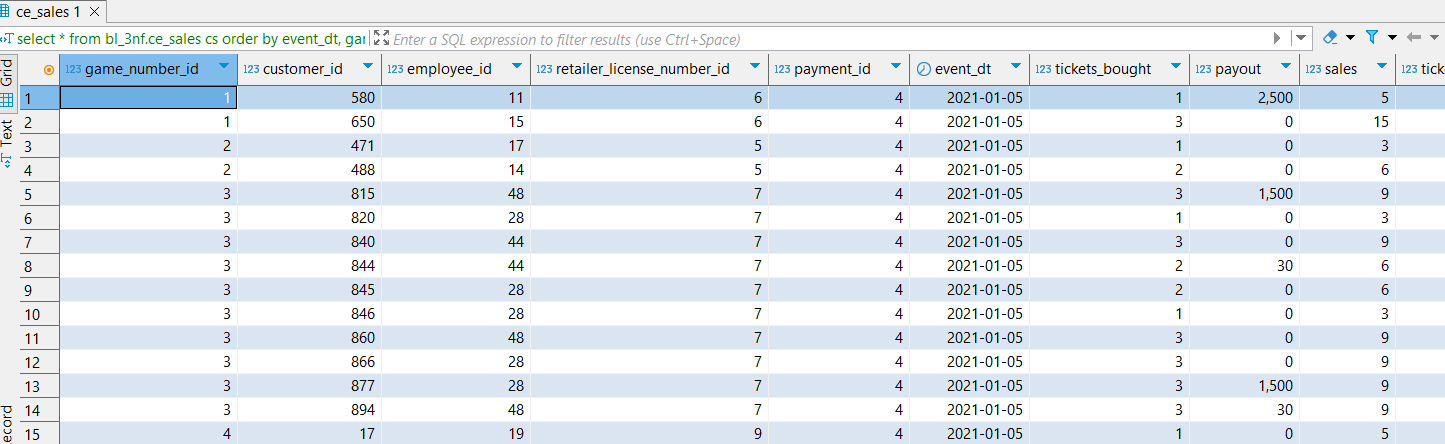


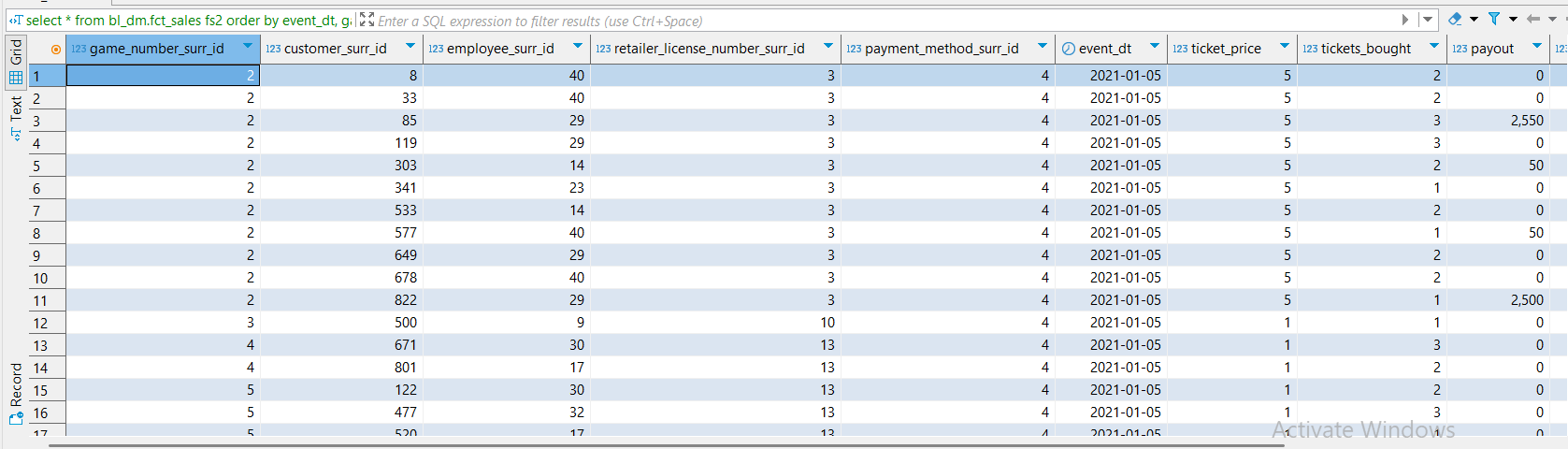
| Business Template  **Loading fact on 3NF / DM** |
| --- |
|  |

----------------------------------------------------------------------------------------------------------------------------------------

---A. Data loaded on 3nf and dm layer of the fact table sales

----------------------------------------------------------------------------------------------------------------------------------------





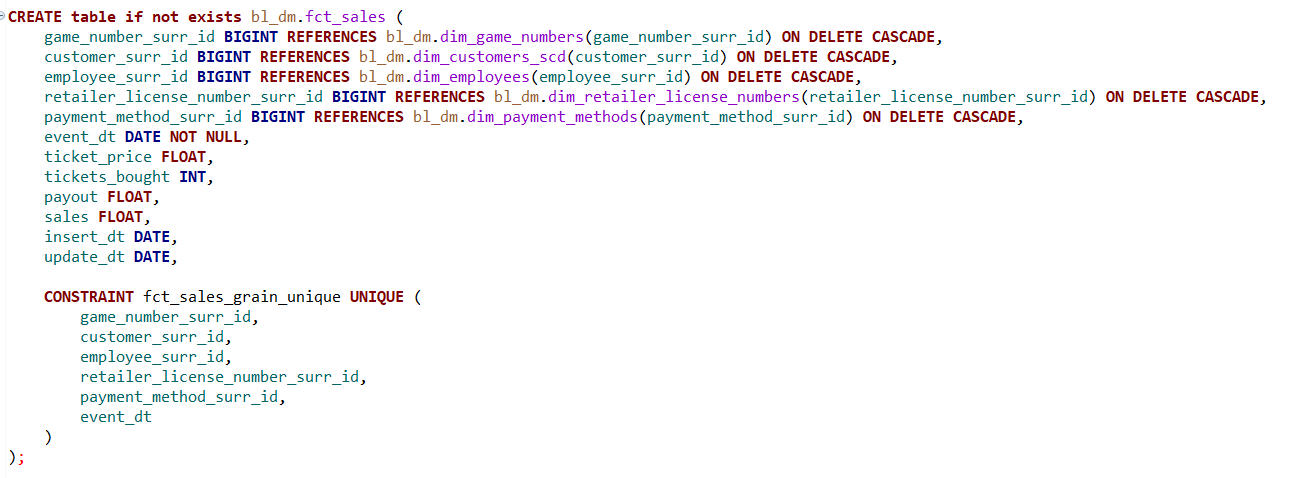
----------------------------------------------------------------------------------------------------------------------------------------

---B. Procedures

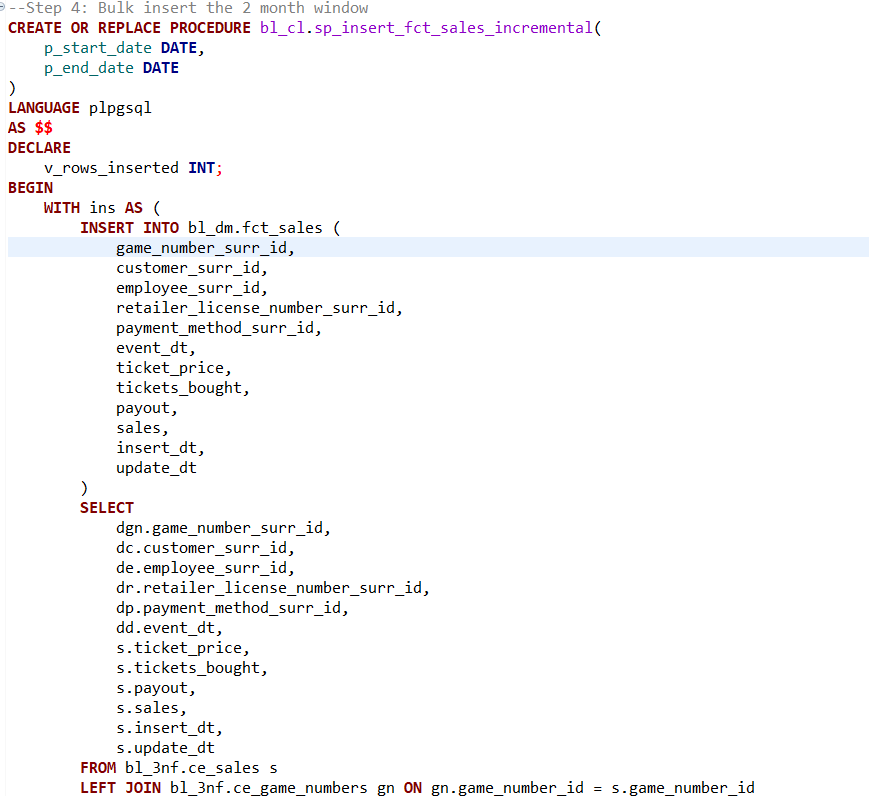
----------------------------------------------------------------------------------------------------------------------------------------

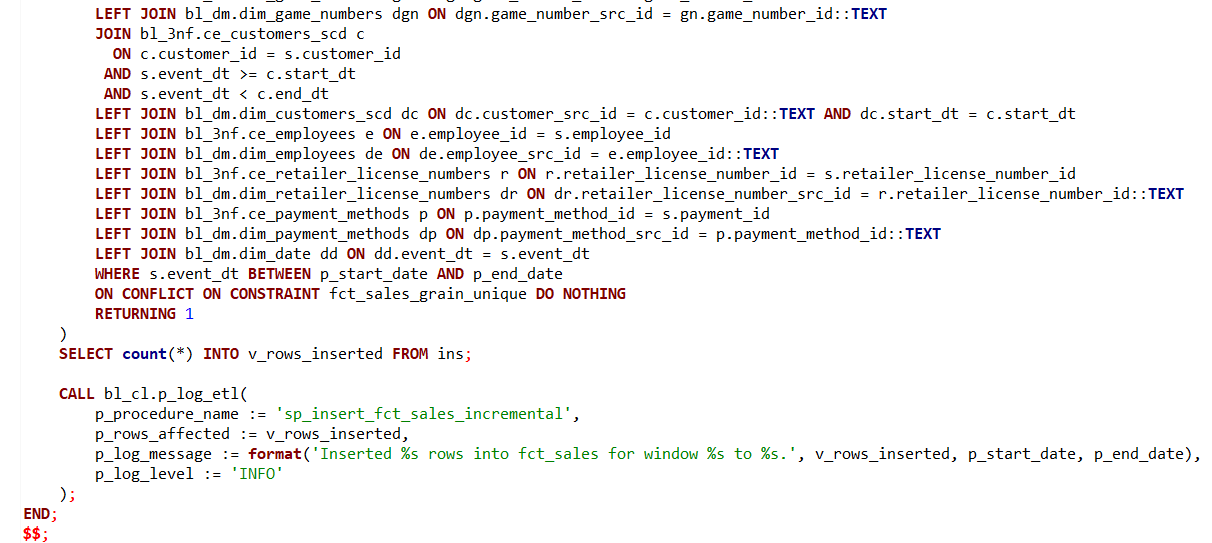


Here, we create a fact sales table partitioned by a range of event dates.

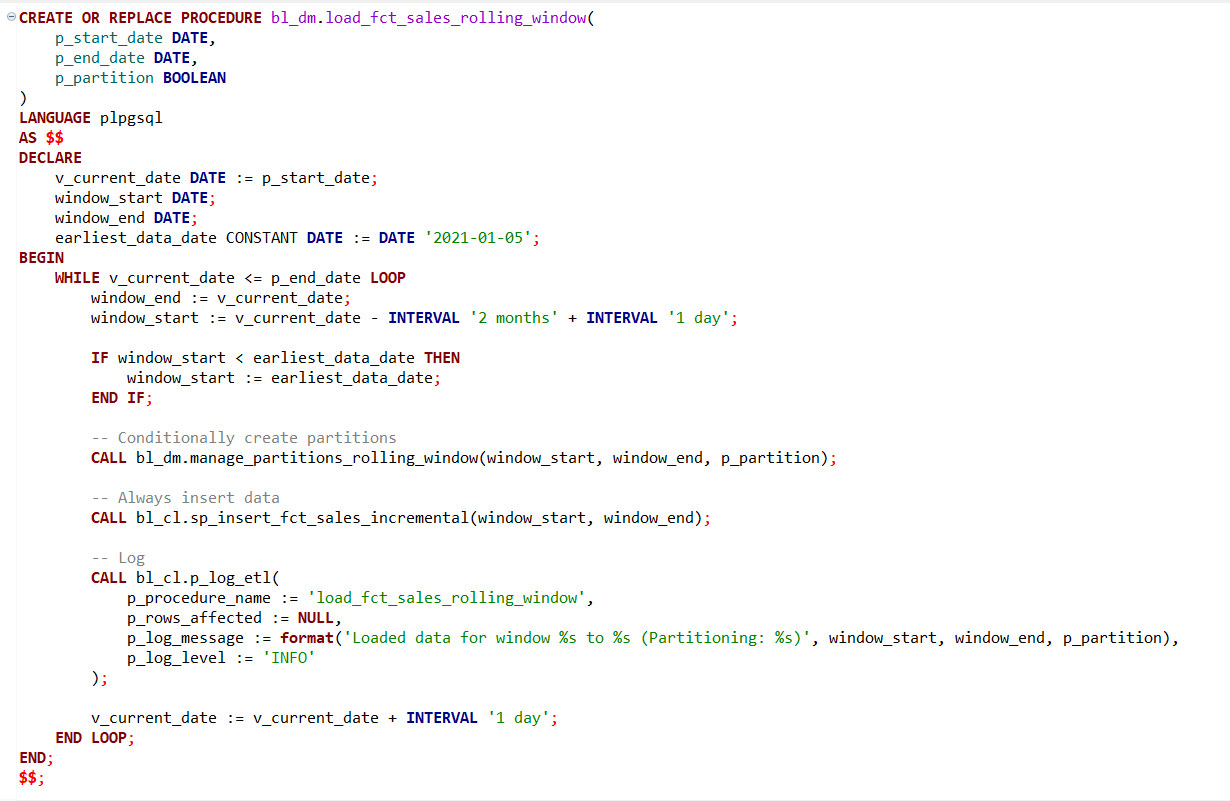


Here, we consider defining a fct sales table where we have no partition on.

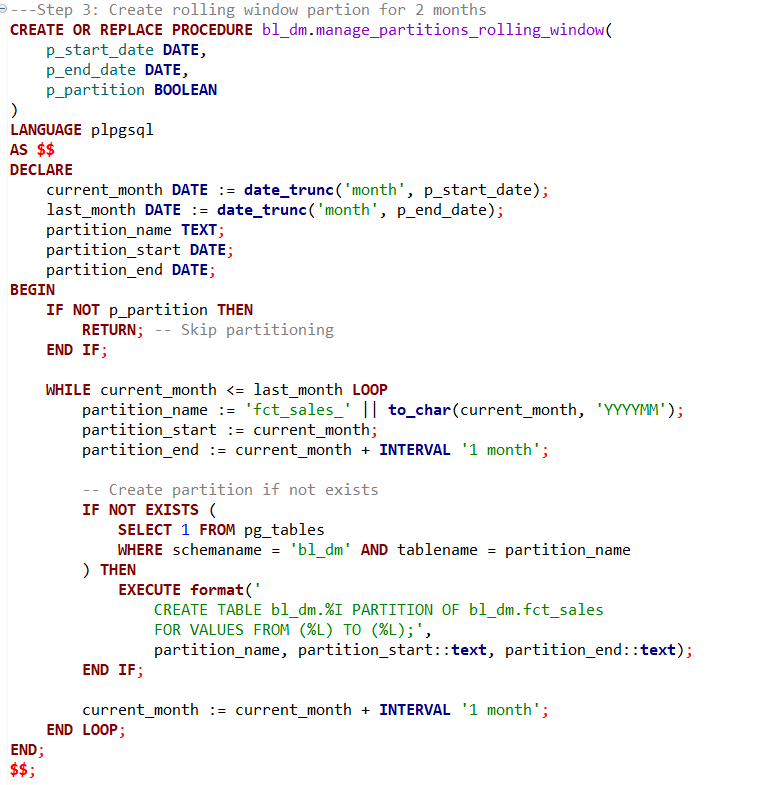




As in Task 8, we begin by ingesting data from the 3NF sales tables. However, this time we use bulk ingestion rather than daily ingestion. Instead of identifying active customers at the time of data insertion, we assign customer entries based on whether the event date falls between the customer's start\_dt and end\_dt. This ensures that, at any given time, the correct customer—with the appropriate historical attributes—is linked to the sales record. This association is enforced through a foreign key constraint between the sales dimension table and the type 2 Slowly Changing Dimension (SCD) customer table.



Here, we implement a rolling window approach, ingesting two months of data in bulk within a moving window framework. This results in overlapping data loads, but it’s an effective strategy for handling sales data, as most use cases require access to the most recent two months. By using this rolling window method, the latest two months of data are consistently available on a daily basis.



Here, we simply attach the monthly partition tables to the existing partitioned table.

----Test that the procedure can be executed repeatedly with consistent results--------------------

--------------------------------------------------------------------------------------------------------------------------

---1.1.Run the procedure:

-----------------------------------------------------------------------------------------------

**CALL** bl\_dm.load\_fct\_sales\_rolling\_window(**DATE** '2021-01-05', **DATE** '2021-04-05',**TRUE**);---with partitions

**CALL** bl\_dm.load\_fct\_sales\_rolling\_window(**DATE** '2021-01-05', **DATE** '2021-04-05', **FALSE**);----no partitions

----------------------------------------------------------------------------------------------------------------------------------------

---1.2.Query the logging table for affected rows:

-------------------------------------------------------------------------------------------------------------------------------------------------

**SELECT** \*

**FROM** bl\_3nf.etl\_logs

**WHERE** procedure\_name **IN** (

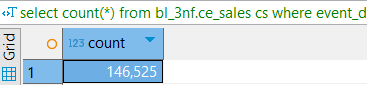
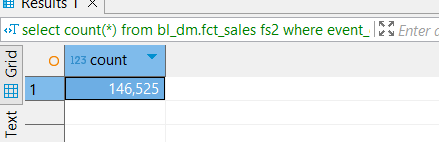
'sp\_insert\_fct\_sales\_incremental'

)

**order** **by** procedure\_name, log\_id ;

**select** **count**(\*) **from** bl\_3nf.ce\_sales *cs* **where** event\_dt **between** **DATE** '2021-01-05'**and** **DATE** '2021-04-05' ;

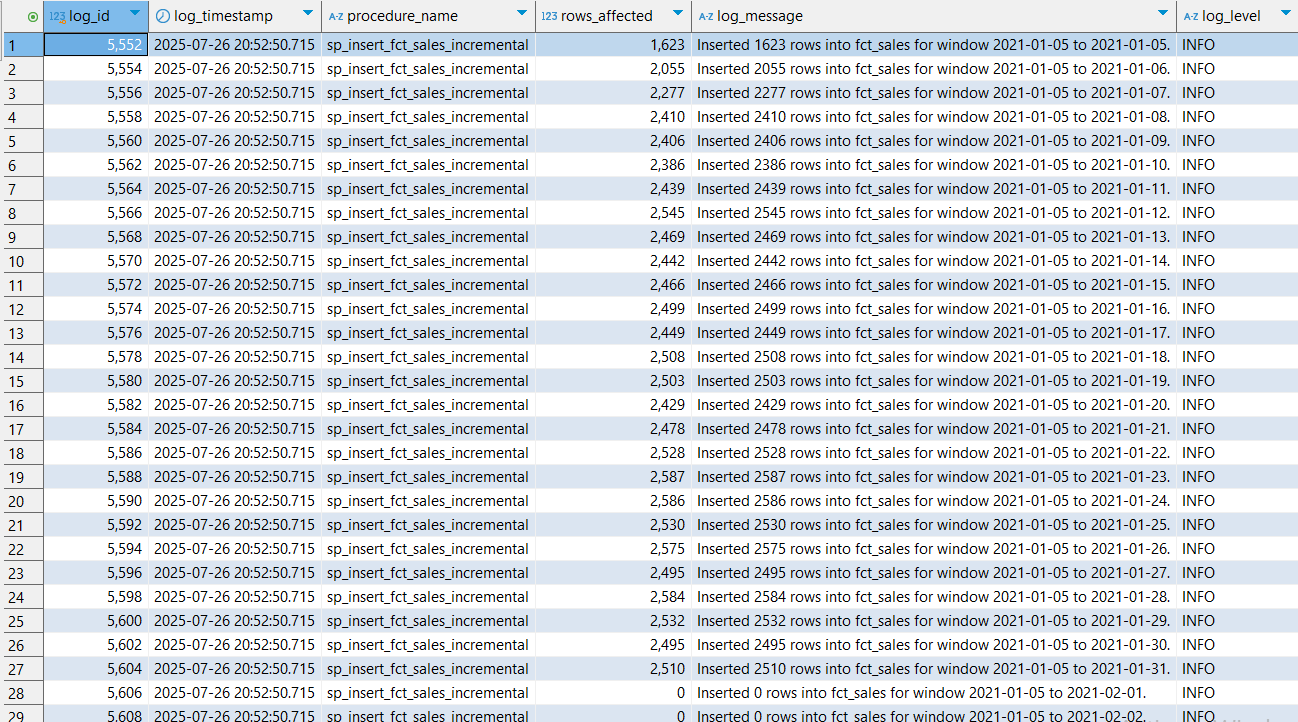
**select** **count**(\*) **from** bl\_dm.fct\_sales *fs2* **where** event\_dt **between** **DATE** '2021-01-05' **and** **DATE** '2021-04-05' ;

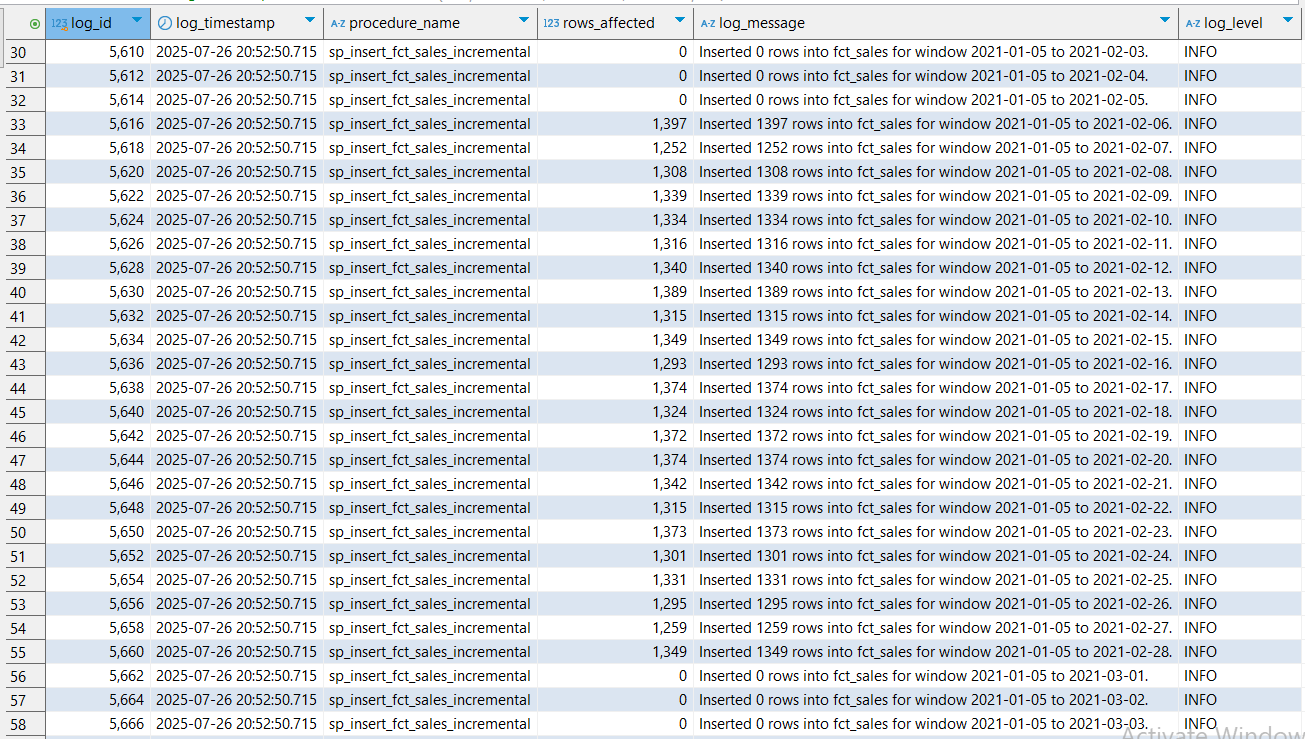


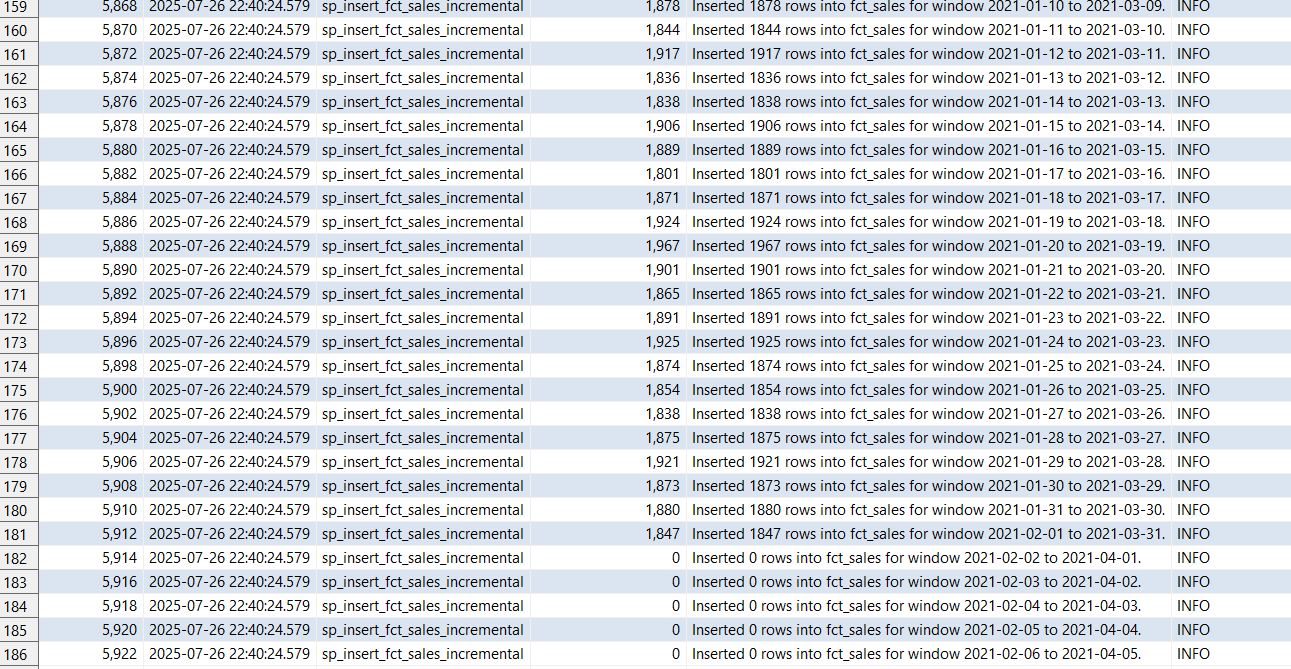
----------------------------------------------------------------------------------------------------------------------------------------

---1.3.Take a screenshot of the logging result

-----------------------------------------------------------------------------------------------------------------------------------------------







The core idea was to use a 2-month rolling window for bulk loading data into the DM layer from the 3NF layer, specifically targeting the sales fact table. I began by running a procedure I had previously developed to load three full fiscal months—January, February, and March—into the 3NF layer. Once all required data was available, I focused on partitioning the sales fact table in the DM layer by month, based on date ranges.

Next, I defined a procedure that partitions the table using specified start and end dates. Building on this, I created another procedure to handle bulk ingestion of 2-month intervals, rather than ingesting data on a daily basis.

We then implemented a process that moved this rolling window forward one day at a time, covering the period from January 5th, 2021, to April 5th, 2021. At each step, a new 2-month window was generated to insert the relevant data into the DM layer.

To validate the process, we compared the row counts between the 3NF and DM layers and confirmed they matched, indicating a successful load. Additionally, the total time to load the entire date range was just 1 minute and 9 seconds.

----------------------------------------------------------------------------------------------------------------------------------------

---1.4. Run the procedure again with the same input

--------------------------------------------------------------------------------------------------------------------------------------------------

**CALL** bl\_dm.load\_fct\_sales\_rolling\_window(**DATE** '2021-01-05', **DATE** '2021-04-05',**TRUE**);---with partitions

**CALL** bl\_dm.load\_fct\_sales\_rolling\_window(**DATE** '2021-01-05', **DATE** '2021-04-05', **FALSE**);----no partitions

----------------------------------------------------------------------------------------------------------------------------------------

---1.5. Query the login table again

--------------------------------------------------------------------------------------------------------------------------------------------------

**SELECT** \*

**FROM** bl\_3nf.etl\_logs

**WHERE** procedure\_name **IN** (

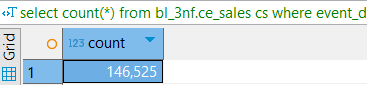
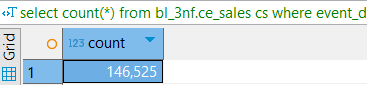
'sp\_insert\_fct\_sales\_incremental'

)

**order** **by** procedure\_name, log\_id ;

**select** **count**(\*) **from** bl\_3nf.ce\_sales *cs* **where** event\_dt **between** **DATE** '2021-02-04'**and** **DATE** '2021-02-27' ;

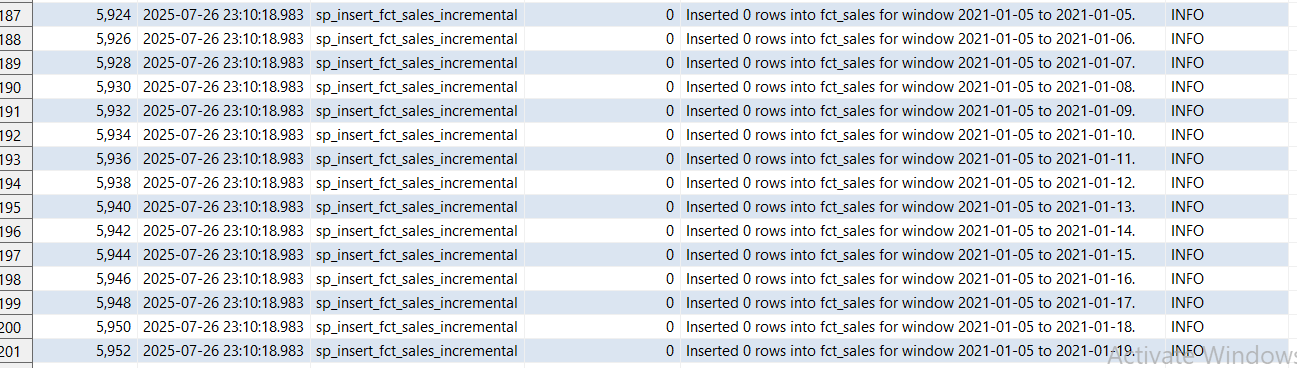
**select** **count**(\*) **from** bl\_dm.fct\_sales *fs2* **where** event\_dt **between** **DATE** '2021-02-04' **and** **DATE** '2021-02-27' ;

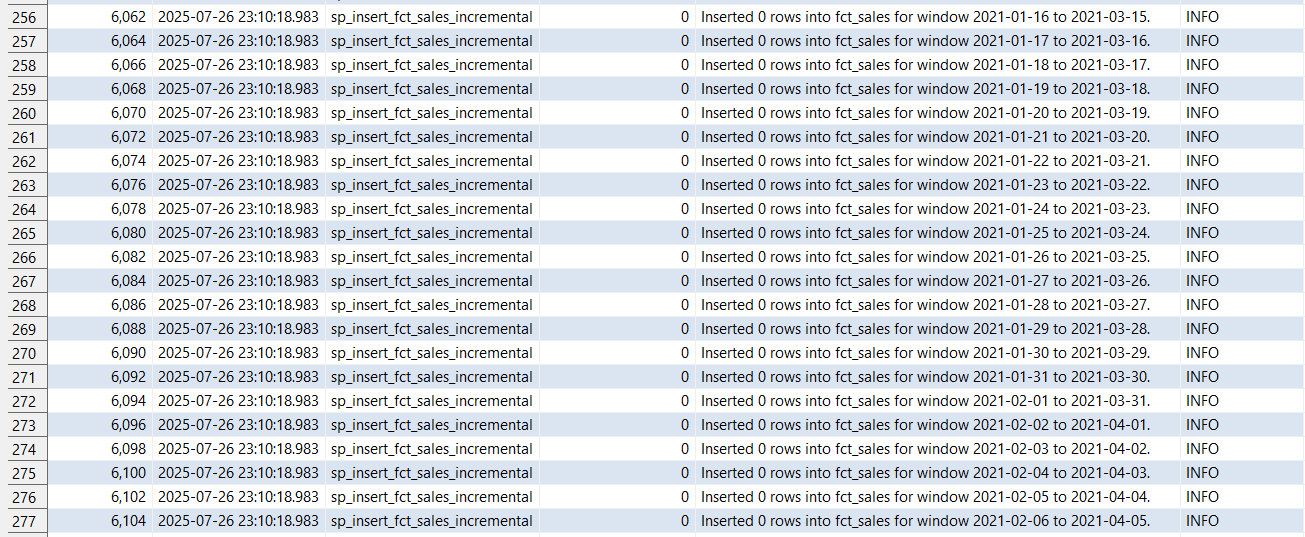


----------------------------------------------------------------------------------------------------------------------------------------

---1.6. Confirm number of rows affected is 0 (or explain why)

----------------------------------------------------------------------------------------------------------------------------------------





As expected, there were no new rows inserted during this run, since the sales table is treated as a static dimension—once a record is inserted, it remains unchanged. Therefore, when the procedure was executed again, PostgreSQL identified that all entries already existed and skipped the inserts. This run took 1 minute and 12 seconds, primarily due to the system re-checking all existing records. However, since the table is currently partitioned by month, in the next step we will remove the partitions and evaluate how that impacts performance.

When running the procedures detaching the procedures, we could see against expectations a decrease in the running time, which may be because we have to attach a partition every time we decide on a new rolling window which may take a while to prepare even though the execution time may be lower after.

---------------------------------------------------------------------------------------------------------------------------------------

---2. Tests for nulls duplicates and missing values

----------------------------------------------------------------------------------------------------------------------------------------

