Streams y Tareas

---

1. Configuración inicial

Previamente tenemos que clonar las bases de datos para no hacer lo que dice la nota de arriba, y así no alterar la base de datos que estamos utilizando para el resto del curso.

```
USE ROLE CURSO_DATA_ENGINEERING;
CREATE OR REPLACE SCHEMA ALUMNOX_DB.BRONZE CLONE CURSO_SNOWFL
CREATE OR REPLACE SCHEMA ALUMNOX_DB.SILVER CLONE CURSO_SNOWFL
CREATE OR REPLACE SCHEMA ALUMNOX_DB.GOLD CLONE CURSO_SNOWFLAK
```

Los profes previamente hemos creado una tabla de pedidos que nos va a permitir ver el histórico de los datos y así poder compararlo para cuando

realicemos el stream y las tasks. Con la siguiente consulta podemos ver los 10 primeros registros de la tabla de `ORDERS_HIST` :

A continuación vamos a crear un procedure que lo que va a hacer es crear una tabla que resuma el estado y la fecha de la creación de los pedidos:

```
CREATE OR REPLACE PROCEDURE ALUMNOX_DB.GOLD.update_gold_order
RETURNS VARCHAR
LANGUAGE SQL
AS
BEGIN
    CREATE OR REPLACE TABLE ALUMNOX_DB.GOLD.ORDERS_STATUS_DAT
        SELECT
            TO_DATE(CREATED_AT) AS FECHA_CREACION_PEDIDO,
            STATUS,
            COUNT(DISTINCT ORDER_ID) AS NUM_PEDIDOS
        FROM
            ALUMNO24_DB.SILVER.ORDERS
        GROUP BY
            TO_DATE(CREATED_AT),
            STATUS
        ORDER BY 1 DESC
    );
    return 'Tabla ORDERS_STATUS_DATE actualizada con éxito';
END;
```

Para llamar a este procedure, utilizamos la siguiente query:

```
CALL ALUMNOX_DB.GOLD.UPDATE_GOLD_ORDERS_STATUS();
```

2. Creación del Stream y las Tasks

Ahora, vamos a seguir con la creación del Stream:

```
CREATE OR REPLACE STREAM ALUMNOX_DB.BRONZE.ORDERS_STREAM
ON TABLE CURSO_SNOWFLAKE_DE_2023.BRONZE.ORDERS_HIST
```

```
APPEND_ONLY = TRUE;
```

Este Stream lo que va a hacer es detectar cambios en `ORDERS_HIST`, es decir, nos indica en tiempo real si ha habido cambios del tipo inserts o updates en nuestra tabla.

Si queremos ver los streams que hemos creado, podemos hacer:

```
SHOW STREAMS;
```

Una vez creado el Stream, vamos con las tasks. Crearemos 2, una que va a ser la principal (`ROOT_TASK`) y la otra que será una task que dependerá de la task principal (`TASK_HIJA`) .

Creando la ROOT_TASK

Esta tarea lo que va a hacer es ejecutar un `MERGE` entre el Stream y la tabla SILVER_ORDERS. Si encuentra coincidencias actualiza los datos; si no, inserta nuevos registros. Ésta, se ejecutará cada minuto si hay datos nuevos en el Stream.

```
CREATE OR REPLACE TASK ALUMNOX_DB.BRONZE.ROOT_TASK
WAREHOUSE = 'WH_BASICO'
SCHEDULE = '1 MINUTE'
WHEN SYSTEM$STREAM_HAS_DATA('ORDERS_STREAM')
AS
    --MERGE
    MERGE INTO SILVER.ORDERS t
    USING
    (
        SELECT *
        FROM
            ALUMNOX_DB.BRONZE.ORDERS_STREAM
    ) s ON t.ORDER_ID = s.ORDER_ID
    WHEN NOT MATCHED THEN
    INSERT VALUES (
        s.ORDER_ID::varchar(50),
        s.SHIPPING_SERVICE::varchar(20),
```

```
        replace(s.SHIPPING_COST,',','.')::decimal,
        s.ADDRESS_ID::varchar(50),
        s.CREATED_AT::timestamp_ntz,
        IFNULL(s.promo_id,'N/A'),
        s.ESTIMATED_DELIVERY_AT::timestamp_ntz,
        (replace(s.ORDER_COST,',','.'))::decimal,
        s.USER_ID::varchar(50),
        (replace(s.ORDER_TOTAL,',','.'))::decimal,
        s.DELIVERED_AT::timestamp_ntz,
        s.TRACKING_ID::varchar(50),
        s.STATUS::varchar(20),
        TIMESTAMPDIFF(HOUR,s.created_at,s.delivered_at)
    )
    WHEN MATCHED THEN UPDATE SET
        t.ORDER_ID = s.ORDER_ID::varchar(50),
        t.SHIPPING_SERVICE = s.SHIPPING_SERVICE::varchar(20),
        t.SHIPPING_COST = replace(s.SHIPPING_COST,',','.')::d
        t.ADDRESS_ID = s.ADDRESS_ID::varchar(50),
        t.CREATED_AT = s.CREATED_AT::timestamp_ntz,
        t.PROMO_NAME = IFNULL(s.promo_id,'N/A'),
        t.ESTIMATED_DELIVERY_AT = s.ESTIMATED_DELIVERY_AT::ti
        t.ORDER_COST = (REPLACE(s.ORDER_COST,',','.'))::decim
        t.USER_ID = s.USER_ID::varchar(50),
        t.ORDER_TOTAL = (replace(s.ORDER_TOTAL,',','.'))::dec
        t.DELIVERED_AT = s.DELIVERED_AT::timestamp_ntz,
        t.TRACKING_ID = s.TRACKING_ID::varchar(50),
        t.STATUS = s.STATUS::varchar(20),
        t.DELIVERY_TIME_HOURS = TIMESTAMPDIFF(HOUR, s.CREATED_
;
```

Cuando las creamos, Snowflake por defecto las deja en estado “suspendido” y no se ejecutará hasta que explícitamente se active o se reanude. Esto se hace para evitar que la tarea empiece a correr automáticamente antes de que estemos completamente seguros de que se deba iniciar el procesamiento (imagínate que te has equivocado en la task y esto afecta a los datos de producción 😱).

```
ALTER TASK IF EXISTS ALUMNOX_DB.BRONZE.ROOT_TASK RESUME;
```

Y ya la tendríamos activa.

Creando la TASK_HIJA

A continuación, vamos a crear la tarea hija:

```
CREATE OR REPLACE TASK ALUMNOX_DB.BRONZE.TASK_HIJA
WAREHOUSE = 'WH_BASICO'
AFTER ROOT_TASK
AS
CALL GOLD.UPDATE_GOLD_ORDERS_STATUS();
```

No obstante, ahora tenemos una tarea secundaria que depende de una tarea principal. Según la documentación de Snowflake, la ejecución de comandos DDL en cualquier tarea de un árbol de tareas requiere que la **tarea raíz esté suspendida**. Esto garantiza que las tareas hijas estén listas y esperando datos o desencadenantes de la tarea padre, evitando errores de ejecución y asegurando que las dependencias se manejen correctamente. Entonces, primero suspendemos la ROOT_TASK y luego activamos la TASK_HIJA y después volvemos a activar la tarea principal (ROOT_TASK)

```
-- Suspendemos la principal
ALTER TASK IF EXISTS ALUMNOX_DB.BRONZE.ROOT_TASK SUSPEND;
-- Activamos la secundaria y la principal
ALTER TASK IF EXISTS ALUMNOX_DB.BRONZE.TASK_HIJA RESUME;
ALTER TASK IF EXISTS ALUMNOX_DB.BRONZE.ROOT_TASK RESUME;
```

Monitorización del Stream y las Tasks creadas

Ahora vamos a ir haciendo consultas de tipo SELECT para ver cómo se están agregando o actualizando los registros en respuesta a los cambios capturados por el Stream y procesados por las tareas:

```
-- Para ver si se ha ejecutado o no la tarea
SELECT * FROM TABLE(INFORMATION_SCHEMA.TASK_HISTORY()) WHERE
```

```
SELECT * FROM TABLE(INFORMATION_SCHEMA.TASK_HISTORY()) WHERE

-- Para ir viendo los cambios producidos en nuestra tabla
SELECT * FROM ALUMNOX_DB.GOLD.ORDERS_STATUS_DATE;
SELECT * FROM ALUMNOX_DB.BRONZE.ORDERS_STREAM;
```

```
SELECT TOP 10 * FROM CURSO_SNOWFLAKE_DE_2023.BRONZE.ORDERS_HI
```