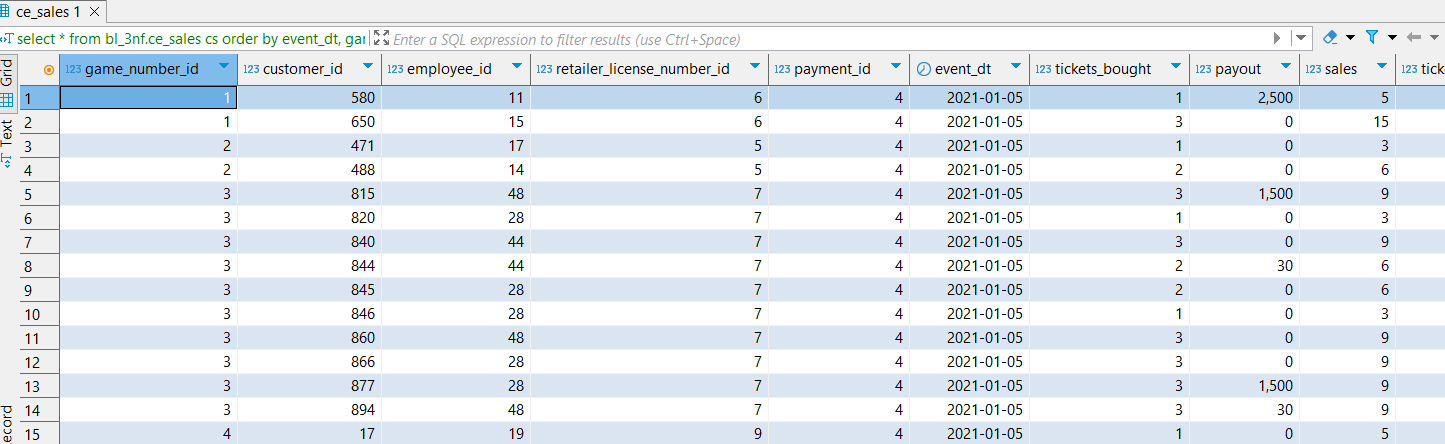


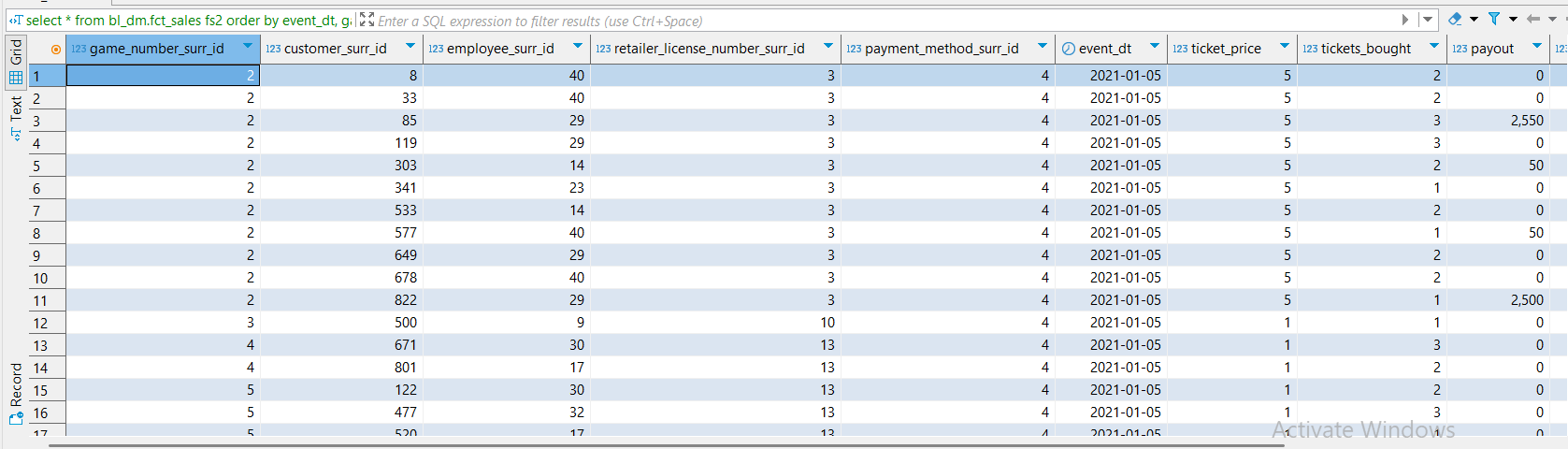
| Business Template  **Loading fact on 3NF / DM** |
| --- |
|  |

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

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

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





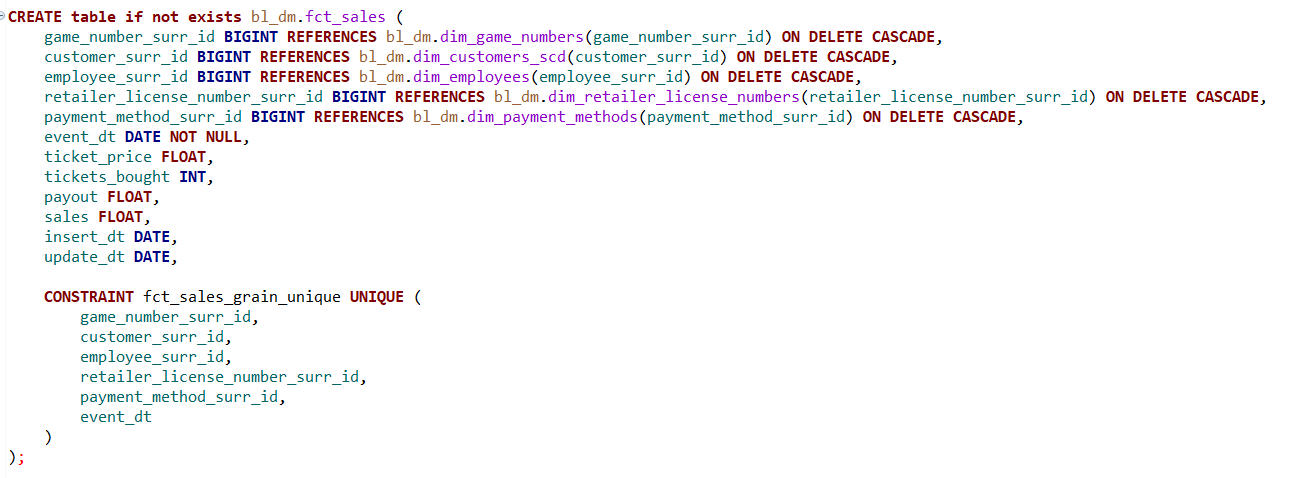
----------------------------------------------------------------------------------------------------------------------------------------

---B. Procedures

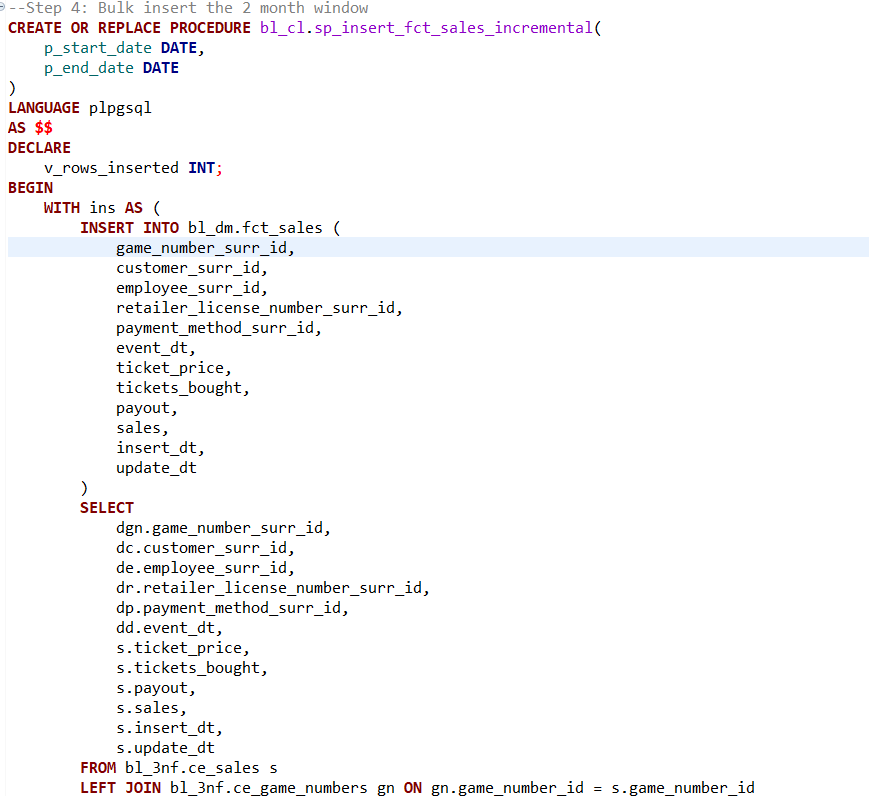
----------------------------------------------------------------------------------------------------------------------------------------

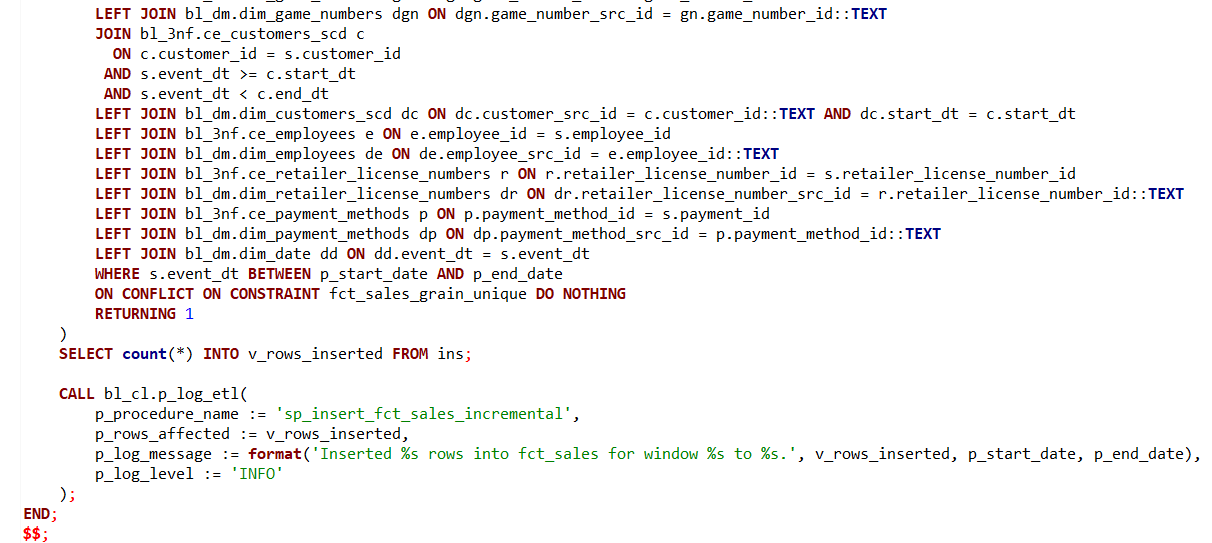


Here, we create a fact sales table partitioned by a range of event dates.

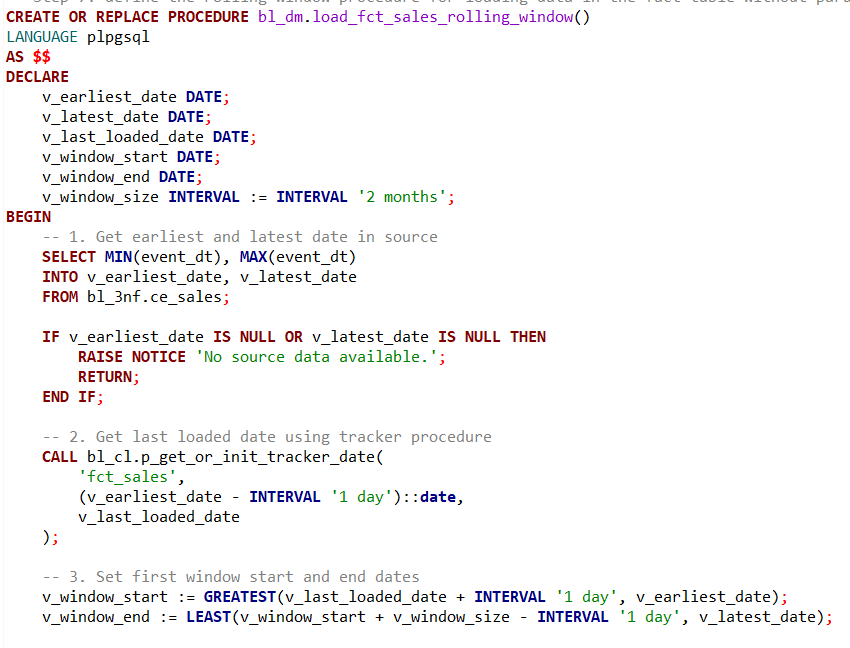
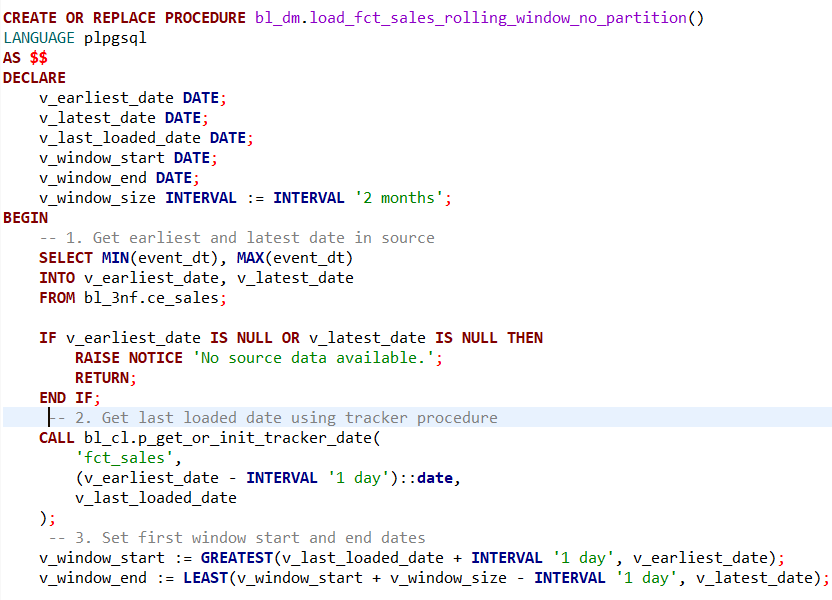


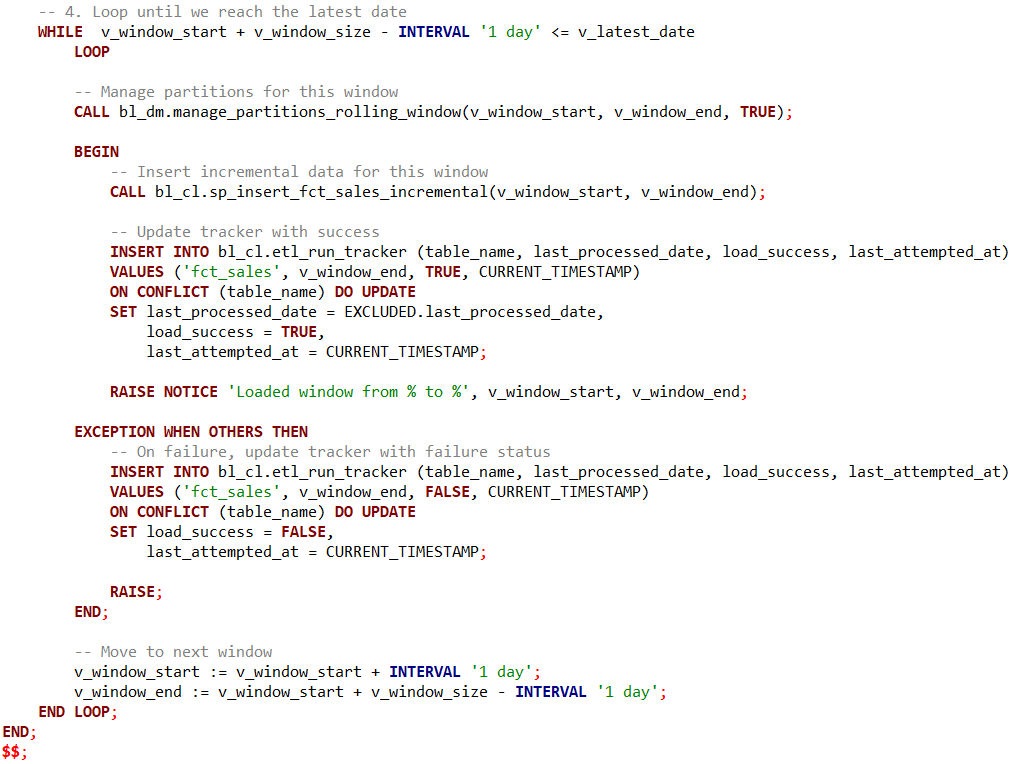
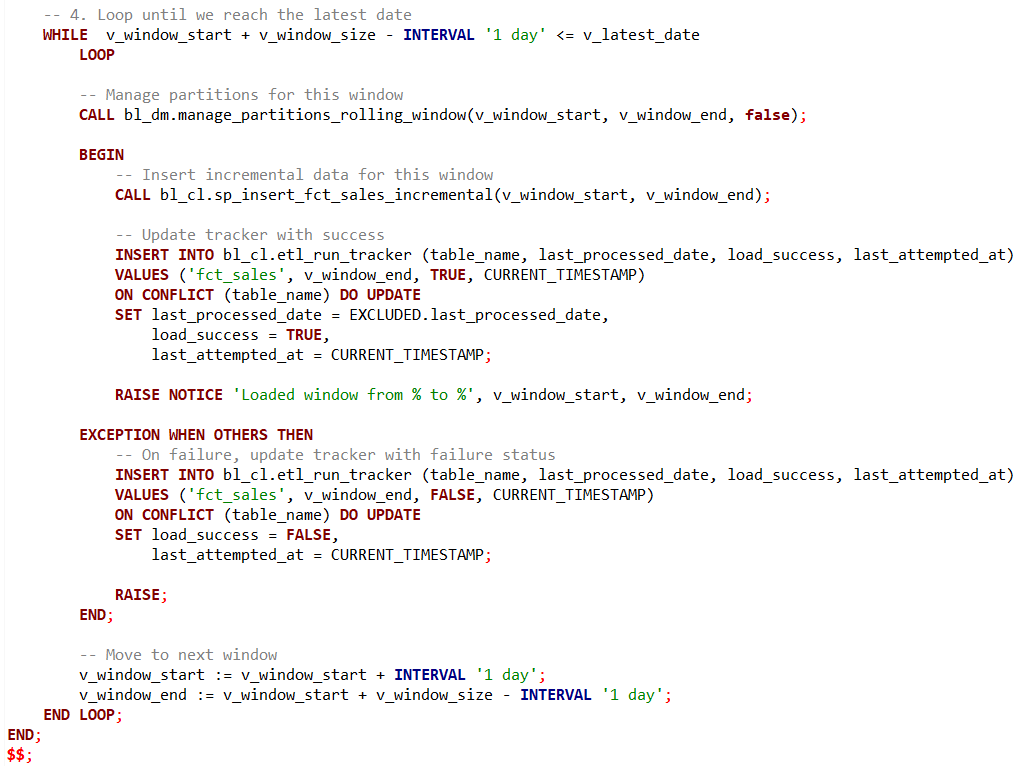
Here, we consider defining a fct sales table where we have no partition on.



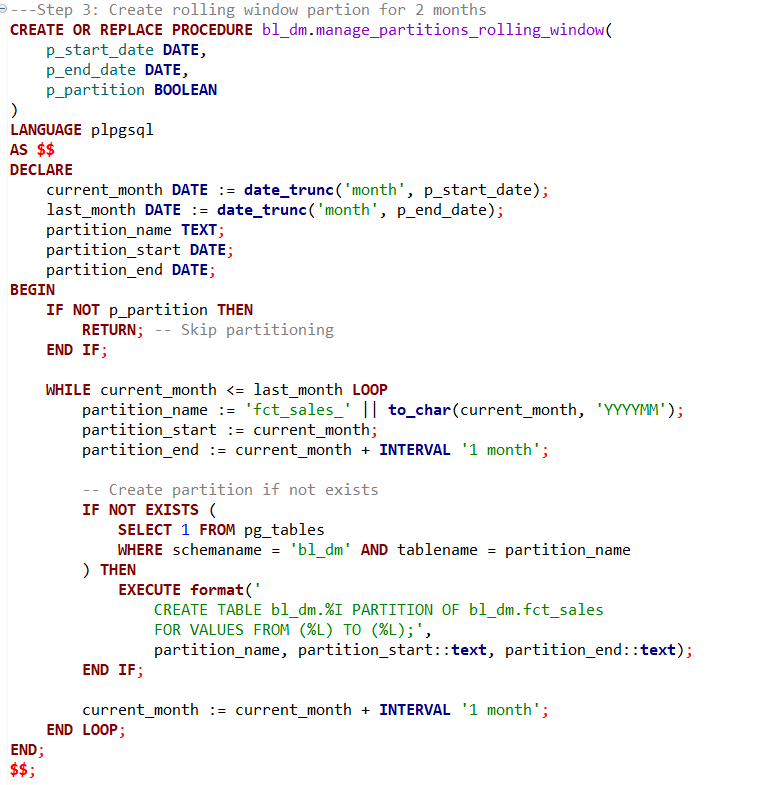


As in Task 8, we begin by ingesting data from the 3NF sales tables. However, this time we use bulk ingestion rather than daily ingestion. Instead of identifying active customers at the time of data insertion, we assign customer entries based on whether the event date falls between the customer's start\_dt 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.

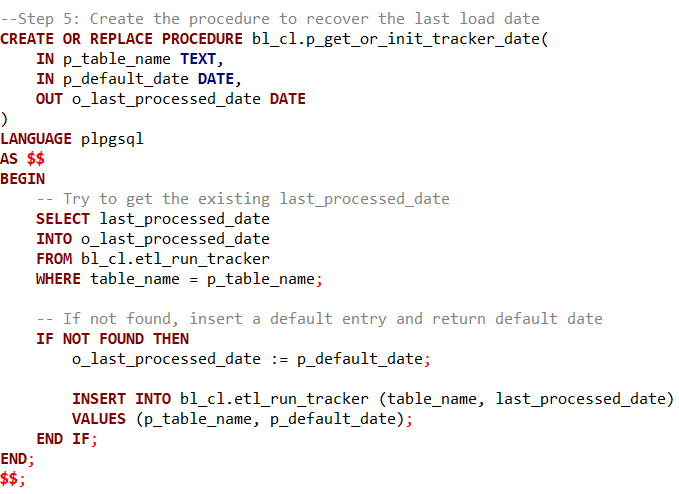
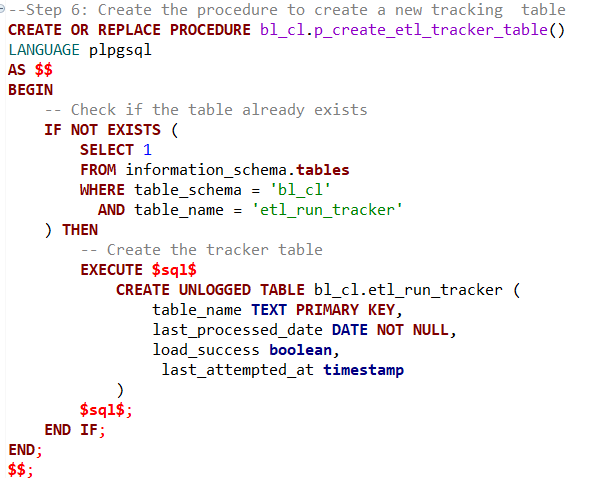




Here, we implement a rolling window approach, ingesting two months of data in bulk within a moving window framework. This results in overlapping data loads, but it’s an effective strategy for handling sales data, as most use cases require access to the most recent two months. By using this rolling window method, the latest two months of data are consistently available on a daily basis.(we have defined 2 different procedures, one that is using a partition and the other no partition)



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



We have defined a tracker table that records the data ingested into the sales fact table within the dimensional model (DM). The main purpose of this table is to ensure that only new data is loaded each time, excluding records that have already been inserted. This avoids redundant inserts and unnecessary joins. As a result, our rolling window data load procedure runs only when the latest load date in the tracker is earlier than the data available in the prefact tables we load from.

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

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

---1.1.Run the procedure:

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

**CALL** bl\_dm.load\_fct\_sales\_rolling\_window();

**CALL**  bl\_dm.load\_fct\_sales\_rolling\_window\_no\_partition();

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

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

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

**SELECT** \*

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

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

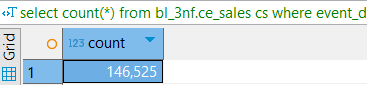
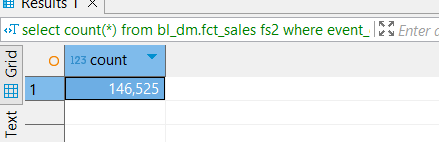
'sp\_insert\_fct\_sales\_incremental'

)

**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' ;

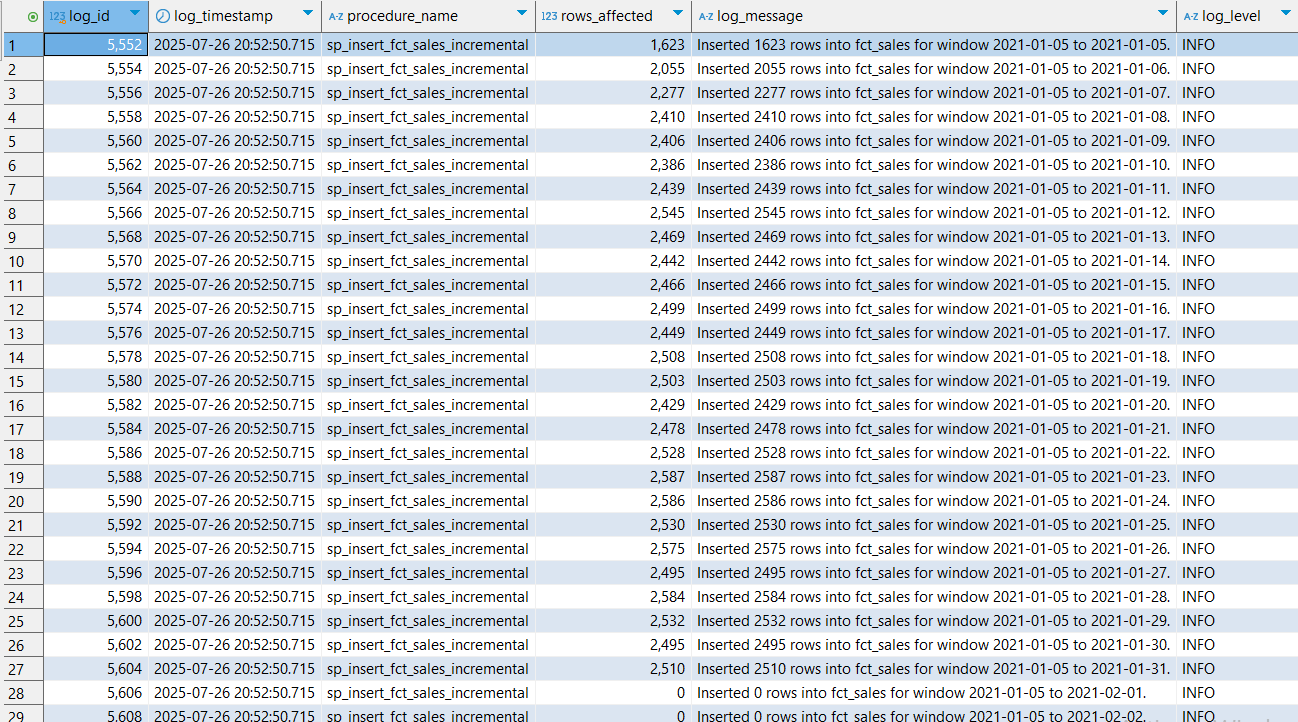
**select** **count**(\*) **from** bl\_dm.fct\_sales *fs2* **where** event\_dt **between** **DATE** '2021-01-05' **and** **DATE** '2021-04-05' ;

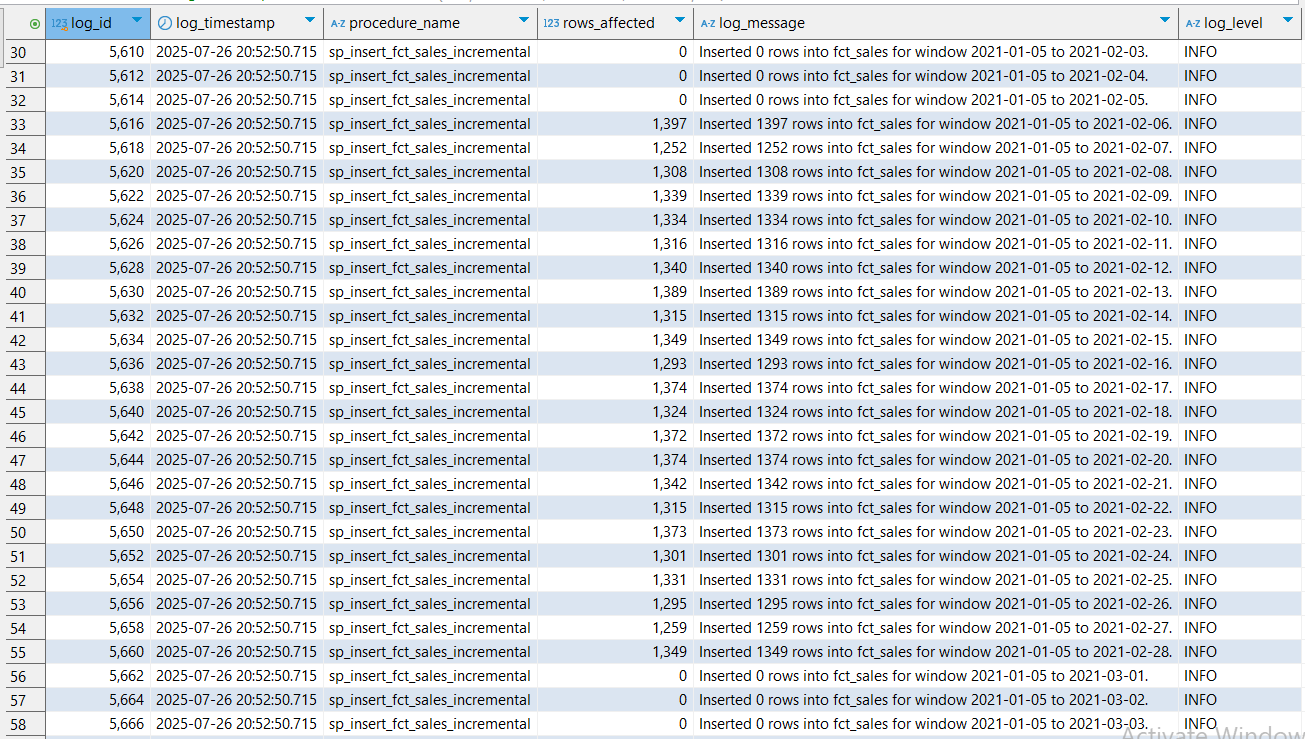


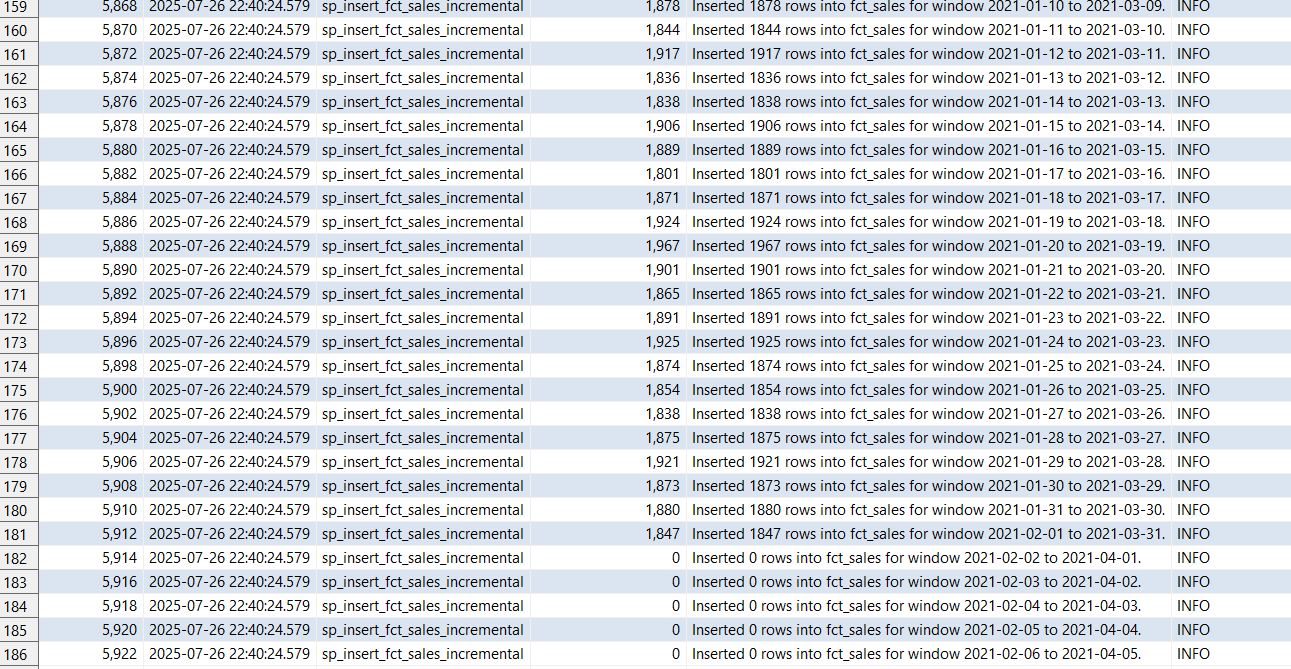
----------------------------------------------------------------------------------------------------------------------------------------

---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 entire period from the incremented sales table (in our case a quarter of data 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 24 seconds.

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

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

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

**CALL** bl\_dm.load\_fct\_sales\_rolling\_window();

**CALL**  bl\_dm.load\_fct\_sales\_rolling\_window\_no\_partition();

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

---1.5. Query the login table again

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

**SELECT** \*

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

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

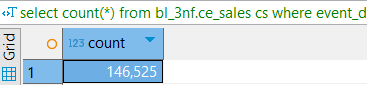
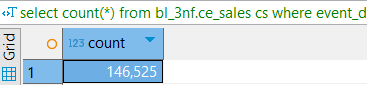
'sp\_insert\_fct\_sales\_incremental'

)

**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' ;

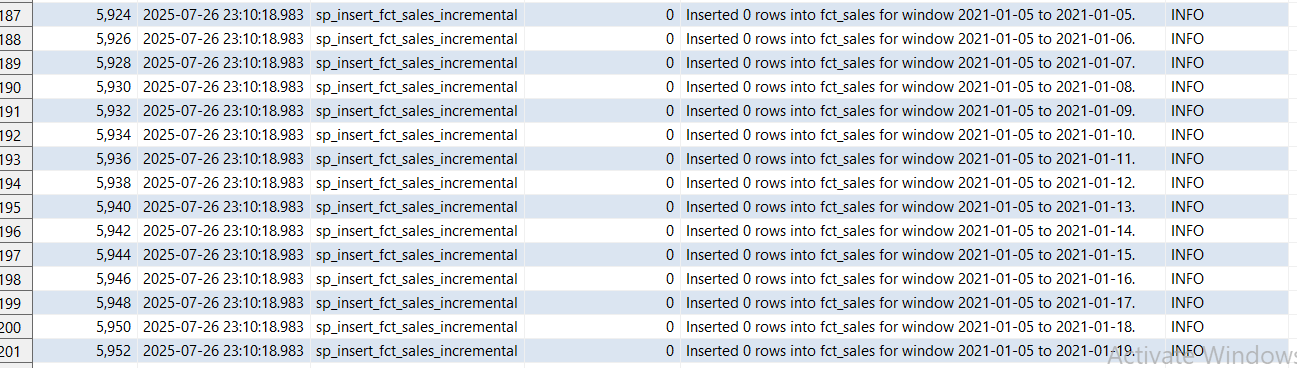
**select** **count**(\*) **from** bl\_dm.fct\_sales *fs2* **where** event\_dt **between** **DATE** '2021-02-04' **and** **DATE** '2021-02-27' ;

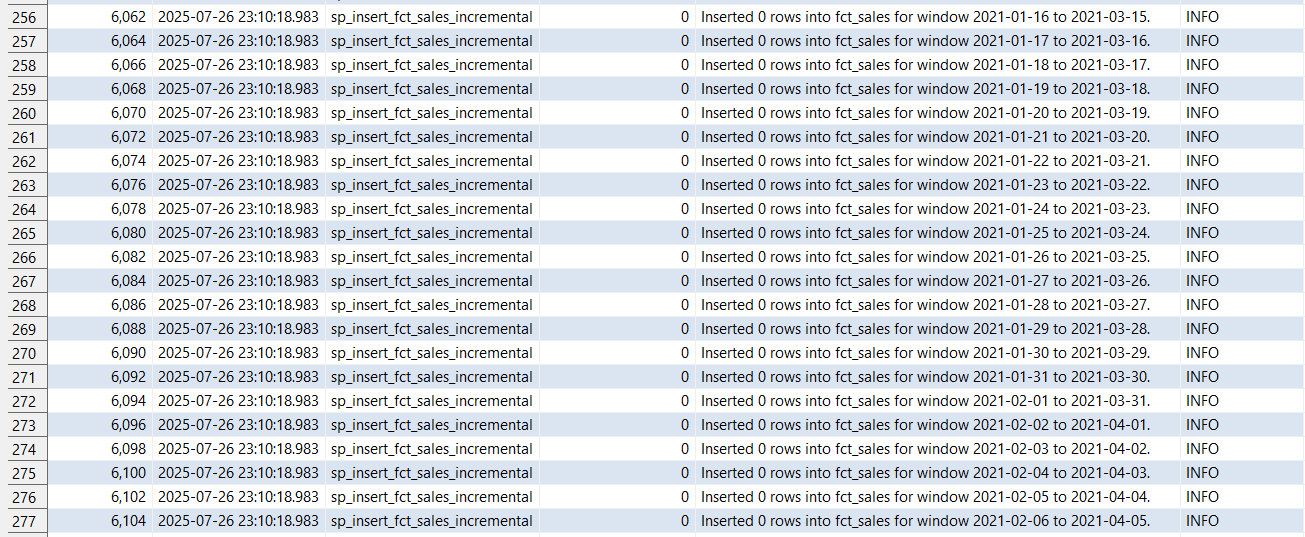


----------------------------------------------------------------------------------------------------------------------------------------

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

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





As expected, no new rows were inserted during this run because the sales table behaves like a static dimension—records are only inserted once and never updated. When the procedure was executed again, PostgreSQL detected that all rows already existed and skipped the insertions. The run completed in 0 seconds, largely because the system recognized that the last load date had already been processed, causing the procedure to exit early.

However, since the table is currently partitioned by month, the next step will be to remove the partitions and assess the impact on performance.

When running the procedures without partitioning, we observed that the execution time remained roughly the same, contrary to expectations. This may be due to the fact that the dataset only spans three months, in which case monthly partitioning offers little to no performance benefit.

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

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

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

