Juin - 2025

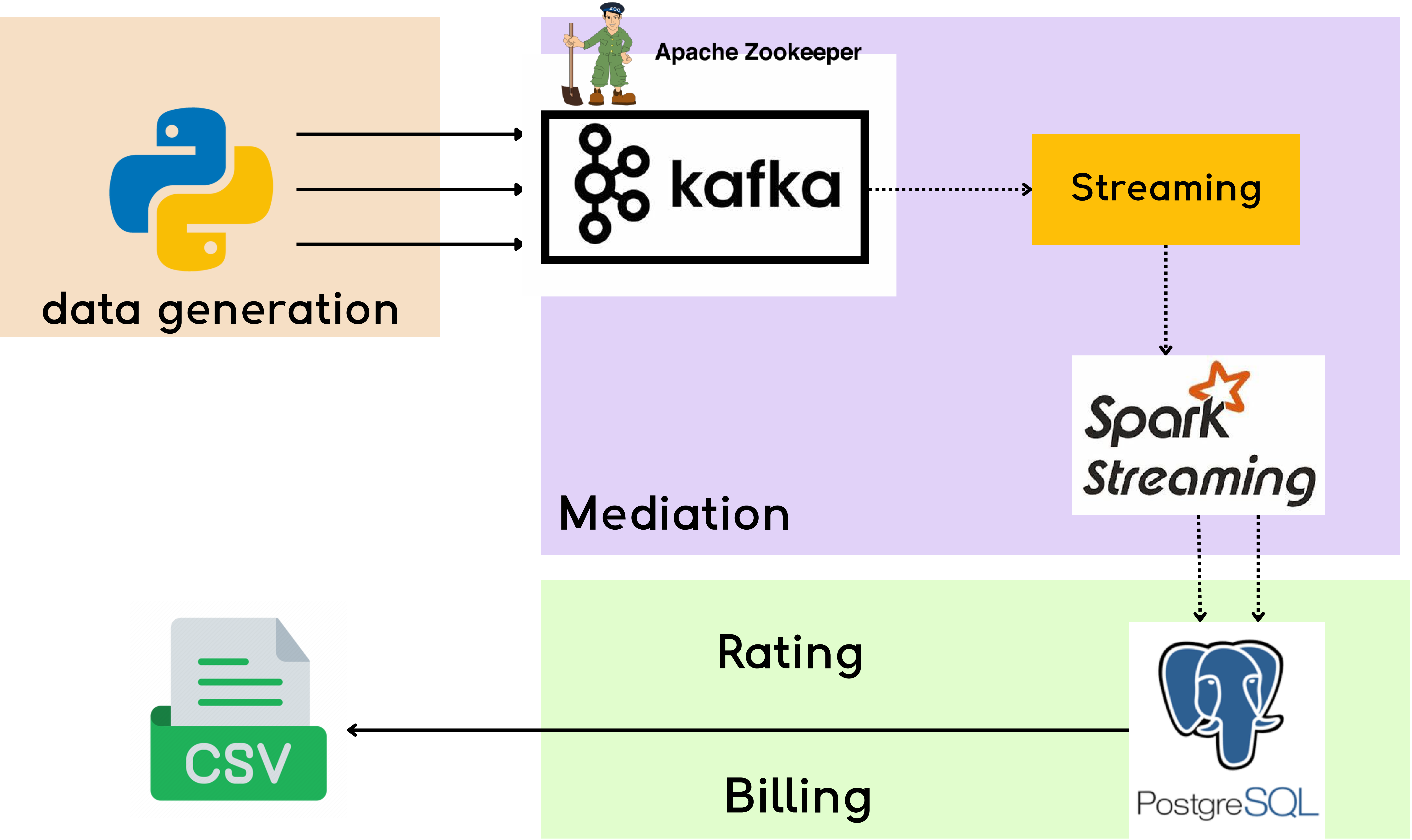
Telecom Data Pipeline

Réalisée par:

Elamrani Mariem
Bakadiri Widad

Encadrée par :

MOHAMED EL MAROUANI



data generation

Apache Zookeeper

Streaming

Mediation

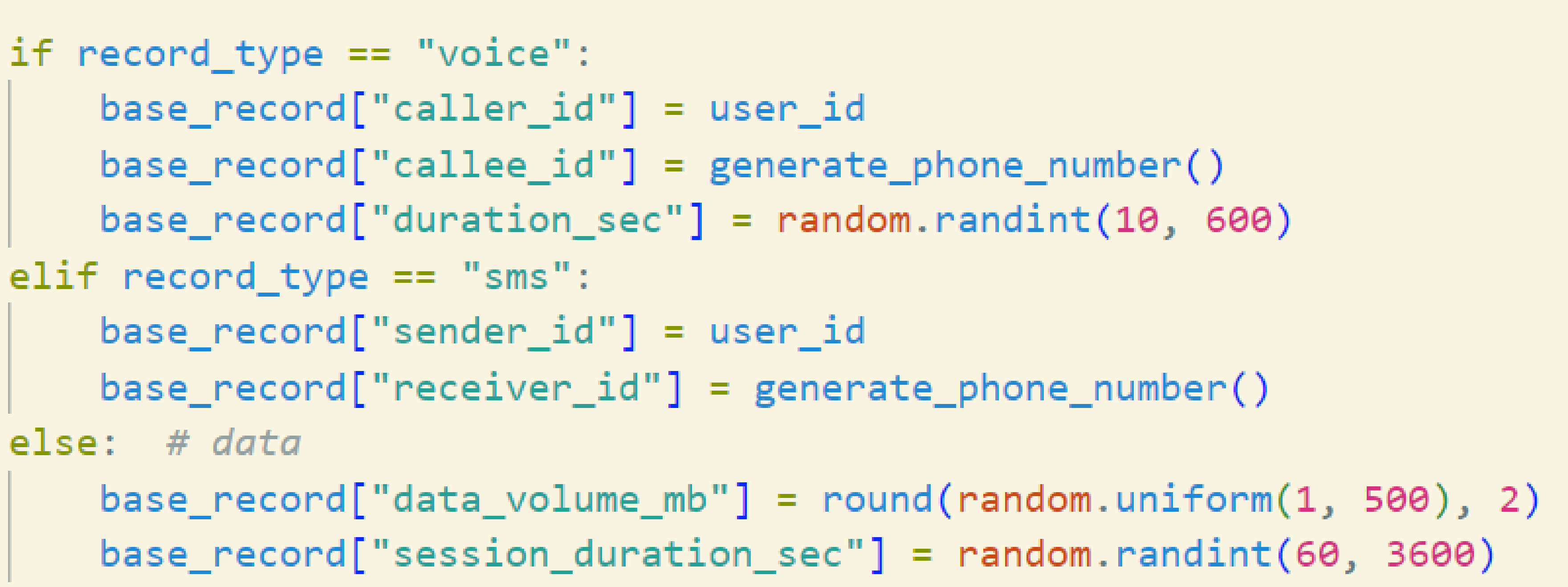
Rating

Billing

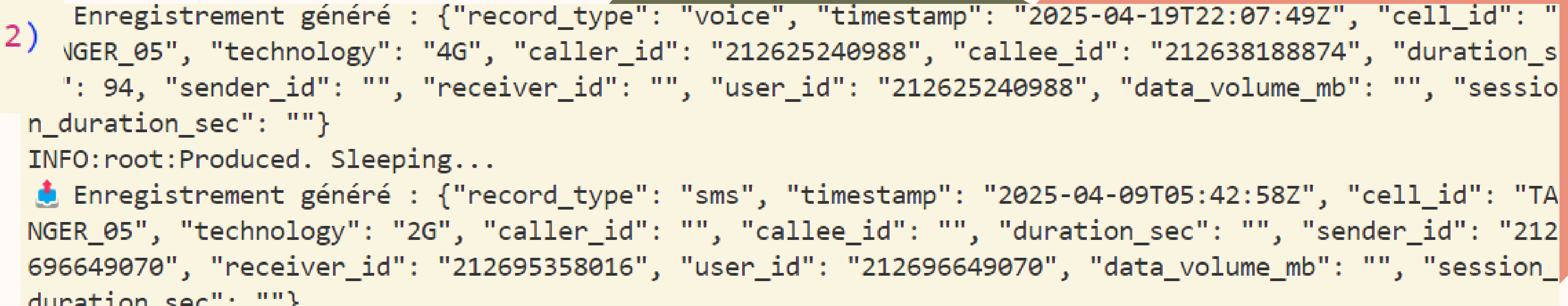
Juin - 2025

Synthetic data generation engine

- Génération de EDR/CDR (voice, sms, data) via un script Python
- Envoi en temps réel vers Kafka (topic records)
- Données réalistes : timestamp, user_id, cell_id, volume, durée…
- Ajout de bruit : 10% données manquantes/corrompues + 5% de doublons
- Récupération dynamique des user_id depuis PostgreSQL



```
if record_type == "voice":
    base_record["caller_id"] = user_id
    base_record["callee_id"] = generate_phone_number()
    base_record["duration_sec"] = random.randint(10, 600)
elif record_type == "sms":
    base_record["sender_id"] = user_id
    base_record["receiver_id"] = generate_phone_number()
else:  # data
    base_record["data_volume_mb"] = round(random.uniform(1, 500), 2)
    base_record["session_duration_sec"] = random.randint(60, 3600)
```



```
 Enregistrement généré : {"record_type": "voice", "timestamp": "2025-04-19T22:07:49Z", "cell_id": "
NGER_05", "technology": "4G", "caller_id": "212625240988", "callee_id": "212638188874", "duration_s
': 94, "sender_id": "", "receiver_id": "", "user_id": "212625240988", "data_volume_mb": "", "sessio
n_duration_sec": ""}
INFO:root:Produced. Sleeping...
📤 Enregistrement généré : {"record_type": "sms", "timestamp": "2025-04-09T05:42:58Z", "cell_id": "TA
NGER_05", "technology": "2G", "caller_id": "", "callee_id": "", "duration_sec": "", "sender_id": "212
696649070", "receiver_id": "212695358016", "user_id": "212696649070", "data_volume_mb": "", "session_
duration_sec": ""}
```

Juin - 2025

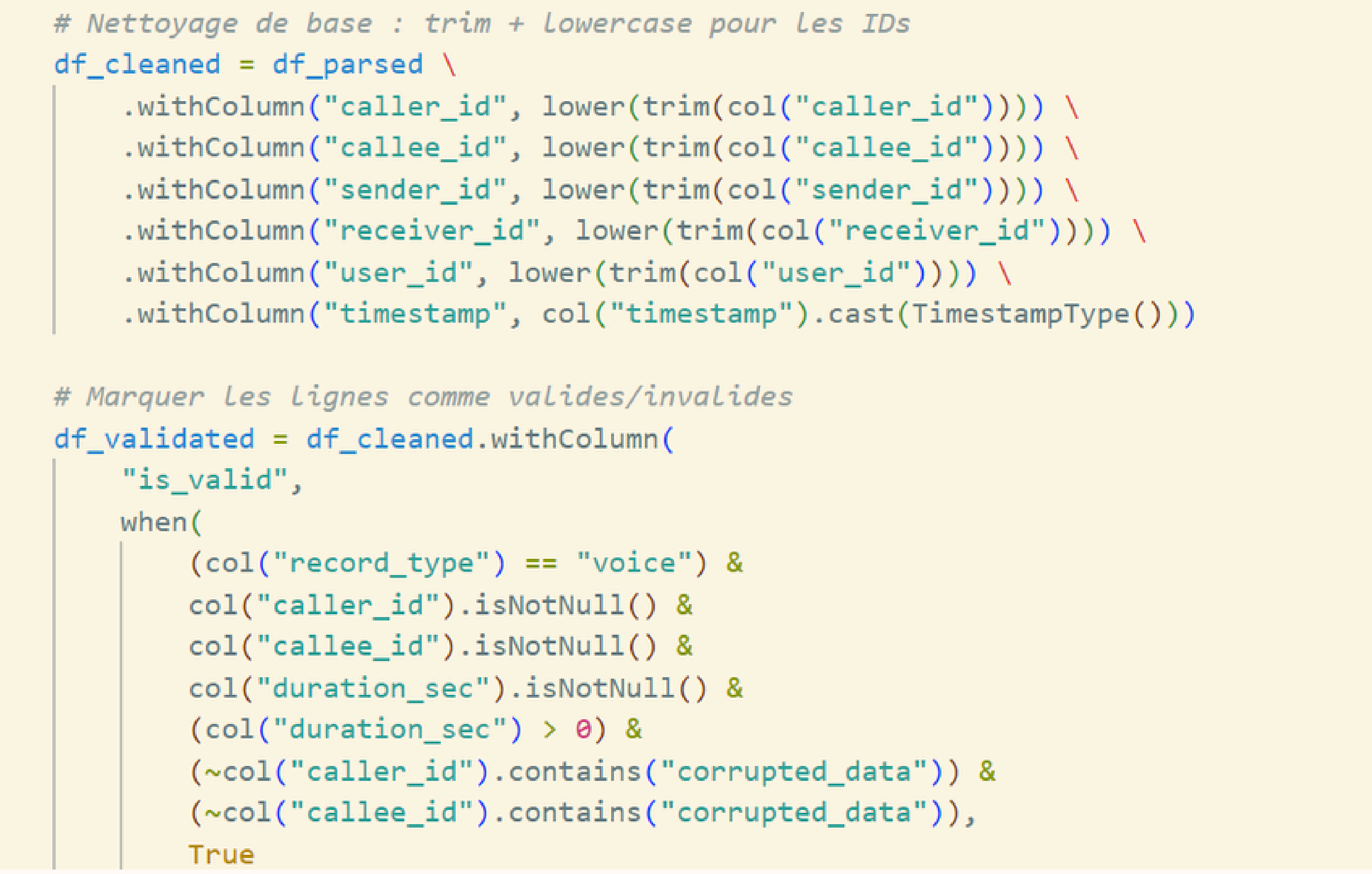
Mediation

- Lecture des messages Kafka en temps réel
- Nettoyage & normalisation :

  Détection/filtrage des champs corrompus

  Complétion des formats

  Écriture vers clean_records et dirty_records



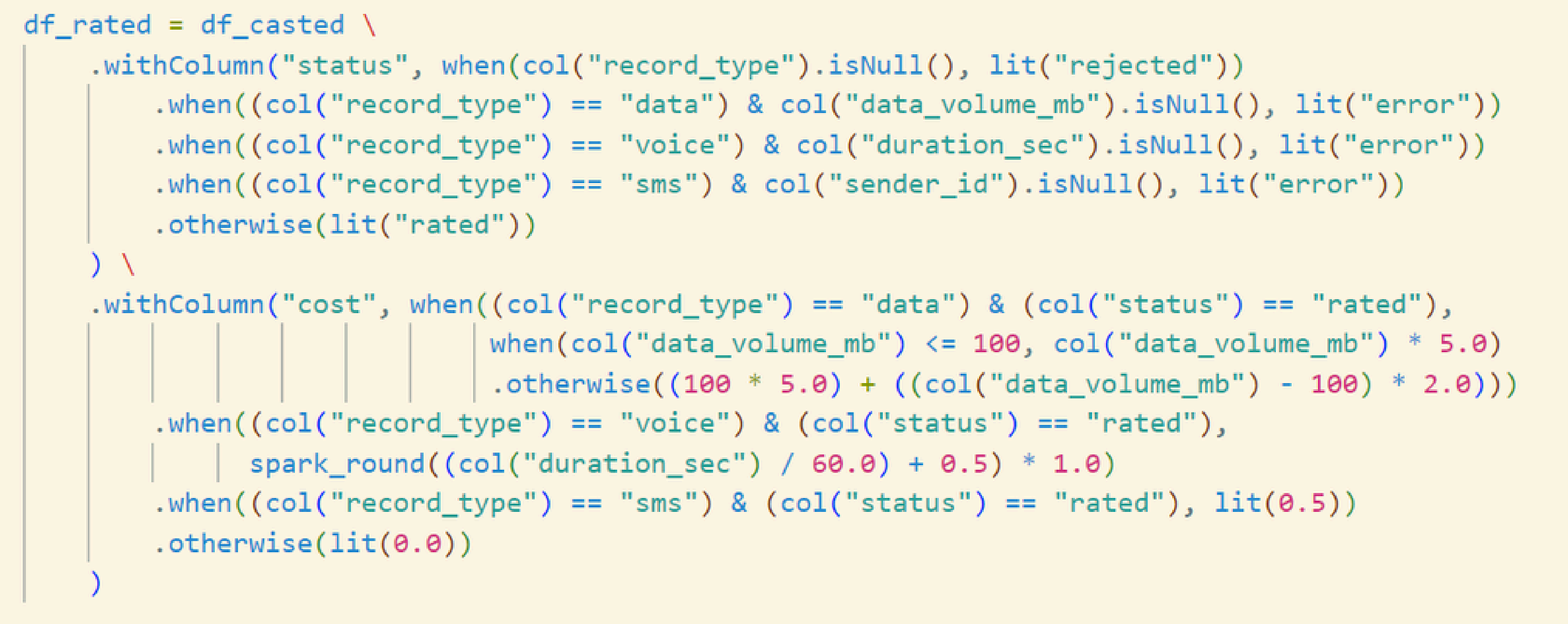
```
# Nettoyage de base : trim + lowercase pour les IDs
df_cleaned = df_parsed \
    .withColumn("caller_id", lower(trim(col("caller_id")))) \
    .withColumn("callee_id", lower(trim(col("callee_id")))) \
    .withColumn("sender_id", lower(trim(col("sender_id")))) \
    .withColumn("receiver_id", lower(trim(col("receiver_id")))) \
    .withColumn("user_id", lower(trim(col("user_id")))) \
    .withColumn("timestamp", col("timestamp").cast(TimestampType()))

# Marquer les lignes comme valides/invalides
df_validated = df_cleaned.withColumn(
    "is_valid",
    when(
        (col("record_type") == "voice") &
        col("caller_id").isNotNull() &
        col("callee_id").isNotNull() &
        col("duration_sec").isNotNull() &
        (col("duration_sec") > 0) &
        (~col("caller_id").contains("corrupted_data")) &
        (~col("callee_id").contains("corrupted_data")),
        True
```

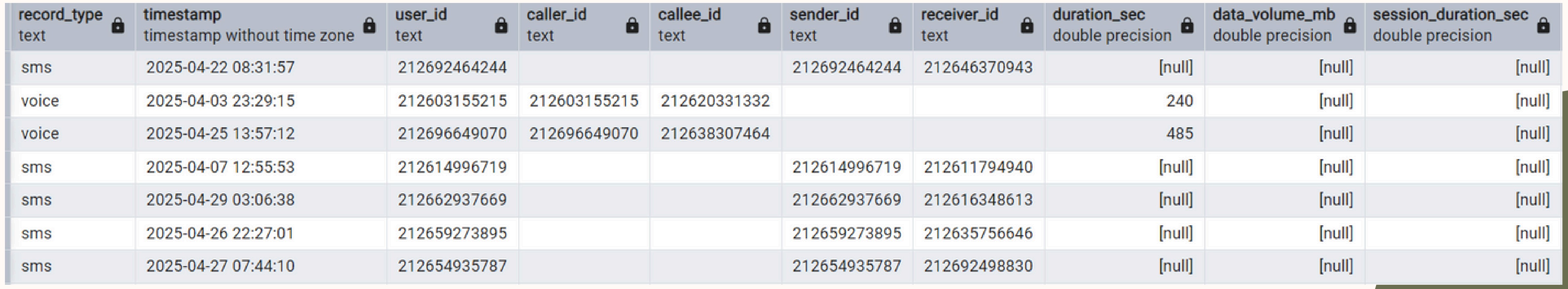
Juin - 2025

Rating Engine

- Lecture des messages Kafka en temps réel
- Application des règles tarifaires :
  voice : X dh/min
  sms : Y dh/unité
  data : Z dh/MB
- Calcul du coût par service et par utilisateur
- Enregistrement des résultats tarifés



```
df_rated = df_casted \
    .withColumn("status", when(col("record_type").isNull(), lit("rejected"))
        .when((col("record_type") == "data") & col("data_volume_mb").isNull(), lit("error"))
        .when((col("record_type") == "voice") & col("duration_sec").isNull(), lit("error"))
        .when((col("record_type") == "sms") & col("sender_id").isNull(), lit("error"))
        .otherwise(lit("rated"))
    ) \
    .withColumn("cost", when((col("record_type") == "data") & (col("status") == "rated"),
                            when(col("data_volume_mb") <= 100, col("data_volume_mb") * 5.0)
                            .otherwise((100 * 5.0) + ((col("data_volume_mb") - 100) * 2.0)))
        .when((col("record_type") == "voice") & (col("status") == "rated"),
              spark_round((col("duration_sec") / 60.0) + 0.5) * 1.0)
        .when((col("record_type") == "sms") & (col("status") == "rated"), lit(0.5))
        .otherwise(lit(0.0))
    )
```



record_type text	timestamp timestamp without time zone	user_id text	caller_id text	callee_id text	sender_id text	receiver_id text	duration_sec double precision	data_volume_mb double precision	session_duration_sec double precision
sms	2025-04-22 08:31:57	212692464244			212692464244	212646370943	[null]	[null]	[null]
voice	2025-04-03 23:29:15	212603155215	212603155215	212620331332			240	[null]	[null]
voice	2025-04-25 13:57:12	212696649070	212696649070	212638307464			485	[null]	[null]
sms	2025-04-07 12:55:53	212614996719			212614996719	212611794940	[null]	[null]	[null]
sms	2025-04-29 03:06:38	212662937669			212662937669	212616348613	[null]	[null]	[null]
sms	2025-04-26 22:27:01	212659273895			212659273895	212635756646	[null]	[null]	[null]
sms	2025-04-27 07:44:10	212654935787			212654935787	212692498830	[null]	[null]	[null]

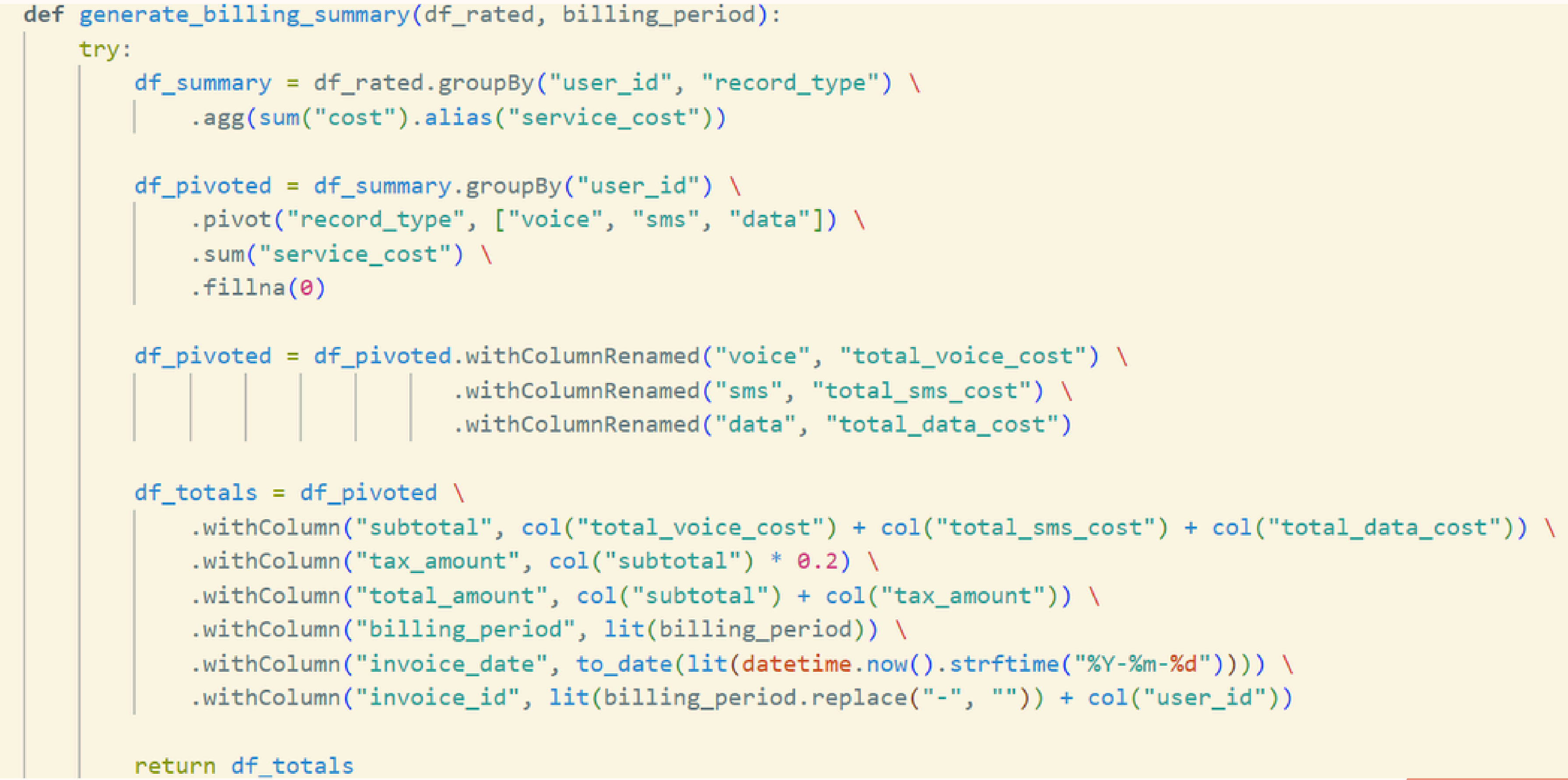
Juin - 2025

Billing Engine

- Agrégation des coûts par utilisateur
- Génération de facture mensuelle :

  Total par service

  Total global à payer
- Enregistrement final (dans un fichier csv)



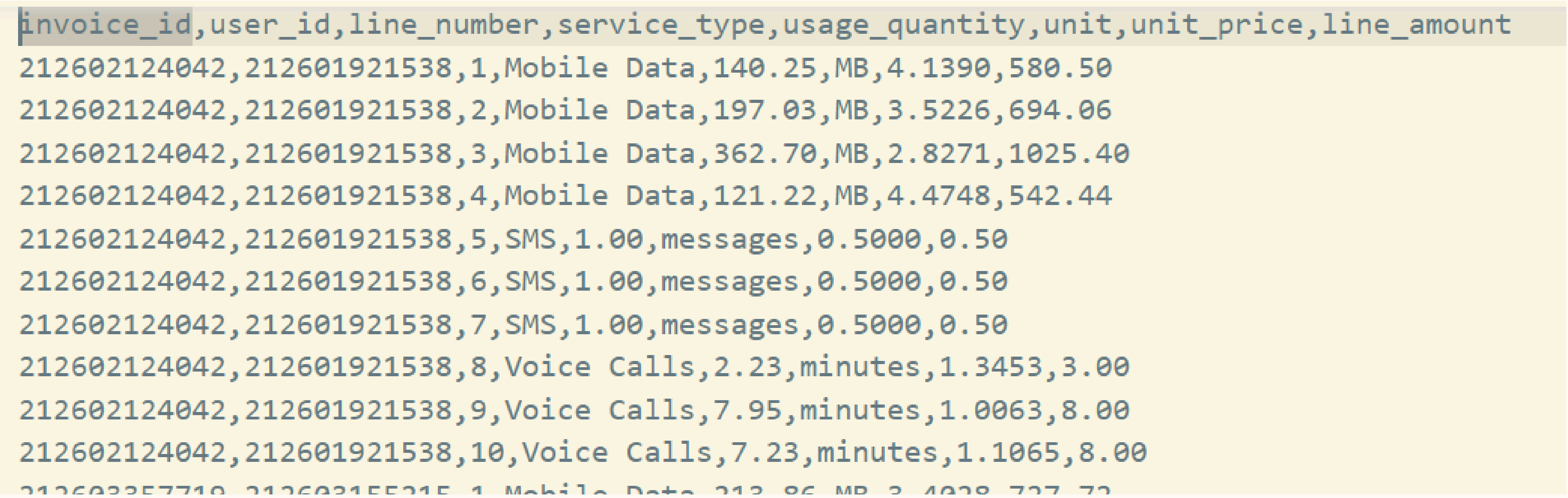
```python
def generate_billing_summary(df_rated, billing_period):
    try:
        df_summary = df_rated.groupBy("user_id", "record_type") \
            .agg(sum("cost").alias("service_cost"))

        df_pivoted = df_summary.groupBy("user_id") \
            .pivot("record_type", ["voice", "sms", "data"]) \
            .sum("service_cost") \
            .fillna(0)

        df_pivoted = df_pivoted.withColumnRenamed("voice", "total_voice_cost") \
                               .withColumnRenamed("sms", "total_sms_cost") \
                               .withColumnRenamed("data", "total_data_cost")

        df_totals = df_pivoted \
            .withColumn("subtotal", col("total_voice_cost") + col("total_sms_cost") + col("total_data_cost")) \
            .withColumn("tax_amount", col("subtotal") * 0.2) \
            .withColumn("total_amount", col("subtotal") + col("tax_amount")) \
            .withColumn("billing_period", lit(billing_period)) \
            .withColumn("invoice_date", to_date(lit(datetime.now().strftime("%Y-%m-%d")))) \
            .withColumn("invoice_id", lit(billing_period.replace("-", "")) + col("user_id"))

        return df_totals
```



```
invoice_id,user_id,line_number,service_type,usage_quantity,unit,unit_price,line_amount
212602124042,212601921538,1,Mobile Data,140.25,MB,4.1390,580.50
212602124042,212601921538,2,Mobile Data,197.03,MB,3.5226,694.06
212602124042,212601921538,3,Mobile Data,362.70,MB,2.8271,1025.40
212602124042,212601921538,4,Mobile Data,121.22,MB,4.4748,542.44
212602124042,212601921538,5,SMS,1.00,messages,0.5000,0.50
212602124042,212601921538,6,SMS,1.00,messages,0.5000,0.50
212602124042,212601921538,7,SMS,1.00,messages,0.5000,0.50
212602124042,212601921538,8,Voice Calls,2.23,minutes,1.3453,3.00
212602124042,212601921538,9,Voice Calls,7.95,minutes,1.0063,8.00
212602124042,212601921538,10,Voice Calls,7.23,minutes,1.1065,8.00
212603357719,212603155215,1,Mobile Data,213.86,MB,3.4028,727.72
```

Juin - 2025

Merci pour votre attention