Education_Silver_Generation_Notebook

New notebook

Medallion Architecture - Silver Tier

In [ ]:
```
from pyspark.sql import functions as F

BRONZE_DB = "bronze"
SILVER_DB = "silver"

def nullif_blank(c):
    return F.when(F.trim(F.col(c)) == "", F.lit(None)).otherwise(F.col(c))

def write_silver(df, table_name: str):
    (df.write.mode("overwrite").format("delta")
       .saveAsTable(f"{SILVER_DB}_{table_name}"))

# ======================
# Silver: DIMENSIONS
# ======================

# dim_date
silver_dimDate = (
    spark.table(f"{BRONZE_DB}_dim_date")
    .select(
        F.col("DateKey").cast("int").alias("DateKey"),
        F.to_date("Date").alias("Date"),
        F.col("Year").cast("int").alias("Year"),
        F.quarter(F.col("Date")).alias("Quarter"),
        F.concat(F.lit("Q"), F.quarter(F.col("Date"))).alias("QuarterLabel"),
        F.col("Month").cast("int").alias("Month"),
        F.date_format(F.col("Date"), "MMMM").alias("MonthName"),
        F.col("Week").cast("int").alias("Week"),
        F.col("AcademicYear").cast("string").alias("AcademicYear"),
        F.col("Term").cast("string").alias("Term"),
        F.col("IsSchoolDay").cast("int").alias("IsSchoolDay"),
        F.date_format(F.col("Date"), "EEEE").alias("DayofWeek")
    )
    .dropDuplicates(["DateKey"])
)
write_silver(silver_dimDate, "dimDate")

#display(silver_dimDate)
```

Not part of the silver workflow - a piece of snippet to alter table in case of adding new columns

In [ ]:
```
Run_Debug = False
if Run_Debug == True:
    spark.sql("""
    ALTER TABLE gold_dimDate
```

```
    ADD COLUMNS
    (
        Quarter INTEGER,
        QuarterLabel STRING,
        MonthName STRING,
        DayofWeek STRING

    )

    """)
```

Continue Silver process

In [ ]:

```
# dim_geography
silver_dimGeography = (
    spark.table(f"{BRONZE_DB}_dim_geography")
    .select(
        F.col("GeoKey").cast("int").alias("GeoKey"),
        nullif_blank("LocalAuthority").cast("string").alias("LocalAuthority"),
        nullif_blank("Ward").cast("string").alias("Ward"),
        nullif_blank("PostcodeArea").cast("string").alias("PostcodeArea"),
        nullif_blank("Region").cast("string").alias("Region"),
    )
    .dropDuplicates(["GeoKey"])
)
write_silver(silver_dimGeography, "dimGeography")

# dim_trust
silver_dimTrust = (
    spark.table(f"{BRONZE_DB}_dim_trust")
    .select(
        F.col("TrustKey").cast("int").alias("TrustKey"),
        nullif_blank("TrustId").cast("string").alias("TrustId"),
        nullif_blank("TrustName").cast("string").alias("TrustName"),
        nullif_blank("TrustType").cast("string").alias("TrustType"),
        nullif_blank("LocalAuthority").cast("string").alias("LocalAuthority"),
    )
    .dropDuplicates(["TrustId"])
)
write_silver(silver_dimTrust, "dimTrust")

# dim_school
silver_dimSchool = (
    spark.table(f"{BRONZE_DB}_dim_school")
    .select(
        F.col("SchoolKey").cast("int").alias("SchoolKey"),
        nullif_blank("SchoolId").cast("string").alias("SchoolId"),
        nullif_blank("SchoolName").cast("string").alias("SchoolName"),
        nullif_blank("SchoolType").cast("string").alias("SchoolType"),
        nullif_blank("TrustId").cast("string").alias("TrustId"),
        F.col("GeoKey").cast("int").alias("GeoKey"),
        nullif_blank("Phase").cast("string").alias("Phase"),
        nullif_blank("IsSendSpecialist").cast("string").alias("IsSendSpecialist"),
        F.col("Capacity").cast("int").alias("Capacity"),
    )
```

```
    .dropDuplicates(["SchoolId"])
)
write_silver(silver_dimSchool, "dimSchool")

# dim_pupil
silver_dimPupil = (
    spark.table(f"{BRONZE_DB}_dim_pupil")
    .select(
        F.col("PupilKey").cast("int").alias("PupilKey"),
        nullif_blank("PupilId").cast("string").alias("PupilId"),
        nullif_blank("UPN_masked").cast("string").alias("UPN_masked"),
        F.to_date("DOB").alias("DOB"),
        nullif_blank("Gender").cast("string").alias("Gender"),
        nullif_blank("Ethnicity").cast("string").alias("Ethnicity"),
        F.col("EAL").cast("int").alias("EAL"),
        F.col("FSM").cast("int").alias("FSM"),
        F.col("LAC_flag").cast("int").alias("LAC_flag"),
        F.col("SEN_flag").cast("int").alias("SEN_flag"),
        nullif_blank("SEN_Status").cast("string").alias("SEN_Status"),
    )
    .dropDuplicates(["PupilKey"])
)
write_silver(silver_dimPupil, "dimPupil")

# dim_send_need
silver_dimSendNeed = (
    spark.table(f"{BRONZE_DB}_dim_send_need")
    .select(
        F.col("NeedKey").cast("int").alias("NeedKey"),
        nullif_blank("PrimaryNeed").cast("string").alias("PrimaryNeed"),
        nullif_blank("PrimaryNeedDesc").cast("string").alias("PrimaryNeedDesc"),
        nullif_blank("SecondaryNeed").cast("string").alias("SecondaryNeed"),
    )
    .dropDuplicates(["NeedKey"])
)
write_silver(silver_dimSendNeed, "dimSendNeed")

# dim_intervention
silver_dimIntervention = (
    spark.table(f"{BRONZE_DB}_dim_intervention")
    .select(
        F.col("InterventionKey").cast("int").alias("InterventionKey"),
        nullif_blank("InterventionType").cast("string").alias("InterventionType"),
        nullif_blank("IntensityBand").cast("string").alias("IntensityBand"),
    )
    .dropDuplicates(["InterventionKey"])
)
write_silver(silver_dimIntervention, "dimIntervention")

# ======================
# Silver: FACTS
# ======================

# fact_enrolment
silver_factEnrolment = (
    spark.table(f"{BRONZE_DB}_fact_enrolment")
```

```
    .select(
        F.col("PupilKey").cast("int").alias("PupilKey"),
        F.col("SchoolKey").cast("int").alias("SchoolKey"),
        F.col("StartDateKey").cast("int").alias("StartDateKey"),
        F.col("EndDateKey").cast("int").alias("EndDateKey"),
        F.col("YearGroup").cast("int").alias("YearGroup"),
        nullif_blank("EntryReason").cast("string").alias("EntryReason"),
        nullif_blank("ExitReason").cast("string").alias("ExitReason"),
    )
)
write_silver(silver_factEnrolment, "factEnrolment")

# fact_attendance_daily (derive sessions + flags)
silver_factAttendance = (
    spark.table(f"{BRONZE_DB}_fact_attendance_daily")
    .select(
        F.col("PupilKey").cast("int").alias("PupilKey"),
        F.col("SchoolKey").cast("int").alias("SchoolKey"),
        F.col("DateKey").cast("int").alias("DateKey"),
        nullif_blank("SessionAM").cast("string").alias("SessionAM"),
        nullif_blank("SessionPM").cast("string").alias("SessionPM"),
        nullif_blank("AttendanceCode").cast("string").alias("AttendanceCode"),
    )
    .withColumn("PresentAM", F.when(F.col("SessionAM").isin("P","L"), 1).otherwise(
    .withColumn("PresentPM", F.when(F.col("SessionPM").isin("P","L"), 1).otherwise(
    .withColumn("SessionsPossible", F.lit(2))
    .withColumn("SessionsPresent", F.col("PresentAM") + F.col("PresentPM"))
    .withColumn("IsAbsentDay", F.when(F.col("SessionsPresent") == 0, 1).otherwise(0
    .withColumn("AttendanceRateDay", F.col("SessionsPresent") / F.col("SessionsPoss
)
write_silver(silver_factAttendance, "factAttendanceDaily")

# fact_attainment_termly (derive AtOrAboveFlag)
silver_factAttainment = (
    spark.table(f"{BRONZE_DB}_fact_attainment_termly")
    .select(
        F.col("PupilKey").cast("int").alias("PupilKey"),
        F.col("SchoolKey").cast("int").alias("SchoolKey"),
        F.col("DateKey").cast("int").alias("DateKey"),  # term end
        nullif_blank("Subject").cast("string").alias("Subject"),
        nullif_blank("GradeBand").cast("string").alias("GradeBand"),
        F.col("ScaledScore").cast("int").alias("ScaledScore"),
        F.col("ExpectedProgressFlag").cast("int").alias("ExpectedProgressFlag"),
    )
    .withColumn("AtOrAboveFlag", F.when(F.col("GradeBand").isin("At","Above"), 1).o
)
write_silver(silver_factAttainment, "factAttainmentTermly")

# fact_behaviour_incidents (derive severity flags)
silver_factBehaviour = (
    spark.table(f"{BRONZE_DB}_fact_behaviour_incidents")
    .select(
        F.col("IncidentKey").cast("int").alias("IncidentKey"),
        F.col("PupilKey").cast("int").alias("PupilKey"),
        F.col("SchoolKey").cast("int").alias("SchoolKey"),
        F.col("DateKey").cast("int").alias("DateKey"),
```

```
        nullif_blank("IncidentType").cast("string").alias("IncidentType"),
        F.col("Severity").cast("int").alias("Severity"),
        F.col("ExclusionFlag").cast("int").alias("ExclusionFlag"),
        nullif_blank("ExclusionType").cast("string").alias("ExclusionType"),
        F.col("ExclusionDays").cast("int").alias("ExclusionDays"),
    )
    .withColumn("IsHighSeverity", F.when(F.col("Severity") >= 4, 1).otherwise(0))
)
write_silver(silver_factBehaviour, "factBehaviourIncidents")

# fact_send_provision (derive funding band)
silver_factSendProvision = (
    spark.table(f"{BRONZE_DB}_fact_send_provision")
    .select(
        F.col("PupilKey").cast("int").alias("PupilKey"),
        F.col("SchoolKey").cast("int").alias("SchoolKey"),
        F.col("DateKey").cast("int").alias("DateKey"),  # month end
        F.col("NeedKey").cast("int").alias("NeedKey"),
        F.col("EHCPFlag").cast("int").alias("EHCPFlag"),
        F.col("ProvisionHoursPerWeek").cast("double").alias("ProvisionHoursPerWeek"
        nullif_blank("PlacementType").cast("string").alias("PlacementType"),
        F.col("AnnualTopUpFunding").cast("double").alias("AnnualTopUpFunding"),
    )
    .withColumn(
        "FundingBand",
        F.when(F.col("AnnualTopUpFunding") < 2000, "0-2k")
         .when(F.col("AnnualTopUpFunding") < 8000, "2k-8k")
         .otherwise("8k+")
    )
)
write_silver(silver_factSendProvision, "factSendProvision")

# fact_intervention_participation (derive delta)
silver_factIntervention = (
    spark.table(f"{BRONZE_DB}_fact_intervention_participation")
    .select(
        F.col("PupilKey").cast("int").alias("PupilKey"),
        F.col("SchoolKey").cast("int").alias("SchoolKey"),
        F.col("InterventionKey").cast("int").alias("InterventionKey"),
        F.col("StartDateKey").cast("int").alias("StartDateKey"),
        F.col("EndDateKey").cast("int").alias("EndDateKey"),
        nullif_blank("TargetOutcome").cast("string").alias("TargetOutcome"),
        F.col("BaselineValue").cast("double").alias("BaselineValue"),
        F.col("EndValue").cast("double").alias("EndValue"),
        nullif_blank("ImpactBand").cast("string").alias("ImpactBand"),
    )
    .withColumn("DeltaValue", F.col("EndValue") - F.col("BaselineValue"))
)
write_silver(silver_factIntervention, "factInterventionParticipation")

print("Silver complete.")
```

```
StatementMeta(, bf075302-dd02-4787-ba0b-644113049d0c, -1, Cancelled, , Cancelled, Tr
ue)
```