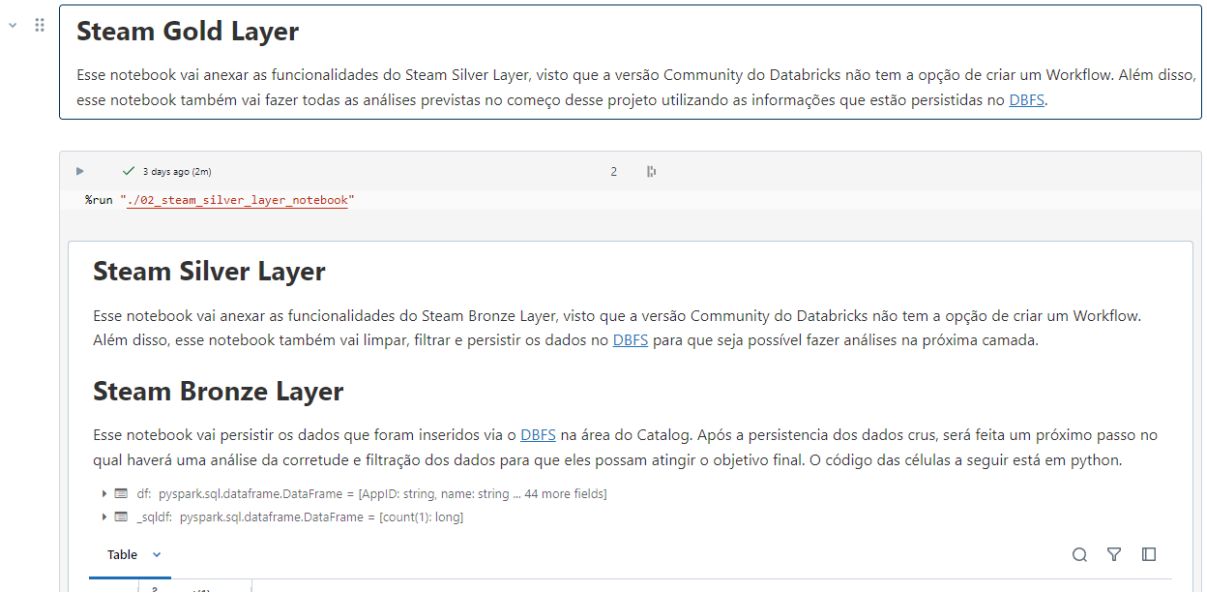
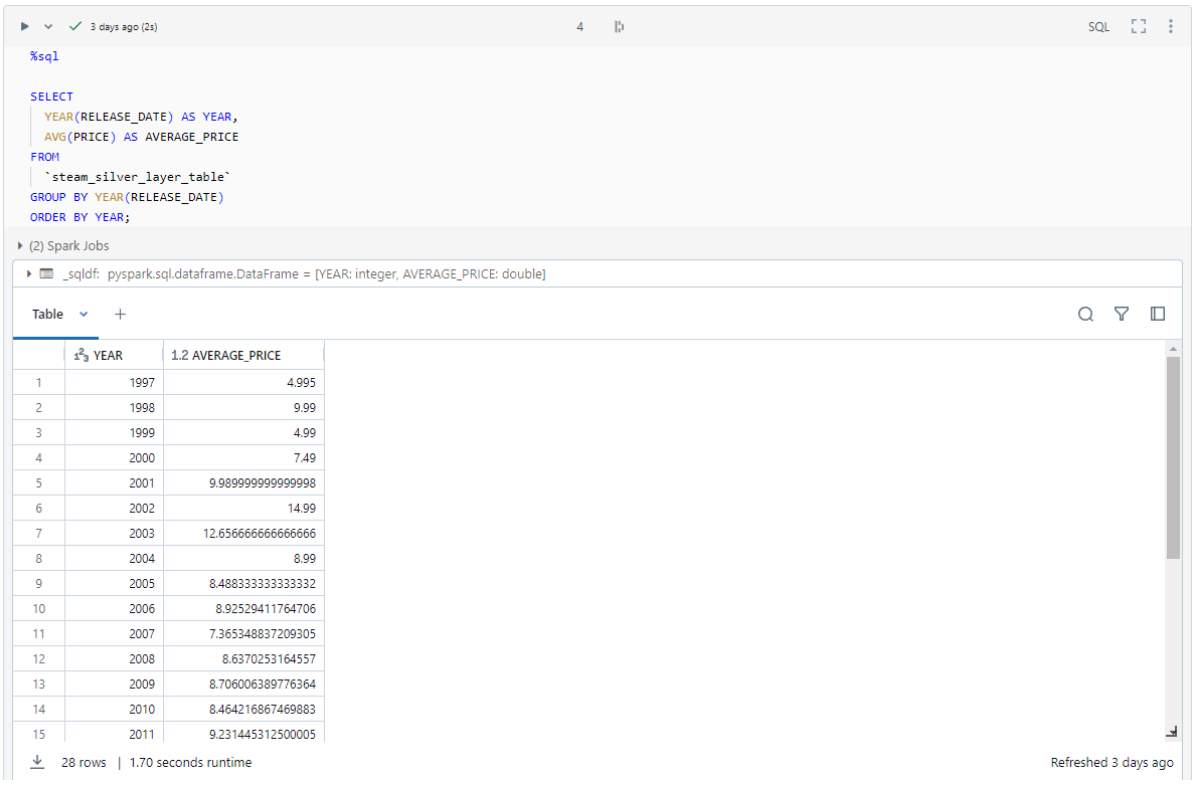


Imagem 24: Garantia de conexão entre a camada prata e a ouro.



```
%sql

SELECT
  YEAR(RELEASE_DATE) AS YEAR,
  AVG(PRICE) AS AVERAGE_PRICE
FROM
  `steam_silver_layer_table`
GROUP BY YEAR(RELEASE_DATE)
ORDER BY YEAR;
```

(2) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [YEAR: integer, AVERAGE_PRICE: double]

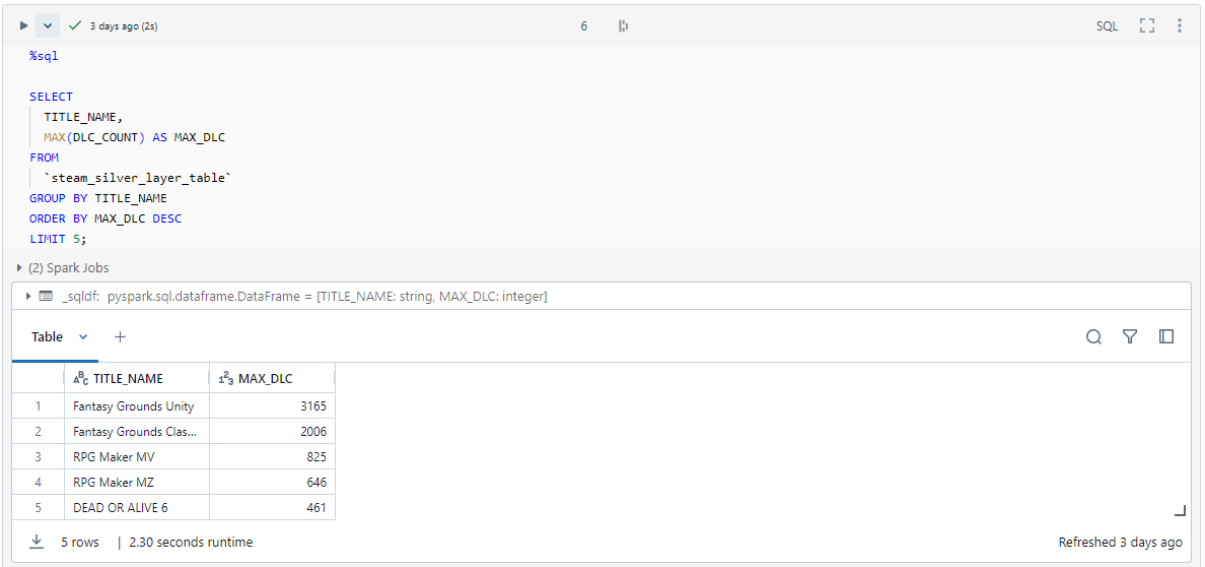
Table

	YEAR	AVERAGE_PRICE
1	1997	4.995
2	1998	9.99
3	1999	4.99
4	2000	7.49
5	2001	9.989999999999998
6	2002	14.99
7	2003	12.656666666666666
8	2004	8.99
9	2005	8.488333333333332
10	2006	8.92529411764706
11	2007	7.365348837209305
12	2008	8.6370253164557
13	2009	8.706006389776364
14	2010	8.464216867469883
15	2011	9.231445312500005

28 rows | 1.70 seconds runtime

Refreshed 3 days ago

Imagem 25: Qual o preço médio por ano dos jogos na Steam?



```
%sql

SELECT
  TITLE_NAME,
  MAX(DLC_COUNT) AS MAX_DLC
FROM
  `steam_silver_layer_table`
GROUP BY TITLE_NAME
ORDER BY MAX_DLC DESC
LIMIT 5;
```

(2) Spark Jobs

_sqldf: pyspark.sql.dataframe.DataFrame = [TITLE_NAME: string, MAX_DLC: integer]

Table

	TITLE_NAME	MAX_DLC
1	Fantasy Grounds Unity	3165
2	Fantasy Grounds Clas...	2006
3	RPG Maker MV	825
4	RPG Maker MZ	646
5	DEAD OR ALIVE 6	461

5 rows | 2.30 seconds runtime

Refreshed 3 days ago

Imagem 26: Quais são os 5 jogos com mais DLC (Downloadable Content)?

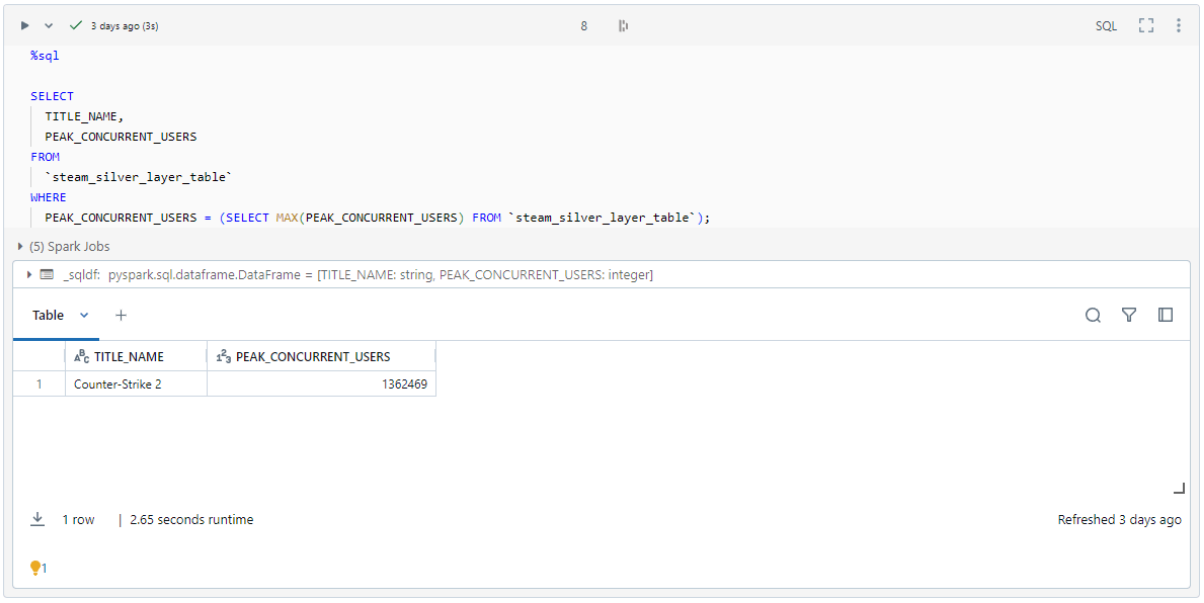


Imagem 27: Qual é o jogo com o maior pico de jogadores na Steam?

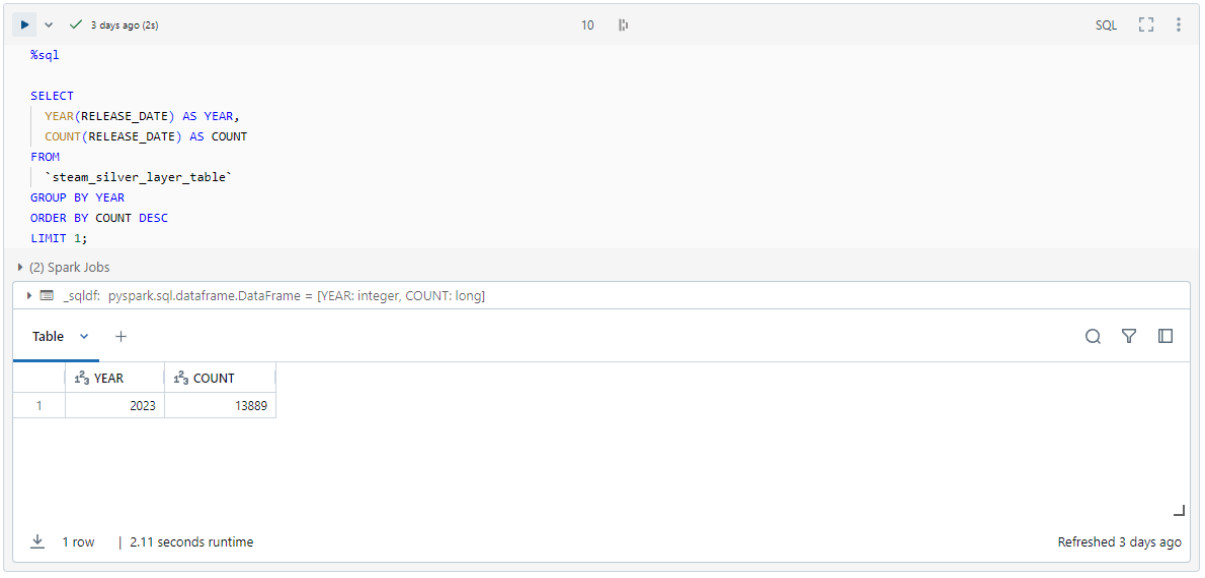


Imagem 28: Qual ano teve a maior quantidade de lançamentos de jogos na Steam?

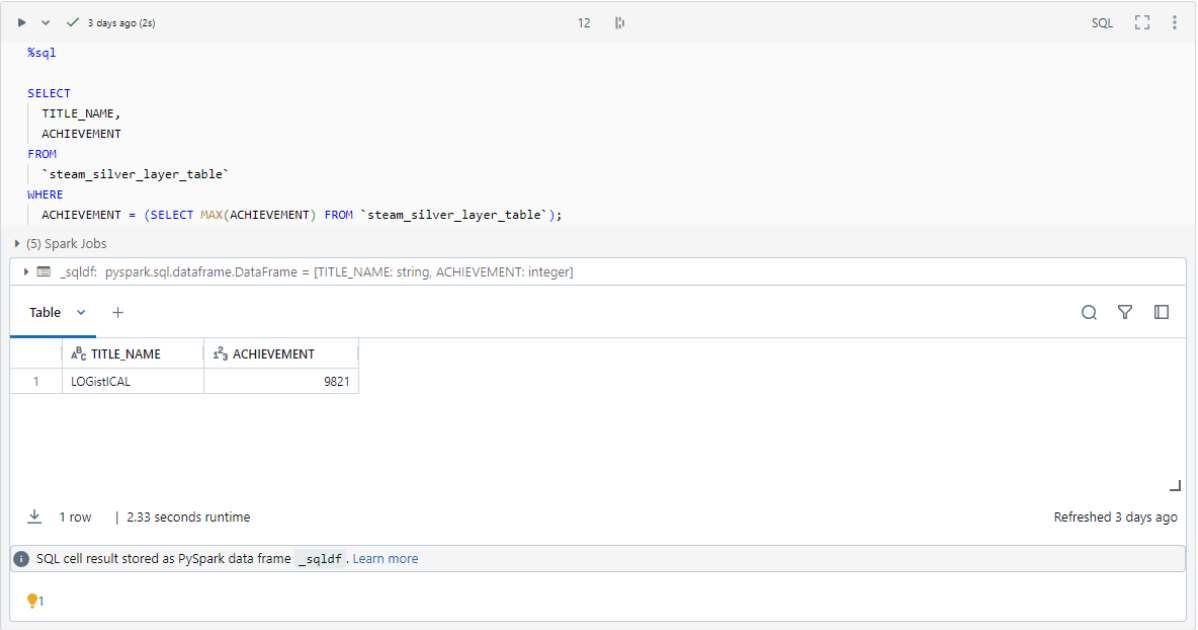


Imagem 28: Qual jogo tem o maior número de conquistas?

Autoavaliação

A autoavaliação desse projeto foi bem positiva. Os dados escolhidos acabaram refletindo numa modelagem de dados flat, o que acabou facilitando as camadas mais avançadas e a identificação da possibilidade de responder as perguntas. O único problema encontrado foi a necessidade da API Key que acabaria gerando a possibilidade de vazamento de credencial se fosse armazenada no GitHub.

Finalizando esse projeto, acredito que os conceitos de criação de pipeline foram utilizados da forma correta e o estudo adicional da arquitetura medallion facilitou o desenvolvimento deste estudo. Seguindo pro futuro, acho que essa análise, embora inicial, possa acabar evoluindo para entender a dinâmica do mercado de jogos e o que os usuários realmente desejam.