

# Notebook 2-kafka2bronze

**Movendo os dados do Kafka para o Delta**

Na primeira etapa, você enviou os dados de clientes, pedidos e itens de pedidos do Postgres para o Kafka. O Apache Kafka armazena as mensagens em estrutura de tópicos, de forma confiável e escalável. Agora, na segunda etapa, iremos ler os dados do Kafka e armazenar na tabela bronze no Delta Lake. A tabela bronze armazena os dados brutos, sem nenhuma transformação, para que posteriormente seja possível aplicar qualquer transformação sobre os dados, sem perder a origem da informação.

**Preparando o ambiente**

O código abaixo adiciona a raiz do projeto, que contém códigos e dados necessários para o "Hands on".

```
In [1]: root = '/home/bigdata/jupyterhub'
import sys
sys.path.append(root)
ud = 'delta'
```

O trecho de código abaixo prepara o ambiente, carregando códigos auxiliares e dados de configuração. O código disponível no pacote `common.utils` na classe `DataframeUtils` contém vários métodos que facilitam a leitura e escrita dos dados do Postgres. A classe `DataframeUtils` também inicia uma instância do Apache Spark com o Delta Lake integrado ao Spark.

Já o arquivo `config.yaml` tem os dados de acesso ao Postgres e Kafka.

```
In [2]: import yaml
from common.utils import DataframeUtils
import pyspark.sql.functions as F
config = yaml.safe_load(open('/home/bigdata/jupyterhub/config.yaml'))
dfu = DataframeUtils(config)
```

**Leitura dos dados do Kafka e escrita no Delta Lake**

O método `process` facilita a leitura dos dados do Kafka utilizando o Apache Spark. Com o trecho de código abaixo, os dados são lidos do tópico do Kafka e armazenados no Dataframe `df` (de forma lazy). Para ler os dados Kafka, tem que passar como argumento o `format('kafka')` para indicar que é uma leitura do Kafka, o host do servidor bootstrap, o tópico que irá ler as mensagens e o offset para indicar a partir de onde irá ler as mensagens.

**Leitura dos dados do Kafka e escrita no Delta Lake**

O método `process` facilita a leitura dos dados do Kafka utilizando o Apache Spark. Com o trecho de código abaixo, os dados são lidos do tópico do Kafka e armazenados no Dataframe `df` (de forma lazy). Para ler os dados Kafka, tem que passar como argumento o `format('kafka')` para indicar que é uma leitura do Kafka, o host do servidor bootstrap, o tópico que irá ler as mensagens e o offset para indicar a partir de onde irá ler as mensagens.

```
df = dfu.spark() \
    .format('kafka') \
    .option('kafka.bootstrap.servers', kafka) \
    .option('subscribe', topic) \
    .option('startingOffsets', 'earliest') \
    .option('maxOffsetsPerTrigger', max_triggers) \
    .load()
```

Os dados lidos do Kafka estão no formato `JSON` e, por isso, tem que ser transformados para o formato tabular para poder inserir no formato Delta. Por isso, a chave `key` do Kafka torna uma coluna do Delta, com nome `key`, que identifica a tupla e o conteúdo da mensagem em `value` é escrito em outra coluna com o nome `value`. Como os dados do Kafka chegam no formato de stream, eles são armazenados utilizando a função `writeStream`, que vai recebendo os dados do Kafka, fazendo a transformação para o formato tabular e armazenando na tabela Gold. O trecho `.option('mergeSchema', 'true')` indica que se houver mudanças no esquema de dados, será feito um "merge" dos esquemas no Delta Lake. O Delta Lake armazena o histórico de alterações dos dados e permite indicar o diretório de checkpoint através da opção `.option('checkpointLocation', checkpoint_dir)`.

```
df \
    .withColumn('key', F.col('key').cast('string')) \
    .withColumn('value', F.col('value').cast('string')) \
    .writeStream \
    .option('mergeSchema', 'true') \
    .format('delta') \
    .outputMode('append') \
    .option('checkpointLocation', checkpoint_dir) \
    .start(output_dir)
```

```
In [3]: def process(topic, output_dir, checkpoint_dir, kafka, max_triggers:int=1000):
    df = dfu.spark() \
        .readStream \
        .format('kafka') \
        .option('kafka.bootstrap.servers', kafka) \
        .option('subscribe', topic) \
        .option('startingOffsets', 'earliest') \
        .option('maxOffsetsPerTrigger', max_triggers) \
        .load()
```

[localhost:8888/notebooks/etl/2-kafka2bronze.ipynb](http://localhost:8888/notebooks/etl/2-kafka2bronze.ipynb)

jupyter 2-kafka2bronze Last Checkpoint 20/10/2025 (autosaved)

In [3]:

```
def process(topic, output_dir, checkpoint_dir, kafka, max_triggers:int=1000):
    df = dfu.spark() \
        .format('kafka') \
        .option('kafka.bootstrap.servers', kafka) \
        .option('subscribe', topic) \
        .option('startingOffsets', 'earliest') \
        .option('maxOffsetsPerTrigger', max_triggers) \
        .load()

    records = df \
        .withColumn('key', F.col("key").cast('string')) \
        .withColumn('value', F.col('value').cast('string')) \
        .writeStream \
        .option("mergeSchema", "true") \
        .format("delta") \
        .outputMode('append') \
        .option('checkpointLocation', checkpoint_dir) \
        .start(output_dir)
```

Nos próximos três trechos de código, serão escritos os dados de cliente em `/delta/dados-clientes-bronze` e os dados de pedidos em `/delta/dados-pedidos-bronze`.

In [4]: kafka = dfu.config()['kafka']['host']

In [5]: dsc = process("clientes", wd + '/data/clientes-bronze', wd + '/checkpoints/clientes-checkpoint', kafka, 10000)

In [6]: dsp = process("pedidos", wd + '/data/pedidos-bronze', wd + '/checkpoints/pedidos-checkpoint', kafka, 10000)

### Exercício

Agora é com você! Neste exercício você irá utilizar os códigos apresentados acima como exemplo para fazer a **leitura dos dados no Kafka e escrita dos dados com o Delta Lake**.

**Não será permitido utilizar a função process()**. Você deverá construir seu próprio código utilizando os dados fornecidos abaixo:

In [7]:

```
spark = dfu.spark()
topic = 'itens'
output_dir = wd + '/data/itens-bronze'
checkpoint_dir = wd + '/checkpoints/itens-checkpoint'
max_triggers=10000
```

Manchete de es...  
Nº vence no...

18:41  
27/10/2025

[localhost:8888/notebooks/etl/2-kafka2bronze.ipynb](http://localhost:8888/notebooks/etl/2-kafka2bronze.ipynb)

jupyter 2-kafka2bronze Last Checkpoint 20/10/2025 (autosaved)

**Exercício**

Agora é com você! Neste exercício você irá utilizar os códigos apresentados acima como exemplo para fazer a **leitura dos dados no Kafka e escrita dos dados com o Delta Lake**.

**Não será permitido utilizar a função process()**. Você deverá construir seu próprio código utilizando os dados fornecidos abaixo:

In [7]:

```
spark = dfu.spark()
topic = 'itens'
output_dir = wd + '/data/itens-bronze'
checkpoint_dir = wd + '/checkpoints/itens-checkpoint'
max_triggers=10000
```

No trecho de código abaixo, você deverá ler o fluxo de dados do tópico do Kafka e armazenar no Dataframe Spark `df_itens`.

In [8]:

```
df_itens = spark \
    .readStream \
    .format('kafka') \
    .option('kafka.bootstrap.servers', kafka) \
    .option('subscribe', topic) \
    .option('startingOffsets', 'earliest') \
    .option('maxOffsetsPerTrigger', max_triggers) \
    .load()
```

In [9]:

```
dsl = df_itens \
    .withColumn('key', F.col("key").cast('string')) \
    .withColumn('value', F.col('value').cast('string')) \
    .writeStream \
    .option("mergeSchema", "true") \
    .format("delta") \
    .outputMode('append') \
    .option('checkpointLocation', checkpoint_dir) \
    .start(output_dir)
```

**Monitorando o fluxo de dados**

Os códigos abaixo monitoram o fluxo de dados que está sendo processado nos Dataframes Spark e escritos no formato Delta.

In [10]:

```
dsp.status
```

Manchete de es...  
Nº vence no...

18:41  
27/10/2025

**Monitorando o fluxo de dados**

Os códigos abaixo monitoram o fluxo de dados que está sendo processado nos Dataframes Spark e escritos no formato Delta.

```
In [10]: dsp.status
Out[10]: {'message': 'Getting offsets from KafkaV2[Subscribe[pedidos]]',
           'isDataAvailable': False,
           'isTriggerActive': True}

In [11]: dsc.status
Out[11]: {'message': 'Getting offsets from KafkaV2[Subscribe[clientes]]',
           'isDataAvailable': False,
           'isTriggerActive': True}

In [12]: dsl.status
Out[12]: {'message': 'Processing new data',
           'isDataAvailable': True,
           'isTriggerActive': True}

In [13]: dsc.lastProgress
Out[13]: {'id': '4be3dc78-c8b0-46b2-9496-7f0334753009',
           'runId': '39e1988-46b0-405a-844d-d0bbff182dc5',
           'name': None,
           'timestamp': '2025-10-27T20:56:26.531Z',
           'batchId': 8,
           'numInputRows': 0,
           'processDurationSecond': 0.0,
           'duration': {'getEndOffset': 0,
                        'setOffsetRange': 4216,
                        'triggerExecution': 4363},
           'stateOperators': [],
           'sources': [{"description": "KafkaV2[Subscribe[clientes]]",
                        "startOffset": {"clientes": "0": 180000}},
                      {"endOffset": {"clientes": "0": 180000}},
                      {"numInputRows": 0,
                       "processedRowsPerSecond": 0.0}],
           'sink': {"description": "DeltaSink[/delta/data/clientes-bronze]"}}

In [14]: dsp.lastProgress
Out[14]: {'id': '8dc13055-a55d-4b4f-8a47-e845e6381bbf',
           'runId': 'edf45ac4-857a-4098-b406-d5a8ea56104e',
           'name': None,
           'timestamp': '2025-10-27T20:56:31.592Z',
           'batchId': 127,
           'numInputRows': 0,
           'processDurationSecond': 0.0,
           'duration': {'getEndOffset': 1,
                        'setOffsetRange': 3168,
                        'triggerExecution': 3273},
           'stateOperators': [],
           'sources': [{"description": "KafkaV2[Subscribe[pedidos]]",
                        "startOffset": {"pedidos": "0": 3424831}},
                      {"endOffset": {"pedidos": "0": 3424831}},
                      {"numInputRows": 0,
                       "processedRowsPerSecond": 0.0}],
           'sink': {"description": "DeltaSink[/delta/data/pedidos-bronze]"}}

In [15]: dsl.lastProgress
In [16]: dfu.print_streaming_chart(dsc)
In [17]: dfu.print_streaming_chart(dsp)
In [18]: dfu.print_streaming_chart(dsl)
```

```
In [14]: dsp.lastProgress
Out[14]: {'id': '8dc13055-a55d-4b4f-8a47-e845e6381bbf',
           'runId': 'edf45ac4-857a-4098-b406-d5a8ea56104e',
           'name': None,
           'timestamp': '2025-10-27T20:56:31.592Z',
           'batchId': 127,
           'numInputRows': 0,
           'processDurationSecond': 0.0,
           'duration': {'getEndOffset': 1,
                        'setOffsetRange': 3168,
                        'triggerExecution': 3273},
           'stateOperators': [],
           'sources': [{"description": "KafkaV2[Subscribe[pedidos]]",
                        "startOffset": {"pedidos": "0": 3424831}},
                      {"endOffset": {"pedidos": "0": 3424831}},
                      {"numInputRows": 0,
                       "processedRowsPerSecond": 0.0}],
           'sink': {"description": "DeltaSink[/delta/data/pedidos-bronze]"}}

In [15]: dsl.lastProgress
In [16]: dfu.print_streaming_chart(dsc)
In [17]: dfu.print_streaming_chart(dsp)
In [18]: dfu.print_streaming_chart(dsl)

-----  

ValueError                                Traceback (most recent call last)  

</ipython-input-18-bec335a99ff4> in <module>  

----> 1 dfu.print_streaming_chart(dsl)  

~/jupyterhub/common/utils.py in print_streaming_chart(self, dsc)
    75         progress = pd.read_json(json.dumps(dsc.recentProgress))
    76
--> 77         fig = px.bar(progress, x='timestamp', y='processedRowsPerSecond')
    78         fig.show()
    79         time.sleep(5)  

  

/opt/miniconda3/lib/python3.7/site-packages/plotly/express/_chart_types.py in bar(data_frame, x, y, color, facet_row, facet_col, facet_col_wrap, hover_name, hover_data, custom_data, text, error_x, error_x_minus, error_y, error_y_minus, animation_frame, animation_group, category_orders, labels, color_discrete_sequence, color_discrete_map, color_continuous_scale, range_color, color_continuous_mixin, orientation, harmonic, title, template, width, height)
```

```

<-- C ⌂ localhost:8888/notebooks/etl/2-kafka2bronze.ipynb
Google GitHub 9.2. math — Mathe... W3Schools Online... SDO| Solar Dynami... Sonic the Hedgehog... Técnica Feynman... particulas virtuais... futurasções quânticas... cn_algoritmos.pdf Gerenciando banco... paa_Grafos_1pp.pdf > Outros favoritos
jupyter 2-kafka2bronze Last Checkpoint 20/10/2025 (autosaved)
File Edit View Insert Cell Kernel Help Trusted Python 3 Logout
B + < > D Run Code
78 fig.show()
79 time.sleep(5)
/opt/miniconda3/lib/python3.7/site-packages/plotly/express/_chart_types.py in bar(data_frame, x, y, color, facet_row, facet_col_wrap, hover_name, hover_data, custom_data, text, error_x, error_x_minus, error_y, error_y_minus, animation_frame, animation_group, category_orders, labels, color_discrete_sequence, color_discrete_map, color_continuous_scale, range_color, col_or_continuous_idioms, opacity, orientation, barmode, log_x, log_y, range_x, range_y, title, template, width, height)
339     constructor=go.Bar,
340     trace_patch=dict(textposition="auto"),
-> 341     layout_patch=dict(barmode=barmode),
342   )
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
627
628
629
629
630
631
632
633
634
635
635
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
1643
1643
1644
1644
1645
1645
1646
1646
1647
1647
1648
1648
1649
1649
1650
1650
1651
1651
1652
1652
1653
1653
1654
1654
1655
1655
1656
1656
1657
1657
1658
1658
1659
1659
1660
166
```

[localhost:8888/notebooks/etl/2-5-read-bronze.ipynb](http://localhost:8888/notebooks/etl/2-5-read-bronze.ipynb)

jupyter 2-5-read-bronze Last Checkpoint: uma hora atrás (autosaved)

File Edit View Insert Cell Kernel Help Trusted Python 3 Logout

In [3]:

```
pedidos = dfu. \
    spark(). \
    read. \
    format('delta'). \
    load(f'{wd}/data/pedidos-bronze')
```

O trecho abaixo obtém a quantidade de itens no Dataframe, execute-o algumas vezes em intervalos de alguns segundos e observe o valor sendo incrementado.

Veja que não é necessário recarregar o Dataframe para que o valor seja atualizado, o Spark já cuida dessa parte para você. O valor final deverá ser 310.976.

In [4]:

```
pedidos.count()
```

Out[4]: 1243984

O trecho abaixo cria um Dataframe de clientes exatamente da mesma forma que o de pedidos, alterando apenas o diretório no HDFS.

In [5]:

```
clientes = dfu. \
    spark(). \
    read. \
    format('delta'). \
    load(f'{wd}/data/clientes-bronze')
```

A quantidade de clientes é muito pequena, portanto quando você executar o trecho abaixo já terá finalizado a escrita dos 15.000 clientes.

In [6]:

```
clientes.count()
```

Out[6]: 60000

O trecho abaixo cria uma temp view do Spark para que possamos executar uma consulta SQL.

In [7]:

```
clientes.createOrReplaceTempView('clientes_bronze')
```

A SQL abaixo é para verificar que não existem clientes repetidos.

Para isso realizamos um group\_by na chave (key) e adicionamos um having count > 1. O resultado abaixo deverá retornar uma lista vazia.

Manchete de es...  
Norris vence no... 18:43  
27/10/2025

[localhost:8888/notebooks/etl/2-5-read-bronze.ipynb](http://localhost:8888/notebooks/etl/2-5-read-bronze.ipynb)

jupyter 2-5-read-bronze Last Checkpoint: uma hora atrás (autosaved)

File Edit View Insert Cell Kernel Help Trusted Python 3 Logout

In [7]:

```
clientes.createOrReplaceTempView('clientes_bronze')
```

O trecho abaixo cria uma temp view do Spark para que possamos executar uma consulta SQL.

In [8]:

```
dfu.spark().sql("""
    select key, count(value)
    from clientes_bronze
    group by 1
    having count(value) > 1
""")
```

Out[8]:

| key  | count(value) |
|------|--------------|
| 1090 | 4            |
| 1159 | 4            |
| 1436 | 4            |
| 1512 | 4            |
| 1572 | 4            |
| 2069 | 4            |
| 2088 | 4            |
| 2136 | 4            |
| 2162 | 4            |
| 2294 | 4            |
| 2904 | 4            |
| 3210 | 4            |
| 3414 | 4            |
| 3606 | 4            |
| 3959 | 4            |
| 4032 | 4            |
| 4821 | 4            |
| ...  | ...          |

Manchete de es...  
Norris vence no... 18:43  
27/10/2025

```

id      count
1512     4
1572     4
2069     4
2088     4
2136     4
2162     4
2294     4
2904     4
3210     4
3414     4
3606     4
3959     4
4032     4
4821     4
4937     4
5325     4
6194     4
only showing top 20 rows

```

**Exercício**

Quer saber quantos itens de pedido foram todos escritos? Adicione abaixo o código responsável por criar o Dataframe de itens de pedido e imprimir a quantidade de itens do Dataframe.

O diretório no HDFS é o /delta/items-bronze e o count final deverá ser 2.410.176.

## Notebook 3-bronze2silver

**Transformando os dados**

No passo anterior nós criamos as Delta Tables e as salvamos no HDFS. Entretanto esses dados não estão no formato ideal para realizar as análises. Os tipos dos dados também não estão apropriados pois todas as colunas foram mapeadas como string. Nesse notebook iremos transformar os dados da coluna value em formato JSON para formar tabular utilizando a função `from_json` do Spark.

**Preparando o ambiente**

O código abaixo adiciona a raiz do projeto, que contém códigos e dados necessários para o "Hands on".

```
In [1]: root = '/home/bigdata/jupyterhub'
import sys
sys.path.append(root)
wd = './delta'
```

O trecho de código abaixo prepara o ambiente, carregando códigos auxiliares e dados de configuração. O código disponível no pacote `common_utils` na classe `DataframeUtils` contém vários métodos que facilitam a leitura e escrita dos dados do Postgres. A classe `DataframeUtils` também inicia uma instância do Apache Spark com o Delta Lake integrado ao Spark.

```
In [2]: import yaml
from common.utils import DataframeUtils
import pyspark.sql.functions as F
config = yaml.safe_load(open('../config.yaml'))
dfu = DataframeUtils(config)
```

Já o arquivo `config.yaml` tem os dados de acesso ao Postgres e Kafka.

```
In [3]: clientes = dfu.\
    spark().\
    read.\
    format('delta').\
    load(f'{wd}/data/clientes-bronze')
```

[localhost:8888/notebooks/etl/3-bronze2silver.ipynb](http://localhost:8888/notebooks/etl/3-bronze2silver.ipynb)

jupyter 3-bronze2silver Last Checkpoint 20/10/2025 (autosaved)

File Edit View Insert Cell Kernel Help Trusted Python 3

```
config = yaml.safe_load(open('../config.yaml'))
dfu = DataframeUtils(config)

O trecho abaixo cria o Dataframe no formato delta de clientes apontando para o diretório do HDFS.

In [3]: clientes = dfu. \
    spark(). \
    read. \
    format("delta"). \
    load(f'{wd}/data/clientes-bronze')

Antes de iniciar a extração dos dados, vamos visualizar como eles estão armazenados na bronze table.

In [4]: clientes.select('value').limit(1).show(truncate=False)
+-----+
| value |
+-----+
| {"city": "SANTA TEREZA DE...", "client_id": "300102244", "cnae_id": "47.29-6-02", "defaulting": false, "state": "60"} |
+-----+
```

O trecho abaixo cria uma view temporária para executar códigos com o Spark SQL.

```
In [5]: clientes.createOrReplaceTempView("clientes_bronze")
```

Agora que já sabemos como os dados estão armazenados, podemos realizar a extração das propriedades JSON contidas no campo value utilizando a função `from_json` do Spark. Nesse ponto já vamos alterar o tipo da coluna defaulting para boolean.

A extração dos dados segue o mesmo padrão para todas as colunas:

```
from_json(value, 'cliente_id' <tipo do dado>)
```

A função `from_json` retorna um `Row`, por isso precisamos extrair o campo que queremos, o código fica assim:

```
A função from_json retorna um Row, por isso precisamos extrair o campo que queremos, o código fica assim:
```

Manchete de es... Norms vence no... 18:44 27/10/2025

[localhost:8888/notebooks/etl/3-bronze2silver.ipynb](http://localhost:8888/notebooks/etl/3-bronze2silver.ipynb)

jupyter 3-bronze2silver Last Checkpoint 20/10/2025 (autosaved)

File Edit View Insert Cell Kernel Help Trusted Python 3

```
Introduzindo o from_json do Spark. Precisei alterar o tipo da coluna defaulting para boolean.

A extração dos dados segue o mesmo padrão para todas as colunas:
from_json(value, 'cliente_id' <tipo do dado>)

A função from_json retorna um Row, por isso precisamos extrair o campo que queremos. O código fica assim:
from_json(value, 'cliente_id' <tipo do dado>)['cliente_id'] as cliente_id

Na consulta abaixo aplicamos esse padrão de extração em todas as colunas de clientes.

In [6]: df = dfu.spark().sql("""
    select
        key
        , from_json(value, 'cliente_id string')['cliente_id'] as cliente_id
        , from_json(value, 'city string')['city'] as city
        , from_json(value, 'state string')['state'] as state
        , from_json(value, 'cnae_id string')['cnae_id'] as cnae_id
        , from_json(value, 'defaulting string')['defaulting'] as defaulting
        , max(timestamp) as timestamp
    from clientes_bronze
    group by 1,2,3,4,5,6
""")

df.printSchema()
df.limit(5)
```

```
root
 |-- key: string (nullable = true)
 |-- cliente_id: string (nullable = true)
 |-- city: string (nullable = true)
 |-- state: string (nullable = true)
 |-- cnae_id: string (nullable = true)
 |-- defaulting: string (nullable = true)
 |-- timestamp: timestamp (nullable = true)

Out[6]: key cliente_id city state cnae_id defaulting timestamp
1128 58820917 CAXINGO PI 4771701 false 2025-10-27 20:47...
1464 21437411 FORTALEZA CE 8112500 false 2025-10-27 20:47...
1578 21418161 FORTALEZA CE 4789004 false 2025-10-27 20:47...
1848 58822609 SANTALUZIA MA 47717-01 false 2025-10-27 20:47...
1973 21436785 EUSEBIO CE 4789004 false 2025-10-27 20:47...
```

Manchete de es... Norms vence no... 18:44 27/10/2025

O trecho abaixo escreve no HDFS em formato delta o resultado da consulta acima

```
In [7]: df.\n    write.\n        mode('overwrite').\n        format('delta').\n        save(f'{wd}/data/clientes-silver')
```

Agora vamos realizar os mesmos passos para os dados de pedidos. A primeira etapa é criar o Dataframe apontando para o diretório no HDFS.

```
In [8]: pedidos = dfu.\n    spark().\n    read.\n        format('delta').\n        load(f'{wd}/data/pedidos-bronze')
```

Agora podemos visualizar como os dados estão armazenados na bronze table.

```
In [9]: pedidos.select('value').limit(1).show(truncate=False)\n+-----\n| value\n+-----\n|>{"client_id": "2149658", "order_amount": 300.06, "order_date": "2019-08-12T19:09:00.000Z", "order_id": "304000110", "salesman_id": "301"}\n+-----
```

O trecho abaixo cria a temp view para executar consultas usando o Spark SQL.

```
In [10]: pedidos.createOrReplaceTempView('pedidos_bronze')
```

E por fim, a consulta de extração/transformação dos dados de pedidos. Veja que estamos transformando a coluna `order_date` para o tipo date e `order_amount` para o tipo float.

```
In [11]: df = dfu.spark().sql("""\n    select
```

```
\n        key,\n        from_json(value, 'client_id string')['client_id'] as client_id,\n        from_json(value, 'order_id string')['order_id'] as order_id,\n        from_json(value, 'order_date date')['order_date'] as order_date,\n        from_json(value, 'order_amount float')['order_amount'] as order_amount,\n        from_json(value, 'salesman_id string')['salesman_id'] as salesman_id\n    from pedidos_bronze\n    """))\n\ndf.printSchema()\n\ndf.limit(5)
```

```
root\n  |-- key: string (nullable = true)\n  |-- client_id: string (nullable = true)\n  |-- order_id: string (nullable = true)\n  |-- order_date: date (nullable = true)\n  |-- order_amount: float (nullable = true)\n  |-- salesman_id: string (nullable = true)
```

O trecho abaixo escreve o resultado da consulta de pedidos no HDFS em formato delta.

```
In [12]: df.\n    write.\n        mode('overwrite').\n        format('delta').\n        save(f'{wd}/data/pedidos-silver')
```

### Exercício

Agora vamos colocar em prática o que aprendemos sobre a extração de dados. Crie um código para extrair os dados de itens de pedidos e realizar as devidas alterações dos tipos de dados. De forma geral, o código deverá realizar as seguintes etapas:

- Criar o Dataframe apontando para o HDFS /data/itens\_pedidos\_bronze

[localhost:8888/notebooks/etl/3-bronze2silver.ipynb](http://localhost:8888/notebooks/etl/3-bronze2silver.ipynb)

jupyter 3-bronze2silver Last Checkpoint 20/10/2025 (autosaved)

File Edit View Insert Cell Kernel Help Trusted Python 3 Logout

### Exercício

Agora vamos colocar em prática o que aprendemos sobre a extração de dados. Crie um código para extrair os dados de itens de pedidos e realizar as devidas alterações dos tipos de dados. De forma geral, o código deverá realizar as seguintes etapas:

1. Criar o Dataframe apontando para o HDFS (ídeia/data/items-bronze)
2. Visualizar como os dados estão armazenados na tabela bronze.
3. Realizar a extração usando o Spark SQL.
4. Escrever os novos dados em ídeia/data/items-silver.

```
In [13]: # Comece criando o dataframe
items = spark.read.format('delta').load(f'{wd}/data/items-bronze')
```

```
In [14]: # Veja como os dados estão armazenados
items.select('value').limit(1).show(truncate=False)
```

```
+-----+-----+
| value |
+-----+
| {"client_id": "2142", "items_count": 2, "list_price": 1.16, "order_date": "2020-03-23T03:00:00.000Z", "order_id": "316023361", "product_id": "35665", "sale_price": 1.1, "salesman_id": "316", "supplier_id": "156"} |
+-----+
```

```
In [15]: # Crie um temp view
items.createOrReplaceTempView('items_bronze')
```

```
In [16]: # Realize a extração usando o Spark SQL
df1 = df1.spark().sql("""
    select
        key
        , from_json(value, 'client_id string')['client_id'] as client_id
        , from_json(value, 'order_id string')['order_id'] as order_id
        , from_json(value, 'order_date date')['order_date'] as order_date
        , from_json(value, 'items_count integer')['items_count'] as items_count
        , from_json(value, 'list_price float')['list_price'] as list_price
        , from_json(value, 'sale_price float')['sale_price'] as sale_price
        , from_json(value, 'salesman_id string')['salesman_id'] as salesman_id
        , from_json(value, 'product_id string')['product_id'] as product_id
        , from_json(value, 'supplier_id string')['supplier_id'] as supplier_id
    from items_bronze
""")
```

Manchete de es...  
Norris vence no... 18:45  
27/10/2025

[localhost:8888/notebooks/etl/3-bronze2silver.ipynb](http://localhost:8888/notebooks/etl/3-bronze2silver.ipynb)

jupyter 3-bronze2silver Last Checkpoint 20/10/2025 (autosaved)

File Edit View Insert Cell Kernel Help Trusted Python 3 Logout

```
select
key
, from_json(value, 'client_id string')['client_id'] as client_id
, from_json(value, 'order_id string')['order_id'] as order_id
, from_json(value, 'order_date date')['order_date'] as order_date
, from_json(value, 'items_count integer')['items_count'] as items_count
, from_json(value, 'list_price float')['list_price'] as list_price
, from_json(value, 'sale_price float')['sale_price'] as sale_price
, from_json(value, 'salesman_id string')['salesman_id'] as salesman_id
, from_json(value, 'product_id string')['product_id'] as product_id
, from_json(value, 'supplier_id string')['supplier_id'] as supplier_id
from items_bronze
df1.printSchema()
df1.limit(5)
```

```
root
--> key: string (nullable = true)
--> client_id: string (nullable = true)
--> order_id: string (nullable = true)
--> order_date: date (nullable = true)
--> items_count: integer (nullable = true)
--> list_price: float (nullable = true)
--> sale_price: float (nullable = true)
--> salesman_id: string (nullable = true)
--> product_id: string (nullable = true)
--> supplier_id: string (nullable = true)
```

```
Out[16]: key client_id order_id order_date items_count list_price sale_price salesman_id product_id supplier_id
362000 2142 316023361 2020-03-23 2 1.16 1.1 316 35665 156
362001 2142 319024848 2020-03-23 1 1.45 1.45 319 19659 186
362002 2142 226034769 2020-03-23 1 49.21 49.21 226 31576 211
362003 2142 460004780 2020-03-23 1 11.17 11.17 400 5791 55
362004 2142 316023363 2020-03-23 1 4.45 4.45 316 19351 186
```

```
In [17]: # Escreva o resultado em /delta/data/items-silver
df1.write.mode('overwrite').format('delta').save(f'{wd}/data/items-silver')
```

Manchete de es...  
Norris vence no... 18:45  
27/10/2025

The screenshot shows a Jupyter Notebook interface with the URL [localhost:8888/tree/etl](http://localhost:8888/tree/etl). The sidebar has tabs for 'Files', 'Running', and 'Clusters'. The main area displays a file tree under the 'etl' directory. A table lists the files with columns for Name, Last Modified, and File size. The files listed are:

| Name                     | Last Modified            | File size |
|--------------------------|--------------------------|-----------|
| poucos segundos atrás    | 2 anos atrás             | 11.5 kB   |
| Running uma hora atrás   | Running 44 minutos atrás | 8.08 kB   |
| Running uma hora atrás   | Running 25 minutos atrás | 21 kB     |
| Running 25 minutos atrás | 7 dias atrás             | 21.6 kB   |
| 86.3 kB                  | um ano atrás             | 9.98 kB   |

At the bottom, there is a status bar with the message "Manchete de es... Norris vence no..." and system icons.

The screenshot shows the AWS S3 console with the URL [localhost:8080/#/main/view/FILES/auto\\_files\\_instance](http://localhost:8080/#/main/view/FILES/auto_files_instance). The left sidebar shows 'Stack and Versions', 'Service Accounts', 'Kerberos', and 'Service Auto Start'. The main area is titled 'Files View' and shows a list of objects in the 'delta' folder. The table includes columns for Name, Size, Last Modified, Owner, Group, Permission, Erasure Coding, and Encrypted. The objects listed are:

| Name            | Size | Last Modified    | Owner | Group | Permission | Erasure Coding | Encrypted |
|-----------------|------|------------------|-------|-------|------------|----------------|-----------|
| items-silver    | --   | 2025-10-27 16:20 | admin | hdFs  | drwxr-xr-x |                | No        |
| pedidos-silver  | --   | 2025-10-27 16:18 | admin | hdFs  | drwxr-xr-x |                | No        |
| clientes-silver | --   | 2025-10-27 16:18 | admin | hdFs  | drwxr-xr-x |                | No        |
| items-bronze    | --   | 2025-10-27 17:56 | admin | hdFs  | drwxr-xr-x |                | No        |
| pedidos-bronze  | --   | 2025-10-27 17:49 | admin | hdFs  | drwxr-xr-x |                | No        |
| clientes-bronze | --   | 2025-10-27 17:48 | admin | hdFs  | drwxr-xr-x |                | No        |
| gold-parquet    | --   | 2025-08-04 16:01 | admin | hdFs  | drwxr-xr-x |                | No        |
| gold            | --   | 2025-08-04 16:00 | admin | hdFs  | drwxr-xr-x |                | No        |

At the bottom, there is a status bar with the message "Manchete de es... Norris vence no..." and system icons.