## SECCIÓN 2: ETL CON PYSPARK Y CARGAS INCREMENTALES

### CONTEXTO:

En Podemos Progresar, integramos datos de nuestro sistema central de créditos grupales, aplicaciones móviles usadas por oficiales de campo, y sistemas de pagos. Necesitamos procesar eficientemente los datos de formación de grupos, solicitudes, desembolsos y pagos semanales.

### Escenario:

Comparte tu experiencia real implementando soluciones de carga incremental en entornos financieros, especialmente aquellas relacionadas con procesamiento de transacciones frecuentes y datos de clientes.

### Tareas:

2.1. Estrategias de carga incremental para datos de microcréditos:

- Describe un caso real de carga incremental que hayas implementado para datos financieros transaccionales:

**R.** En mi rol en Aserta teníamos la necesidad de tener la información de pagos actualizadas únicamente con el último status, por lo cual implemente un desarrollo haciendo únicamente cargas incrementales definidas por la llave, tomando en cuenta el filtro de fecha y hora, esto ayudo a solo traerme el delta de las transacciones con fecha del día con la hora en curso.

\* ¿Qué estrategia utilizaste para manejar actualizaciones frecuentes como pagos periódicos?

**R.** En Aserta implemente campos de auditoria para identificar solo los registros modificados de la última carga, campos de estatus de pago. Utilice IPC filtrando en el **Source** Qualifier para capturar solo registros nuevos o modificados, worflows programados cada hora para evitar saturación, particionamiento de datos utilizando worklets, pushdown optimization

\* ¿Cómo manejaste la concurrencia en entornos donde tanto datos maestros como transaccionales se actualizan constantemente?

**R.** En Aserta donde los datos maestros ejemplo clientes, productos como las transacciones los pagos o movimientos en el crédito se actualizaban en tiempo real implemente algunas estrategias utilizando IPC para garantizar la consistencia de la información read commited para evitar lecturas sucias, commit cada 1000 registros en sesiones de IPC esto libera bloqueos rápidamente, rollback automático en caso de fallo y particionado de tablas transaccionales

\* ¿Qué soluciones implementaste para garantizar la integridad referencial entre transacciones y datos maestros?

**R.** En el caso de Aserta implemente algunas estrategias para garantizar la integridad referencial por ejemplo constraints en BD, foreing keys en las tablas transaccionales, ejecución controlado datos maestros(clientes), transacciones(pagos) con lookups para validar la relación, catálogo de datos para trazabilidad de dependencia entre las tablas, estos datos se guardaban en una tabla de control para posteriormente ser monitoreada.

- Proporciona fragmentos específicos de código que hayas desarrollado (modificando datos sensibles) para algún caso relacionado con datos financieros o de crédito

**R.** En Walmart diseñe un pipeline que enmascaraba las tarjetas de crédito(proceso PCI) con un script de Python ,donde mi origen fue cloud storage(archivo parquet), transformación en Datafusion Wrangler para enmascarar los datos sensibles, destino Bigquery , el acceso a estas tablas estaba restringido de acuerdo al rol.

def mask\_sensitive\_data(record):

# Enmascarar tarjeta de crédito (últimos 4 dígitos)

if 'tarjeta\_credito' in record:

record['tarjeta\_credito'] = '\*\*\*\*-\*\*\*\*-\*\*\*\*-' + record['tarjeta\_credito'][-4:]

return record

2.2. Solución para escenarios complejos en microfinanzas:

- Comparte un caso donde tuviste que desarrollar una solución para:

\* Conciliación de pagos recibidos por diferentes canales (efectivo, transferencia, etc.)

**R.** En Aserta los pagos llegaban de múltiples fuentes sin un ID único generando duplicados e inconsistencias, se realizó una normalización con reglas de negocio implementado en IPC, igual se implemento una tabla de control para pagos rechazados por transacciones no reconocidas.

\* Cálculo incremental de saldos o indicadores de morosidad

**R.** Los préstamos de Cashi Walmart requerían actualización diaria de los saldos y un alertamiento a cobranza para clientes con atraso >7 días de atraso. La solución implementada fue un Merge con un filtro de fecha\_update para procesar solo los registros actualizados utilizando composer , particionamiento de las tablas en Bigquery y el monitoreo lo realizaban con Looker y las alertas de morosidad se enviaban via correo cuando los dias de atraso >7. Los resultados esperados fue reducción del tiempo de procesamiento, alertas tempranas de morosidad y ahorro en costo porque solo se procesaban registros diarios

\* Integración de datos recolectados offline y luego sincronizados

**R.** En Walmart para la entrega de pedidos a veces por fallas de conexión se registran en Excel o en sistemas no conectados ala nube, lo que se hace son cargas manuales para poder sincronizar los datos

- Incluye pseudocódigo o fragmentos específicos que muestren cómo resolviste estos problemas

def calcular\_morosidad(df\_prestamos, df\_pagos):

df\_prestamos["fecha\_vencimiento"] = pd.to\_datetime(df\_prestamos["fecha\_vencimiento"])

hoy = datetime.now()

# Calcular días de atraso

df\_prestamos["dias\_atraso"] = (hoy - df\_prestamos["fecha\_vencimiento"]).dt.days

df\_prestamos["dias\_atraso"] = df\_prestamos["dias\_atraso"].apply(lambda x: max(x, 0))

# Clasificar morosidad

df\_prestamos["estado"] = pd.cut(

df\_prestamos["dias\_atraso"],

bins=[-1, 0, 30, 60, float('inf')],

labels=["AL DÍA", "LEVE", "MODERADA", "GRAVE"]

)

return df\_prestamos