# Databricks notebook sourcefrom pyspark.sql import DataFrame, functions as F

from pyspark.sql.types import StringType, FloatType

from pyspark.sql.window import Window

import re

# COMMAND ----------

# MAGIC %run ./common\_functions

# COMMAND ----------

def event\_raw\_to\_curated(version, raw\_table\_version=""):

v = add\_underscore(version)

raw\_table = f"pwi\_raw.initial{add\_underscore(raw\_table\_version)}"

df\_event = spark.sql(f"""

select distinct(data\_event\_id) as event\_id,

data\_event\_name as event\_name,

data\_event\_engineProgram as engine\_program

from {raw\_table}

-- where processed = False

""")

df\_event = df\_event.withColumn("load\_timeStamp", F.current\_timestamp()) \

.withColumn("processed", F.lit(False))

# df\_event.display()

save\_table(

df=df\_event,

table\_name="event",

data\_layer="curated",

version=v

)

# COMMAND ----------

def event\_raw\_to\_curated(version, raw\_table\_version=""):

v = add\_underscore(version)

raw\_table = f"pwi\_raw.initial{add\_underscore(raw\_table\_version)}"

df\_event = spark.sql(f"""

select distinct(data\_event\_id) as event\_id,

data\_event\_name as event\_name,

data\_event\_engineProgram as engine\_program

from {raw\_table}

-- where processed = False

""")

df\_event = df\_event.withColumn("load\_timeStamp", F.current\_timestamp()) \

.withColumn("processed", F.lit(False))

# df\_event.display()

save\_table(

df=df\_event,

table\_name="event",

data\_layer="curated",

version=v

)

test\_df =

raw\_table = f"pwi\_raw.initial{add\_underscore(raw\_table\_version)}"

df = spark.read.table(raw\_table)

def raw\_to\_curated\_event(df: DataFrame)

# df\_event = spark.sql(f"""

# select distinct(data\_event\_id) as event\_id,

# data\_event\_name as event\_name,

# data\_event\_engineProgram as engine\_program

# from {raw\_table}

# -- where processed = False

# """)

df\_event = df.select(event\_id, etc.)

df\_event = df\_event.withColumn("load\_timeStamp", F.current\_timestamp()) \

.withColumn("processed", F.lit(False))

return df\_event

# COMMAND ----------

@pytest.fixture

def event\_df():

dictionary = {"row1": ["v1", "v2", "v3"], "row2": ["v4", "v5", "v6"]}

return spark.createDataframe(dictionary)

def test\_event\_raw\_to\_curated():

assert event\_raw\_to\_curated(event\_df()) == expected\_dataframe

# COMMAND ----------

def inspection\_raw\_to\_curated(version, raw\_table\_version=""):

v = add\_underscore(version)

raw\_table = f"pwi\_raw.initial{add\_underscore(raw\_table\_version)}"

df\_inspection = spark.sql(f"""

select distinct(data\_inspection\_uuid) as insp\_uuid,

data\_inspection\_createdAt as insp\_timestamp,

data\_inspection\_name as insp\_name,

data\_inspection\_template\_techData\_id as insp\_description,

data\_inspection\_part\_partNumber as insp\_part\_number,

trim(data\_inspection\_part\_serialNumber) as insp\_serial\_number,

data\_inspection\_part\_batchNumber as insp\_bath\_number,

data\_inspection\_part\_tac as insp\_part\_tac,

data\_inspection\_part\_eot as insp\_part\_eot,

data\_inspection\_part\_efh as insp\_part\_efh,

data\_inspection\_part\_cso as insp\_part\_cso,

data\_inspection\_part\_tso as insp\_part\_tso,

data\_inspection\_part\_engineModel as insp\_engine\_model,

trim(data\_inspection\_part\_engineSerialNumber) as insp\_engine\_serial\_number,

data\_inspection\_disposition\_name as insp\_disposition\_name

from {raw\_table}

""")

df\_inspection = df\_inspection.withColumn(

"insp\_timestamp",

F.to\_timestamp("insp\_timestamp", "M/d[d]/yyyy H[H]:mm")

) \

.withColumn("load\_timeStamp", F.current\_timestamp()) \

.withColumn("processed", F.lit(False))

# df\_inspection.display()

save\_table(

df=df\_inspection,

table\_name="inspection",

data\_layer="curated",

version=v

)

# COMMAND ----------

def observation\_raw\_to\_curated(version, raw\_table\_version=""):

v = add\_underscore(version)

raw\_table = f"pwi\_raw.initial{add\_underscore(raw\_table\_version)}"

df\_observation = spark.sql(f"""

select distinct data\_observation\_uuid as obs\_uuid,

-- not in current data set

-- data\_observation\_inspectionTable as obs\_description,

data\_observation\_areaPath as obs\_distress\_name,

data\_observation\_attributes\_type as obs\_type,

data\_observation\_adHocArea as obs\_adHocArea,

data\_observation\_conditionName as obs\_condition\_name,

data\_observation\_standardConditionName as obs\_condition\_standard\_name,

data\_observation\_disposition as obs\_disposition\_type,

-- cot in current data set

-- data\_observation\_conditionAdHoc as obse\_conditionAdHoc,

data\_observation\_noneObserved as obs\_none\_observed,

-- data\_observation\_attributes\_adHoc as obs\_attribute\_ad\_hoc,

data\_observation\_servicableLimit as obs\_serviceable\_limit\_description,

data\_observation\_repairableLimit as obs\_repairable\_limit\_description

from {raw\_table}

""")

df\_observation = df\_observation.withColumn("load\_timeStamp", F.current\_timestamp()) \

.withColumn("processed", F.lit(False))

# df\_observation.display()

save\_table(

df=df\_observation,

table\_name="observation",

data\_layer="curated",

version=v

)

# COMMAND ----------

def attribute\_raw\_to\_curated(version, raw\_table\_version=""):

v = add\_underscore(version)

raw\_table = f"pwi\_raw.initial{add\_underscore(raw\_table\_version)}"

# The following query returns duplicate records without the group by all columns clause

df\_attribute = spark.sql(f"""

select data\_observation\_uuid as observation\_uuid,

data\_observation\_attributes\_name as attribute\_name,

-- data\_observation\_areaPath as area\_path,

-- regexp\_extract(data\_observation\_attributes\_value, r'^(\d\*\.\d+|\d+)') as attribute\_value,

-- regexp\_extract(data\_observation\_attributes\_value, r'^(?!(\d\*\.\d+|\d+))') as attribute\_comment,

case

when data\_observation\_attributes\_value regexp '^[0-9]\*\.?[0-9]+?' then data\_observation\_attributes\_value

else null

end as value,

case

when data\_observation\_attributes\_value regexp '^[0-9]\*\.?[0-9]+?' then null

else data\_observation\_attributes\_value

end as comment,

data\_observation\_attributes\_adHoc as attribute\_ad\_hoc

from {raw\_table}

group by observation\_uuid,

attribute\_name,

value,

comment,

attribute\_ad\_hoc

""")

df\_attribute = df\_attribute.withColumn("load\_timeStamp", F.current\_timestamp()) \

.withColumn("processed", F.lit(False))

# df\_attribute.display()

save\_table(

df=df\_attribute,

table\_name="attribute",

data\_layer="curated",

version=v

)

# COMMAND ----------

def limit\_raw\_to\_curated(version, raw\_table\_version=""):

v = add\_underscore(version)

raw\_table = f"pwi\_raw.initial{add\_underscore(raw\_table\_version)}"

def extract\_min\_max(limit: str):

\_max = "null"

\_min = "null"

regex = u"([0-9]+\.[0-9]+)\s+inch"

try:

limits = re.findall(regex, limit)

except TypeError:

return

limit = limit.lower()

if "maximum" not in limit and "no longer than" not in limit:

regex = u"([0-9]+\.[0-9]+)(?:\s\*-\s\*([0-9]+\.[0-9]+))?\s+inch"

limits = re.findall(regex, limit)

try:

\_min = limits[0][0]

\_max = limits[0][1]

except IndexError:

pass

else:

\_max = limits[0]

if "minimum" in limit:

\_min = limits[1]

return f"{\_min}, {\_max}"

min\_max\_udf = F.udf(lambda x: extract\_min\_max(x), StringType())

df\_limits = spark.sql("""

with latest\_inspection as (

select

data\_observation\_areaPath,

data\_observation\_conditionName,

max(data\_inspection\_createdAt::timestamp) as latest\_insp

from pwi\_raw.initial\_10\_17

group by 1, 2

),

full\_table\_latest\_inspection as (

-- join cte back to raw table in order to find the servicable limit with an inspection date that matches the latest

select

raw.data\_observation\_areaPath as lim\_distress\_name,

raw.data\_observation\_conditionName as lim\_condition\_name,

raw.data\_inspection\_createdAt::timestamp as lim\_date\_changed,

raw.data\_observation\_servicableLimit as lim\_comment

from pwi\_raw.initial\_10\_17 as raw

left join latest\_inspection

on raw.data\_observation\_areaPath = latest\_inspection.data\_observation\_areaPath

and raw.data\_observation\_conditionName = latest\_inspection.data\_observation\_conditionName

where raw.data\_inspection\_createdAt::timestamp = latest\_inspection.latest\_insp

group by

lim\_distress\_name,

lim\_condition\_name,

lim\_date\_changed,

lim\_comment

),

limits\_w\_count as (

-- add a count of limits (i.e. count of different lim\_comment values) per lim\_distress\_name/lim\_condition\_name/lim\_date\_changed combination

select lim\_distress\_name,

lim\_condition\_name,

lim\_date\_changed,

lim\_comment,

count(\*) over (partition by lim\_distress\_name, lim\_condition\_name, lim\_date\_changed order by lim\_distress\_name) as limit\_count

from full\_table\_latest\_inspection

)

-- for conflicting servicable limits (identified by having limit\_count > 1), exclude those with value of 'not permitted'; this resolves most limit conflicts

select

lim\_distress\_name,

lim\_condition\_name,

lim\_date\_changed,

lim\_comment

from limits\_w\_count

where limit\_count = 1 or (

limit\_count > 1 and lower(lim\_comment) != 'not permitted'

)

order by limit\_count desc, lim\_distress\_name, lim\_condition\_name

""")

df\_limits = df\_limits.withColumn("min\_max", min\_max\_udf(F.col("lim\_comment")))

split\_col = F.split(df\_limits["min\_max"], ", ")

df\_limits = df\_limits.withColumn(

"lim\_minimum",

F.when(split\_col.getItem(0) == "null", None) \

.otherwise(split\_col.getItem(0)) \

.cast(FloatType())

) \

.withColumn(

"lim\_maximum",

F.when(split\_col.getItem(1) == "null", None) \

.otherwise(split\_col.getItem(1)) \

.cast(FloatType())

)

df\_limits = df\_limits.select(

"lim\_distress\_name",

"lim\_condition\_name",

"lim\_minimum",

"lim\_maximum",

"lim\_date\_changed",

"lim\_comment"

)

# df\_limits still contains a few conflicting limits, so we select the lowest lim\_minimum value per lim\_distress\_name/lim\_condition\_name/lim\_date\_changed combination and choose the record with the matching lim\_minimum

window\_spec = Window.partitionBy(

"lim\_distress\_name",

"lim\_condition\_name",

"lim\_date\_changed"

).orderBy("lim\_distress\_name")

df\_limits = df\_limits.withColumn("min\_minimum", F.min("lim\_minimum").over(window\_spec))

df\_filtered = df\_limits.filter((F.col("lim\_minimum").isNull()) | (F.col("lim\_minimum") == F.col("min\_minimum")))

# in at least one case, two conflicting servicable limits have the same lim\_min and lim\_max values; in such cases, we simply pick the first

window\_spec\_1 = Window.partitionBy(

"lim\_distress\_name",

"lim\_condition\_name",

"lim\_date\_changed"

).orderBy("lim\_date\_changed")

df\_filtered = df\_filtered.withColumn("row\_num", F.row\_number().over(window\_spec\_1))

df\_filtered = df\_filtered.filter(F.col("row\_num") == 1).drop("row\_num")

save\_table(

df=df\_filtered,

table\_name="limit",

data\_layer="curated",

version=v

)

# COMMAND ----------

version = ""

raw\_version = "10\_17"

# event\_raw\_to\_curated(version, raw\_version)

inspection\_raw\_to\_curated(version, raw\_version)

# observation\_raw\_to\_curated(version, raw\_version)

# attribute\_raw\_to\_curated(version, raw\_version)

# limit\_raw\_to\_curated(version, raw\_version)

# COMMAND ----------

drop\_table\_delete\_data("pwi\_curated.inspection")

# COMMAND ----------

# MAGIC %sql

# MAGIC

# MAGIC -- query to show Chris on 10/30

# MAGIC

# MAGIC select \*

# MAGIC from pwi\_curated.limit\_10\_25

# MAGIC where lim\_comment != 'Not permitted'

# MAGIC qualify count(\*) over (partition by lim\_distress\_name, lim\_condition\_name, lim\_date\_changed order by lim\_distress\_name) > 1

# COMMAND ----------

# MAGIC %sql

# MAGIC

# MAGIC select lim\_condition\_name

# MAGIC from pwi\_curated.limit

# MAGIC group by 1

# COMMAND ----------

# MAGIC %sql

# MAGIC

# MAGIC select obs\_distress\_name, count(\*)

# MAGIC from pwi\_curated.observation

# MAGIC group by 1

# MAGIC order by 1

# COMMAND ----------

# MAGIC %sql

# MAGIC

# MAGIC select obs\_distress\_name, count(\*)

# MAGIC from pwi\_curated.observation

# MAGIC where obs\_distress\_name

# MAGIC ilike '%ID EDGES%'

# MAGIC group by 1