Obtención Tablas del Bloque 1: Atributos básicos, Nº de cuentas e indicador de nómina en los últimos 12 meses

El código genera, empleando las funciones contenidas en el código funciones_genéricas.ipynb, las distintas tablas del bloque 1:

1. Aux_clientes_contratos
2. Clientes_contratos
3. Num_titulares_cuenta
4. Clientes_Num_Cuentas
5. Aux_movimientos_cuenta
6. Contratos_nomina
7. Clientes_contratos_nomina
8. Contratos_nomina_12m
9. Clientes_nomina_12m
10. Clientes_Atr_Nom_Cuentas

Entorno:

In [4]:

```
import sys, os
import getpass
```

In [5]:

```
## kinit :

pw = getpass.getpass()
!kinit <<< $pw
```

```
........
Password for e056982@CLOUDERAAN.COM:
```

In [6]:

```
sys.path.insert(0, '/opt/cloudera/parcels/SPARK2/lib/spark2/python')
sys.path.insert(0, '/opt/cloudera/parcels/SPARK2/lib/spark2/python/lib/py4j-0.10.7-s



os.environ['PYSPARK_PYTHON'] = '/DYA/da_tech/envs/advice_pfm/bin/python'
os.environ["PYSPARK_DRIVER_PYTHON"] = '/DYA/da_tech/envs/advice_pfm/bin/python'
```

In [7]:

```
import pyspark
import datetime
import time
```

In [9]:

```
from __future__ import division
from dateutil.relativedelta import relativedelta

#proj_path = '/us/xe67625/Proyecto_QbBA/qbba'
#sys.path.insert(0,proj_path)

#from Python.DI_Common_Functions import Functions,logger
#from Python.QbBA_Functions import test
#from conf.proj_paths import *

import pyspark.sql.functions as F
import pyspark.sql.types as Types
from pyspark.sql import HiveContext, Window, Row

from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml import Pipeline

#creamos kerberos
#Functions.check_kerberos()

#creamos la clase test
#Test = test.Test()

CONF = (pyspark.SparkConf() \
            .setAppName(f"advice_pfm_agr_{datetime.datetime.now().strftime('%Y-%m-%d
            )

CONF = CONF \
    .set("spark.rdd.compress", "true") \
    .set("spark.broadcast.compress", "true") \
    .set("spark.shuffle.compress", "true") \
    .set("spark.shuffle.spill.compress", "true") \
    .set("spark.unsafe.sorter.spill.read.ahead.enabled", "false") \
    .set('spark.dynamicAllocation.maxExecutors', 150) \
    .set('spark.dynamicAllocation.minExecutors', 10) \
    .set("spark.executor.memory", "6g") \
    .set("spark.driver.memory", "6g") \
    .set("spark.yarn.queue", "root.DyA.DS") \
    .set("spark.yarn.executor.memoryOverhead", "2g") \
    .set("spark.yarn.driver.memoryOverhead", "2g") \
    .set('spark.dynamicAllocation.enabled', 'true')\
    .set('spark.executor.cores', 3)


from pyspark.sql import SparkSession
sq = SparkSession.builder.config(conf=CONF).enableHiveSupport().getOrCreate()
```

```
  File "<ipython-input-9-870a503b2136>", line 26
    CONF = (pyspark.SparkConf()             .setAppName(f"advice_pfm_a
gr_{datetime.datetime.now().strftime('%Y-%m-%d_%H:%M:%S')}")

^
SyntaxError: invalid syntax
```

Importación de funciones genéricas:

In [ ]:

```
%run funciones_generales.ipynb
```

Inicialización de parámetros:

- partition_ini: fecha relativa a la primera foto del estudio
- partition_fin: fecha relativa a la última foto del estudio
- schema: nombre del esquema donde se guardarán las tablas

In [ ]:

```
partition_ini = '20171201'
partition_fin = '20190930'
schema = "da_financial_health"
```

- partiton_est: fecha relativa a la primera foto necesaria para la construcción de todas las tablas, salvo "aux_clientes_contratos" y "clientes_contratos". Se corresponde con la fecha relativa a 11 meses antes de "partition_ini"
- partition_est_aux: fecha relativa a la primera foto necesaria para la construcción de las tablas "aux_clientes_contratos" y "clientes_contratos". Se corresponde con la fecha relativa a 1 mes antes de "partition_est"

In [ ]:

```
partition_est = fec_n_meses(partition_ini, -11)
partition_est_aux = fec_n_meses(partition_est, -1)
```

- table_list: listado de tablas existentes en el esquema definido en la variable "schema"

In [ ]:

```
table_list=sq.sql(f"""show tables in {schema}""")
```

1. Aux_clientes_contratos:

Creación de la tabla "aux_clientes_contratos". Ésta contiene la siguiente información relativa a los titulares de contratos vivos en la Entidad:

- cod_paisoalf: código del pais originario del contrato
- cod_entalfa: código de la entidad originaria del contrato
- cod_idcontra: código de contrato
- cod_persctpn: código de cliente
- cod_pgccontr: código de contrapartida del contrato
- fec_ininter: fecha de inicio de la relación cliente-contrato
- fec_fin_vin: fecha fin de la relación cliente-contrato

In [ ]:

```
nombre_tabla = "aux_clientes_contratos"
comentario_tabla = "Relacional cliente-contrato"

table_name=table_list.filter(table_list.tableName==nombre_tabla).collect()

if len(table_name)>0:

    print(f"La tabla {nombre_tabla} ya existe")

else:

    query_aux_clientes_contatos = f'''
    create table {schema}.{nombre_tabla} as
    select distinct T_conv.cod_paisoalf, T_conv.cod_entalfa, T_conv.cod_idcontic as
    T.cod_persctpn, T.cod_pgccontr, replace(T.fec_ininter,'-','') as fec_ininter,
    replace(T.fec_fin_vin,'-','') as fec_fin_vin
    from pi_master.pi_vkidspnd as T
    inner join
        (select cod_paisoalf, cod_entalfa,
        case when cod_idcontra is null then cod_idcontic else cod_idcontra end as co
        case when cod_idcontic is null then cod_idcontra else cod_idcontic end as co
        from pi_master.pi_vkidscff) as T_conv
    on T.cod_paisoalf = T_conv.cod_paisoalf and T.cod_entalfa = T_conv.cod_entalfa
    and T.cod_idcontra = T_conv.cod_idcontra
    where cast(replace(T.fec_fin_vin,'-','') as int) >= {partition_est_aux}
    and cast(replace(T.fec_ininter,'-','') as int) <= {partition_fin}
    and cast(replace(T.fec_fin_vin,'-','') as int) >= cast(replace(T.fec_ininter,'-
    and T.cod_ctippe = 'TIT'
    '''
    sq.sql(query_aux_clientes_contatos)
```

2. Clientes_contratos:

Creación de la tabla "clientes_contratos". Ésta contiene la siguiente información relativa a los clientes vivos en la Entidad para cada fecha final de mes comprendida entre las fechas de estudio:

- cod_paisoalf: código del pais originario del contrato
- cod_entalfa: código de la entidad originaria del contrato
- cod_idcontra: código de contrato
- cod_persctpn: código de cliente
- cod_pgccontr_a: conjunto de códigos de contrapartida del contrato (un mismo contrato puede tener varios códigos de contrapartida)
- ind_cuenta: indicador de si el contrato es una cuenta (toma el valor 1 si lo es y 0, en caso contrario)
- fec_ininter: fecha de inicio de la relación cliente-contrato
- fec_fin_vin: fecha fin de la relación cliente-contrato
- fec_contra: fecha de formalización del contrato
- cod_situdw_a: conjunto de estados del contrato (A: activo, I: inactivo, C: cancelado o M: moroso)
- fec_vencimiento: fecha de vencimiento del contrato
- fam_prod: familia de productos a la que pertenece el contrato
- partition_id: fecha de la que data la información

In [ ]:

```
nombre_tabla = "clientes_contratos"
comentario_tabla = "Relacional cliente-contrato"

table_name=table_list.filter(table_list.tableName==nombre_tabla).collect()

if len(table_name)>0:

    print(f"La tabla {nombre_tabla} ya existe")

else:

    dic_variables = {"cod_paisoalf": "string",
                     "cod_entalfa": "string",
                     "cod_idcontra": "string",
                     "cod_persctpn": "string",
                     "cod_pgccontr_a": "array<string>",
                     "ind_cuenta": "int",
                     "fec_ininter": "string",
                     "fec_fin_vin": "string",
                     "fec_contra": "string",
                     "cod_situdw_a": "string",
                     "fec_vencimiento": "string",
                     "fam_prod": "string"}

    crear_tabla (schema, nombre_tabla, comentario_tabla, "partition_id", **dic_varia
```

Creación e inserción de la información relativa a fecha final de mes comprendida entre las fechas inicial y final de estudio:

In [ ]:

```
#CREAR QUERY ALTERNATIVA CON SPARK PARA EVITAR REPETIR EL ARRAY_CONTAINS.....
```

In [ ]:

```
# schema : 0, nombre_tabla: 1, p_ini: 2, particion: 3
query_clientes_contatos = '''
insert overwrite table {0}.{1} partition (partition_id={3})
select cl_cont.cod_paisoalf, cl_cont.cod_entalfa, cl_cont.cod_idcontra,
cl_cont.cod_persctpn, cl_cont.cod_pgccontr_a,
case when
    array_contains(cl_cont.cod_pgccontr_a,'0000') or
    array_contains(cl_cont.cod_pgccontr_a,'0001') or
    array_contains(cl_cont.cod_pgccontr_a,'0005') or
    array_contains(cl_cont.cod_pgccontr_a,'0006') or
    array_contains(cl_cont.cod_pgccontr_a,'0007') or
    array_contains(cl_cont.cod_pgccontr_a,'0009') or
    array_contains(cl_cont.cod_pgccontr_a,'0020') or
    array_contains(cl_cont.cod_pgccontr_a,'0021') or
    array_contains(cl_cont.cod_pgccontr_a,'0029') or
    array_contains(cl_cont.cod_pgccontr_a,'0030') then 1
    else 0
end as ind_cuenta,
cl_cont.fec_ininter, cl_cont.fec_fin_vin,
cont.fec_contra, cont.cod_situdw_a, cont.fec_vencimiento,
case
    when array_contains(fam_prod_a, 'Cuenta')                                   the
    when array_contains(fam_prod_a, 'Tarjetas. Wallet')                         the
    when array_contains(fam_prod_a, 'Tarjetas. Prepago')                        the
    when array_contains(fam_prod_a, 'Tarjetas. Débito')                         the
    when array_contains(fam_prod_a, 'Tarjetas. Crédito')                        the
    when array_contains(fam_prod_a, 'Tarjetas. Privada')                        the
    when array_contains(fam_prod_a, 'Tarjetas. Empresas')                       the
    when array_contains(fam_prod_a, 'Nómina o pensión. Pensión')                the
    when array_contains(fam_prod_a, 'Nómina o pensión. Nómina')                 the
    when array_contains(fam_prod_a, 'Domiciliación Recibos')                    the
    when array_contains(fam_prod_a, 'Hipoteca')                                 the
    when array_contains(fam_prod_a, 'Préstamos. Preconcedido')                  the
    when array_contains(fam_prod_a, 'Préstamos. Coche')                         the
    when array_contains(fam_prod_a, 'Préstamos. Personal')                      the
    when array_contains(fam_prod_a, 'Depósitos')                                the
    when array_contains(fam_prod_a, 'Fondos de inversión')                      the
    when array_contains(fam_prod_a, 'Planes de pensiones')                      the
    when array_contains(fam_prod_a, 'Planes de pensiones. PAS')                 the
    when array_contains(fam_prod_a, 'Planes de pensiones. PPA')                 the
    when array_contains(fam_prod_a, 'PIAS')                                     the
    when array_contains(fam_prod_a, 'Valores')                                  the
    when array_contains(fam_prod_a, 'Seguros. Vida')                            the
    when array_contains(fam_prod_a, 'Seguros. Salud')                           the
    when array_contains(fam_prod_a, 'Seguros. Vivienda')                        the
    when array_contains(fam_prod_a, 'Seguros. Coche o moto')                    the
    when array_contains(fam_prod_a, 'Seguros. Miniseguros')                     the
    when array_contains(fam_prod_a, 'Seguros. Otros')                           the
    when array_contains(fam_prod_a, 'Seguros. Empresas')                        the
    when array_contains(fam_prod_a, 'Préstamos. Empresas')                      the
    when array_contains(fam_prod_a, 'Empresas')                                 the
    when array_contains(fam_prod_a, 'Cuenta. Otros epigrafes')                  the
    when array_contains(fam_prod_a, 'Tarjetas. Otros epigrafes')                the
    when array_contains(fam_prod_a, 'Tarjetas. Privada. Otros epigrafes')       the
    when array_contains(fam_prod_a, 'Nómina o pensión. Otros epigrafes')        the
    when array_contains(fam_prod_a, 'Domiciliación Recibos. Otros epigrafes')   the
    when array_contains(fam_prod_a, 'Hipoteca. Otros epigrafes')                the
    when array_contains(fam_prod_a, 'Depósitos. Otros epigrafes')               the
    when array_contains(fam_prod_a, 'Fondos de inversión. Otros epigrafes')     the
```

```
    when array_contains(fam_prod_a, 'Planes de pensiones. Otros epigrafes')      the
    when array_contains(fam_prod_a, 'Valores. Otros epigrafes')                  the
    when array_contains(fam_prod_a, 'Seguros. Otros epigrafes')                  the
    when array_contains(fam_prod_a, 'Empresas. Otros epigrafes')                 the
    when array_contains(fam_prod_a, 'Préstamos/Empresas. Otros epigrafes')       the
    when array_contains(fam_prod_a, 'Tarjetas. Contrato')                        the
    else 'Otros'
end as fam_prod
from
(select cod_paisoalf, cod_entalfa, cod_idcontra, cod_persctpn, collect_set(cod_pgcc
cast(min(cast(fec_ininter as int)) as string) as fec_ininter,
cast(max(cast(fec_fin_vin as int)) as string) as fec_fin_vin
from {0}.aux_clientes_contratos
where cast(replace(fec_fin_vin,'-','') as int) >= {2} and cast(replace(fec_ininter,
and cast(replace(fec_fin_vin,'-','') as int) >= cast(replace(fec_ininter,'-','') as
group by cod_paisoalf, cod_entalfa, cod_idcontra, cod_persctpn) as cl_cont
left join
(select cod_persona, cod_idcontra, min(fec_contra) as fec_contra, collect_set(cod_s
min(fec_vencimiento) as fec_vencimiento,
collect_set(fam_prod_a) as fam_prod_a
from (select cod_idcontra, cod_persona, fec_contra, cod_situdw_a, fec_vencimiento,
case
    when array_contains(fam_prod_a, 'Cuenta')                                    the
    when array_contains(fam_prod_a, 'Tarjetas. Wallet')                          the
    when array_contains(fam_prod_a, 'Tarjetas. Prepago')                         the
    when array_contains(fam_prod_a, 'Tarjetas. Débito')                          the
    when array_contains(fam_prod_a, 'Tarjetas. Crédito')                         the
    when array_contains(fam_prod_a, 'Tarjetas. Privada')                         the
    when array_contains(fam_prod_a, 'Tarjetas. Empresas')                        the
    when array_contains(fam_prod_a, 'Nómina o pensión. Pensión')                 the
    when array_contains(fam_prod_a, 'Nómina o pensión. Nómina')                  the
    when array_contains(fam_prod_a, 'Domiciliación Recibos')                     the
    when array_contains(fam_prod_a, 'Hipoteca')                                  the
    when array_contains(fam_prod_a, 'Préstamos. Preconcedido')                   the
    when array_contains(fam_prod_a, 'Préstamos. Coche')                          the
    when array_contains(fam_prod_a, 'Préstamos. Personal')                       the
    when array_contains(fam_prod_a, 'Depósitos')                                 the
    when array_contains(fam_prod_a, 'Fondos de inversión')                       the
    when array_contains(fam_prod_a, 'Planes de pensiones')                       the
    when array_contains(fam_prod_a, 'Planes de pensiones. PAS')                  the
    when array_contains(fam_prod_a, 'Planes de pensiones. PPA')                  the
    when array_contains(fam_prod_a, 'PIAS')                                      the
    when array_contains(fam_prod_a, 'Valores')                                   the
    when array_contains(fam_prod_a, 'Seguros. Vida')                             the
    when array_contains(fam_prod_a, 'Seguros. Salud')                            the
    when array_contains(fam_prod_a, 'Seguros. Vivienda')                         the
    when array_contains(fam_prod_a, 'Seguros. Coche o moto')                     the
    when array_contains(fam_prod_a, 'Seguros. Miniseguros')                      the
    when array_contains(fam_prod_a, 'Seguros. Otros')                            the
    when array_contains(fam_prod_a, 'Seguros. Empresas')                         the
    when array_contains(fam_prod_a, 'Préstamos. Empresas')                       the
    when array_contains(fam_prod_a, 'Empresas')                                  the
    when array_contains(fam_prod_a, 'Cuenta. Otros epigrafes')                   the
    when array_contains(fam_prod_a, 'Tarjetas. Otros epigrafes')                 the
    when array_contains(fam_prod_a, 'Tarjetas. Privada. Otros epigrafes')        the
    when array_contains(fam_prod_a, 'Nómina o pensión. Otros epigrafes')         the
    when array_contains(fam_prod_a, 'Domiciliación Recibos. Otros epigrafes')    the
    when array_contains(fam_prod_a, 'Hipoteca. Otros epigrafes')                 the
    when array_contains(fam_prod_a, 'Depósitos. Otros epigrafes')                the
    when array_contains(fam_prod_a, 'Fondos de inversión. Otros epigrafes')      the
    when array_contains(fam_prod_a, 'Planes de pensiones. Otros epigrafes')      the
```

```
    when array_contains(fam_prod_a, 'Valores. Otros epigrafes')                  the
    when array_contains(fam_prod_a, 'Seguros. Otros epigrafes')                  the
    when array_contains(fam_prod_a, 'Empresas. Otros epigrafes')                 the
    when array_contains(fam_prod_a, 'Préstamos/Empresas. Otros epigrafes')       the
    when array_contains(fam_prod_a, 'Tarjetas. Contrato')                        the
    else 'Otros'
end as fam_prod_a
from di_qbba.cli_corp_cont_productos_iuc where partition_id = {3}) as cont_aux
group by cod_persona, cod_idcontra) as cont
on cl_cont.cod_idcontra=cont.cod_idcontra and cl_cont.cod_persctpn=cont.cod_persona
where (cast(replace(cont.fec_vencimiento, "-", "") as int) >= {2} or cont.fec_vencir
cast(replace(cont.fec_contra,"-","") as int) <= {3}
'''

generar_particion (schema, nombre_tabla, partition_est_aux, partition_fin, "partitic
```

3. Num_titulares_cuenta:

Creación de la tabla "num_titulares_cuenta". Ésta contiene la siguiente información relativa al número de titulares de las cuentas vivas en la Entidad para cada fecha final de mes comprendida entre las fechas de estudio:

- cod_paisoalf: código del pais originario del contrato
- cod_entalfa: código de la entidad originaria del contrato
- cod_idcontra: código de contrato
- num_titulares: número de titulares de la cuenta
- num_cuentas_indiv: indicador de cuenta individual (toma valor 1 si sólo tiene un titular y 0, en caso contrario)
- num_cuentas_comp: indicador de cuenta compartida (toma valor 1 si tiene más de un titular y 0, en otro caso)
- partition_id: fecha de la que data la información

In [ ]:

```
nombre_tabla = "num_titulares_cuenta"
comentario_tabla = "Número de titulares por cuenta"

table_name=table_list.filter(table_list.tableName==nombre_tabla).collect()

if len(table_name)>0:

    print(f"La tabla {nombre_tabla} ya existe")

else:

    dic_variables = { "cod_paisoalf": "string",
                      "cod_entalfa": "string",
                      "cod_idcontra": "string",
                      "num_titulares": "int",
                      "num_cuentas_indiv": "int",
                      "num_cuentas_comp": "int"}

    crear_tabla (schema, nombre_tabla, comentario_tabla, "partition_id", **dic_varia
```

Generación de las distintas particiones e inserción de las mismas:

In [ ]:

```
# schema : 0, nombre_tabla: 1, p_ini: 2 (no empleado), particion: 3
query_titulares = '''
insert overwrite table {0}.{1} partition (partition_id={3})
select cod_paisoalf, cod_entalfa, cod_idcontra, count(distinct cod_persctpn) as num_
case when count(distinct cod_persctpn) = 1 then 1 else 0 end as num_cuentas_indiv,
case when count(distinct cod_persctpn) > 1 then 1 else 0 end as num_cuentas_comp
from {0}.clientes_contratos
where partition_id = {3}
and ind_cuenta = 1
group by cod_paisoalf, cod_entalfa, cod_idcontra
'''

generar_particion (schema, nombre_tabla, partition_ini, partition_fin, "partition_id
```

4. Clientes_Num_Cuentas:

Creación de la tabla que contiene el número de cuentas individuales y compartidas que tiene cada titular:

In [ ]:

```
nombre_tabla = "clientes_num_cuentas"
comentario_tabla = "Número de cuentas individuales y compartidas"

table_name=table_list.filter(table_list.tableName==nombre_tabla).collect()

if len(table_name)>0:

    print(f"La tabla {nombre_tabla} ya existe")

else:

    dic_variables = { "cod_paisoalf": "string",
                      "cod_entalfa": "string",
                      'cod_persctpn': "string",
                      "num_cuentas_indiv": "string",
                      "num_cuentas_comp": "string"}

    crear_tabla (schema, nombre_tabla, comentario_tabla, "partition_id", **dic_varia
```

Generación de las distintas particiones e inserción de las mismas:

In [ ]:

```
for particion in partition_list(last_day_of_month(partition_ini), partition_fin, Fal

    texto = 'Ejecutando la particion {0} de {1}'.format(particion, nombre_tabla)

    print(texto)

    if len(table_name)>0:

        if sq.sql(f'''select count(*) as N from {schema}.{nombre_tabla} where partit
        .toPandas()['N'].tolist()[0] > 0:

            texto = "La partición {0} de {1} ya existe".format(particion, nombre_tab

            print(texto)

            no_existe = 0

        else:

            no_existe = 1
    else:

        no_existe = 1

    if no_existe == 1:

        num_titulares_cuenta = sq.sql(f'''select * from {schema}.num_titulares_cuent

        clientes_contratos = sq.sql(f'''select * from {schema}.clientes_contratos wh

        aux_clientes = num_titulares_cuenta.join(clientes_contratos.select('cod_pais
                                                 'cod_persctpn','partition_id'), on=[
                                                 'partition_id'])

        clientes_num_cuentas = aux_clientes.groupBy("cod_paisoalf", "cod_entalfa", '
            .agg(F.sum("num_cuentas_indiv").alias("num_cuentas_indiv"), F.sum("num_c

        clientes_num_cuentas.write.mode("append").partitionBy("partition_id").saveAs

        texto = "Ejecutada la particion {0} de {1}".format(particion, nombre_tabla)

        print(texto)
```

5. Aux_movimientos_cuenta:

Creación de la tabla que contiene una extracción de los movimientos de cuenta:

In [ ]:

```
nombre_tabla = "aux_movimientos_cuenta"
comentario_tabla = "Movimientos de cuenta"

table_name=table_list.filter(table_list.tableName==nombre_tabla).collect()

if len(table_name)>0:

    print(f"La tabla {nombre_tabla} ya existe")

else:

    dic_variables = { "cod_paisoalf": "string",
                      "cod_entalfa": "string",
                      "cod_idcontra": "string",
                      "fec_movim": "string",
                      "fec_movimien": "string",
                      "imp_mov": "decimal(19,2)",
                      "des_concepto" : "string",
                      "cod_talon": "int",
                      "id_subcategoria": "int"}

    crear_tabla (schema, nombre_tabla, comentario_tabla, "partition_id", **dic_varia
```

Generación de las distintas particiones e inserción de las mismas:

In [ ]:

```
'''
-- Traspasos codificados como id_subcategoria = 121212
-- Gastos de tarjeta de crédito (cod_talon = 120) -> id_subcategoria = 121213
-- Ingresos financiados de tarjeta de crédito (cod_talon = 320) -> id_subcategoria =
-- Gastos financiados de tarjeta de crédito (cod_talon = 620) -> id_subcategoria = 1
-- Gastos en hipoteca (pago mensual) -> id_subcategoria = 121216
-- NO CATEGORIZABLE: Movimientos tarjeta prepago, crédito y traspasos.
'''

# schema : 0, nombre_tabla: 1, p_ini: 2, particion: 3
query_aux_mov = '''
insert overwrite table {0}.{1} partition (partition_id={3})
select cod_paisoalf, cod_entalfa, cod_idcontra,
concat(substr(fec_movimien,1,4), substr(fec_movimien,6,2), substr(fec_movimien,9,2))
fec_movimien, imp_mov, des_concepto, cod_talon,
case when upper(des_concepto) like "%TRASP%" and id_subcategoria_base = 0 then 12121
when cod_talon = 120 then 121213
when cod_talon = 320 then 121214
when cod_talon = 620 then 121215
when cod_talon = 379 and id_subcategoria_base = 76 then 121216
else id_subcategoria_base
end as id_subcategoria
from custom_data.movimientos_cuentas_pfm
where cod_divisa = 'EUR' and
cod_pgccontr in ('0000', '0001', '0005', '0006', '0007', '0009', '0020', '0021', '00
and cast(partition_id as int) >= {2} and cast(partition_id as int) <= {3}
and cast(concat(substr(fec_movimien,1,4), substr(fec_movimien,6,2), substr(fec_movir
and cast(concat(substr(fec_movimien,1,4), substr(fec_movimien,6,2), substr(fec_movir
'''

generar_particion (schema, nombre_tabla, partition_est, partition_fin, "partition_id
```

6. Contratos_nomina:

Creación de la tabla que contiene las nóminas cobradas mensualmente en cada contrato:

In [ ]:

```
nombre_tabla = "contratos_nomina"
comentario_tabla = "Relacional contrato-nómina"

table_name=table_list.filter(table_list.tableName==nombre_tabla).collect()

if len(table_name)>0:

    print(f"La tabla {nombre_tabla} ya existe")

else:

    dic_variables = { "cod_paisoalf": "string",
                      "cod_entalfa": "string",
                      "cod_idcontra": "string",
                      "fec_movim": "string",
                      "fecha_mensual": "string"}

    crear_tabla (schema, nombre_tabla, comentario_tabla, "partition_id", **dic_varia
```

Generación de las distintas particiones e inserción de las mismas:

In [ ]:

```
for particion in partition_list(last_day_of_month(partition_est), partition_fin, Fal

    texto = 'Ejecutando la particion {0} de {1}'.format(particion, nombre_tabla)

    print(texto)

    if len(table_name)>0:

        if sq.sql(f'''select count(*) as N from {schema}.{nombre_tabla} where partit
        .toPandas()['N'].tolist()[0] > 0:

            texto = "La partición {0} de {1} ya existe".format(particion, nombre_tab

            print(texto)

            no_existe = 0

        else:

            no_existe = 1
    else:

        no_existe = 1

    if no_existe == 1:

        cont_nominas = sq.sql(f'''select cod_paisoalf, cod_entalfa, cod_idcontra, fe
        substr(fec_movim,1,6) as fecha_mensual,
        partition_id
        from {schema}.aux_movimientos_cuenta
        where id_subcategoria = 41 and imp_mov > 0
        and partition_id = {particion}
        ''')

        cont_nominas.write.mode("append").partitionBy("partition_id").saveAsTable(f'

        texto = "Ejecutada la particion {0} de {1}".format(particion, nombre_tabla)

        print(texto)
```

7. Clientes_contratos_nomina:

Creación de la tabla que contiene las nóminas cobradas mensualmente en cada contrato, incluyéndose sus titulares:

In [ ]:

```
nombre_tabla = "clientes_contratos_nomina"
comentario_tabla = "Relacional cliente-contrato-nómina"

table_name=table_list.filter(table_list.tableName==nombre_tabla).collect()

if len(table_name)>0:

    print(f"La tabla {nombre_tabla} ya existe")

else:

    dic_variables = { "cod_paisoalf": "string",
                      "cod_entalfa": "string",
                      'cod_persctpn': "string",
                      "cod_idcontra": "string",
                      "fec_movim": "string",
                      "fecha_mensual": "string"}

    crear_tabla (schema, nombre_tabla, comentario_tabla, "partition_id", **dic_varia
```

Generación de las distintas particiones e inserción de las mismas:

In [ ]:

```
for particion in partition_list(last_day_of_month(partition_est), partition_fin, Fal

    texto = 'Ejecutando la particion {0} de {1}'.format(particion, nombre_tabla)

    print(texto)

    if len(table_name)>0:

        if sq.sql(f'''select count(*) as N from {schema}.{nombre_tabla} where partit
        .toPandas()['N'].tolist()[0] > 0:

            texto = "La partición {0} de {1} ya existe".format(particion, nombre_tab

            print(texto)

            no_existe = 0

        else:

            no_existe = 1
    else:

        no_existe = 1

    if no_existe == 1:

        clientes_contratos = sq.sql(f'''select * from {schema}.clientes_contratos wh

        cont_nominas = sq.sql(f'''select * from {schema}.contratos_nomina where part

        cliente_cont_nominas = clientes_contratos.select('cod_paisoalf','cod_entalfa
                                                         'partition_id').join(cont_nomina
                                                         'cod_entalfa','cod_idcontra', 'p

        cliente_cont_nominas = cliente_cont_nominas.sort('cod_paisoalf', 'cod_entalf

        cliente_cont_nominas.write.mode("append").partitionBy("partition_id")\
                            .saveAsTable(f"{schema}.clientes_contratos_nomina")

        texto = "Ejecutada la particion {0} de {1}".format(particion, nombre_tabla)

        print(texto)
```

8. Contratos_nomina_12m:

Creación de la tabla que contiene el número de nóminas cobradas en los últimos 12 meses a nivel contrato:

In [ ]:

```
#### Creación de la tabla que contiene las nóminas cobradas mensualmente en cada co
nombre_tabla = "contratos_nomina_12m"
comentario_tabla = "Indicador de si el cliente ha cobrado a nivel contrato una nómi

table_name=table_list.filter(table_list.tableName==nombre_tabla).collect()

for particion in partition_list(last_day_of_month(partition_ini), partition_fin, Fal

    if anyo_bisiesto(int(particion[0:4])) and particion[4:6] == '02':

        particion_ant = str(int(particion[0:4])-1) + '0229'

    else:

        particion_ant = str(int(particion[0:4])-1) + particion[4:]

    texto = 'Ejecutando la particion {0} de {1}'.format(particion, nombre_tabla)

    print(texto)

    if len(table_name)>0:

        if sq.sql(f'''select count(*) as N from {schema}.{nombre_tabla} where partit
        .toPandas()['N'].tolist()[0] > 0:

            texto = "La partición {0} de {1} ya existe".format(particion, nombre_tab

            print(texto)

            no_existe = 0

        else:

            no_existe = 1
    else:

        no_existe = 1

    if no_existe == 1:

        if 'cliente_cont_nominas' not in globals():

            cliente_cont_nominas = sq.sql(f"select * from {schema}.clientes_contrato

        aux_cont_nomina_12m = cliente_cont_nominas.filter(f'''cast(partition_id as i
                        and cast(partition_id as int) <= {particion}''')

        aux_cont_nomina =  aux_cont_nomina_12m.groupby('cod_paisoalf','cod_entalfa',
                        'fecha_mensual').agg(F.count('cod_idcontra').alias('Num_nomi
                        'cod_entalfa','cod_persctpn', 'cod_idcontra', 'fecha_mensual

        aux_cont_nomina_mes = aux_cont_nomina.groupby('cod_paisoalf','cod_entalfa',
                                        .agg(F.sum('Num_nominas').alias('Num_nominas
                                             F.countDistinct('fecha_mensual').alias(

        if int(particion[4:6])%2 == 1:

            expr_agrup_bimes = '''
            floor(cast(substr(fecha_mensual, 5, 2) as int)/2)%6
```

```
    '''

else:

    expr_agrup_bimes = '''
    ceil(cast(substr(fecha_mensual,5,2) as int)/2)
    '''

aux_cont_nomina_bimes = aux_cont_nomina.withColumn('agrupacion_bimes', F.exp

aux_cont_nomina_bimes = aux_cont_nomina_bimes.groupby('cod_paisoalf','cod_en
                    'agrupacion_bimes').agg(F.sum('Num_nominas').alias('Num_

aux_cont_nomina_bimes = aux_cont_nomina_bimes.filter('Num_nominas_bimes > 1
                                                    .groupby('cod_paisoalf',
                                                    .agg(F.count('Num_nominas

contratos_nomina_12m = aux_cont_nomina_mes.join(aux_cont_nomina_bimes,\
                    on = ['cod_paisoalf','cod_entalfa','cod_idcontra', 'cod_

expr_mes = 'case when Num_nominas_mes > 11 then 1 else 0 end'
expr_bimes = 'case when Num_nominas_bimes > 5 then 1 else 0 end'

contratos_nomina_12m = contratos_nomina_12m.withColumn('Ind_nomina_mes', F.e

contratos_nomina_12m = contratos_nomina_12m.withColumn('Ind_nomina_bimes', I

contratos_nomina_12m = contratos_nomina_12m.withColumn("partition_id", F.exp

contratos_nomina_12m = contratos_nomina_12m.select('cod_paisoalf','cod_ental
                                                    'Num_nominas_tot', 'Num_n
                                                    'Num_nominas_bimes', 'Ind
#insertInto
contratos_nomina_12m.write.mode("append").partitionBy("partition_id").saveAs

texto = "Ejecutada la particion {0} de {1}".format(particion, nombre_tabla)

print(texto)
```

9. Clientes_nomina_12m:

Creación de la tabla que contiene el número de nóminas cobradas en los últimos 12 meses a nivel cliente:

In [ ]:

```
nombre_tabla = "clientes_nomina_12m"
comentario_tabla = "Indicador de si el cliente ha cobrado una nómina al mes en los ú

table_name=table_list.filter(table_list.tableName==nombre_tabla).collect()

for particion in partition_list(last_day_of_month(partition_ini), partition_fin, Fal

    if anyo_bisiesto(int(particion[0:4])) and particion[4:6] == '02':

        particion_ant = str(int(particion[0:4])-1) + '0229'

    else:

        particion_ant = str(int(particion[0:4])-1) + particion[4:]

    texto = 'Ejecutando la particion {0} de {1}'.format(particion, nombre_tabla)

    print(texto)

    if len(table_name)>0:

        if sq.sql(f'''select count(*) as N from {schema}.{nombre_tabla} where partit
        .toPandas()['N'].tolist()[0] > 0:

            texto = "La partición {0} de {1} ya existe".format(particion, nombre_tab

            print(texto)

            no_existe = 0

        else:

            no_existe = 1
    else:

        no_existe = 1

    if no_existe == 1:

        if 'cliente_cont_nominas' not in globals():

            cliente_cont_nominas = sq.sql(f"select * from {schema}.clientes_contrato

        aux_clientes_nomina_12m = cliente_cont_nominas.filter(f'''cast(partition_id
                        and cast(partition_id as int) <= {particion}''')

        aux_clientes_nomina =  aux_clientes_nomina_12m.groupby('cod_paisoalf','cod_e
                        'fecha_mensual').agg(F.count('cod_persctpn').alias('Num_nomi
                        'cod_entalfa','cod_persctpn', 'fecha_mensual')

        aux_clientes_nomina_mes = aux_clientes_nomina.groupby('cod_paisoalf','cod_en
                                        .agg(F.sum('Num_nominas').alias('Num_nominas
                                             F.countDistinct('fecha_mensual').alias(

        if int(particion[4:6])%2 == 1:

            expr_agrup_bimes = '''
            case when substr(fecha_mensual,5,2)='12' then
            floor(cast(substr(fecha_mensual, 5, 2) as int)/2) - 6
```

```
        else floor(cast(substr(fecha_mensual, 5, 2) as int)/2) end'''

    else:

        expr_agrup_bimes = '''
        ceil(cast(substr(fecha_mensual,5,2) as int)/2)'''

    aux_clientes_nomina_bimes = aux_clientes_nomina.withColumn('agrupacion_bimes

    aux_clientes_nomina_bimes = aux_clientes_nomina_bimes.groupby('cod_paisoalf
                        'agrupacion_bimes').agg(F.sum('Num_nominas').alias('Num_

    aux_clientes_nomina_bimes = aux_clientes_nomina_bimes.filter('Num_nominas_bi
                                                        .groupby('cod_paisoalf',
                                                        .agg(F.count('Num_nominas

    clientes_nomina_12m = aux_clientes_nomina_mes.join(aux_clientes_nomina_bimes
                        on = ['cod_paisoalf','cod_entalfa', 'cod_persctpn'], hov

    expr_mes = 'case when Num_nominas_mes > 11 then 1 else 0 end'
    expr_bimes = 'case when Num_nominas_bimes > 5 then 1 else 0 end'

    clientes_nomina_12m = clientes_nomina_12m.withColumn('Ind_nomina_mes', F.exp

    clientes_nomina_12m = clientes_nomina_12m.withColumn('Ind_nomina_bimes', F.e

    clientes_nomina_12m = clientes_nomina_12m.withColumn("partition_id", F.expr(

    clientes_nomina_12m = clientes_nomina_12m.select('cod_paisoalf','cod_entalfa
                                                    'Num_nominas_mes', 'Ind_nomi
                                                     'Ind_nomina_bimes', 'partit
    #insertInto
    clientes_nomina_12m.write.mode("append").partitionBy("partition_id").saveAsT

    texto = "Ejecutada la particion {0} de {1}".format(particion, nombre_tabla)

    print(texto)
```

10. Clientes_Atr_Nom_Cuentas:

Creación de la tabla que contiene los atributos, número de nóminas cobradas en los últimos 12 meses y número de cuentas individuales y compartidas del cliente:

In [ ]:

```
nombre_tabla = "clientes_atr_nom_cuentas"
comentario_tabla = "Relacional cliente-atributos-nominas-cuentas"

table_name=table_list.filter(table_list.tableName==nombre_tabla).collect()

if len(table_name)>0:

    print(f"La tabla {nombre_tabla} ya existe")

else:

    dic_variables = {"cod_paisoalf": "string",
                     "cod_entalfa": "string",
                     "cod_persctpn": "string",
                     "cod_situdw": "string",
                     "edad": "int",
                     "cod_segmento_global" : "int",
                     "des_segmento_global" : "string",
                     "cod_segmento_plan_uno" : "string",
                     "des_segmento_plan_uno" : "string",
                     "tipo_persona" : "string",
                     "xti_empleado" : "tinyint",
                     "num_cuentas_indiv" : "int",
                     "num_cuentas_comp" : "int",
                     "Num_nominas_tot" : "int",
                     "Num_nominas_mes" : "int",
                     "Ind_nomina_mes" : "int",
                     "Num_nominas_bimes" : "int",
                     "Ind_nomina_bimes" : "int"
                     }

    crear_tabla (schema, nombre_tabla, comentario_tabla, "partition_id", **dic_varia
```

Generación de las distintas particiones e inserción de las mismas:

In [ ]:

```
for particion in partition_list(last_day_of_month(partition_ini), partition_fin, Fal

    texto = 'Ejecutando la particion {0} de {1}'.format(particion, nombre_tabla)

    print(texto)

    if len(table_name)>0:

        if sq.sql(f'''select count(*) as N from {schema}.{nombre_tabla} where partit
        .toPandas()['N'].tolist()[0] > 0:

            texto = "La partición {0} de {1} ya existe".format(particion, nombre_tab

            print(texto)

            no_existe = 0

        else:

            no_existe = 1
    else:

        no_existe = 1

    if no_existe == 1:

        query_atributos = f'''
        select distinct cod_persona as cod_persctpn, cod_situdw, edad, cod_segmento_
        cod_segmento_plan_uno, des_segmento_plan_uno, tipo_persona, xti_empleado, pa
        from di_qbba.atributos_basicos
        where cast(partition_id as int) = {particion}
        '''

        atributos = sq.sql(query_atributos)

        query_catalogo = f'''
        select cod_clavecdt as cod_segmento_global, des_valorcon as des_segmento_glo
        from pi_master.pi_vkidszva4010010
        where cast(partition_id as int) = {particion}
        '''

        catalogo_segm = sq.sql(query_catalogo)

        atributos = atributos.join(catalogo_segm, on = ['cod_segmento_global', 'part
                .select("cod_persctpn", "cod_situdw", "edad", "cod_segmento_global",
                      "cod_segmento_plan_uno", "des_segmento_plan_uno", "tipo_perso

        if 'clientes_contratos' not in globals():

            clientes_contratos = sq.sql(f'''select * from {schema}.clientes_contrato

        clientes = clientes_contratos.filter(f'cast(partition_id as int) = {particio
        .select('cod_paisoalf','cod_entalfa','cod_persctpn','partition_id').distinct

        clientes_atr_nom_cuentas = clientes.join(atributos, on = ['cod_persctpn','pa
                                            how = 'left')

        if 'clientes_num_cuentas' not in globals():
```

```
        clientes_num_cuentas = sq.sql(f'''select * from {schema}.clientes_num_cu

    clientes_atr_nom_cuentas = clientes_atr_nom_cuentas.filter(f"cast(partition_
                               .join(clientes_num_cuentas, on = ['cod_paisoalf
                                                           'cod_persctpn', 'pa

    if 'clientes_nomina_12m' not in globals():

        clientes_nomina_12m = sq.sql(f'''select * from {schema}.clientes_nomina_

    clientes_atr_nom_cuentas = clientes_atr_nom_cuentas\
                               .join(clientes_nomina_12m.filter(f"cast(partitio
                                                            on = ['cod_paisoal
                                                           'cod_persctpn', 'pa

    clientes_atr_nom_cuentas = clientes_atr_nom_cuentas.fillna(0, subset=['num_c
                               'Num_nominas_tot', 'Num_nominas_mes', 'Ind_nomi
                               'Ind_nomina_bimes'])


    clientes_atr_nom_cuentas.write.mode("append").partitionBy("partition_id").sa

    texto = "Ejecutada la particion {0} de {1}".format(particion, nombre_tabla)

    print(texto)
```