Questions and Answers

1. Is there a difference in the number of rows updated when comparing the 1st run of the script against the 2nd run of the script. If your answer is yes, what do you think caused this?

Answer:

Yes, there are differences in the number of rows updated in each run. This happened because in the 1st run, rows after 17,500 records were not updated and instead forcefully rolled back. In the 2nd run, the rows that were updated in the 1st run, were left out and remaining 8,376 rows were updated.

2. Is there a difference in the number of rows updated by SQL that simply updates the table vs. the number of rows updated by the part that uses the CURSOR when you total the 2 runs together? If so, what is the reason for this?

Answer:
The answer to this question is the same as one mentioned above. The first run updated 17,500 rows and 8,376 rows were updated in the second run. If we add them up it gives us the total 25,869 rows that are common between Salaries and Appearances table.

3. Why is there a difference in the number of cursor rows in the first and second run?

Answer:
The reason for having different number of cursor rows in the first and second run is because the number of rows that were updated in the first run (17,500) were not considered for the second Open UpdateCursor statement, which had 8,376 rows to update, while first time it had all 25,869 rows.

4. Calculate the number of rows updated per second for the DML and the Transaction script?

Answer:
For the DML it took almost 1.95 e^{-5} seconds and for the transaction script it took approximately 5.7 e^{-5} seconds.

5. What are some of the reasons the transaction script takes longer?

Answer:
Some of the reasons for the transaction script taking longer are the update queries, commits and rollback, along with various validation checks.

6. What happened the second time the Transaction part of the script was run?

Answer:
On the second run the Cursor omits the data that were already updated in the first run and proceeds processing remaining 8,376 data.

7. What does the cursor function do? Note: this should not be a copy/paste from the internet. The answer needs to be specific to the script.

Answer:
The purpose of this cursor is to select the players that are present in the Appearances table as well as Salaries table and do not have their records updated on a current day. These records are fetched by the cursor one at a time and updated accordingly. However, the cursor COMMITs after every 100 records processed and ROLLBACKs if the number reaches 17,543

8. Assuming the DML script abended at some point in the update process. What would you have to do to make sure the data was not updated twice when it is rerun?

Answer:
In general, if there were to be any abends in between – we should check the row at which the abend occurred, and restart processing transactions after the last commit. However, with this cursor since the last_update check is

already present, even if we restart cursor we would not be overriding data as updated rows would never be selected the next time.

9. When you run the scripts twice, is the data in the DML script updated appropriately? Explain your answer. Note: you can examine the contents of the JM234_Last_Update, JM234_SQL_Last_Update and M234_SQL_COUNT, and jm234_CURSOR_Count columns in the Appearances table to help with your answer.

Answer:
Yes, it did. Even in the second time the DML script updated 25,869 rows – the JM234_SQL_Count column is proof of that as it shows 2 for every tuple and their sum totals 51,738 which is the exact double of 25,869.

10. What happens if you run the script a 3rd time? What caused the results to be different if they were?

Answer:
While running the script the 3rd time returns 0 rows to be updated upon OPEN Cursor being executed. However, the JM234_SQL_Count column gets updated a third time – and its sum reflects 77,607(as every row now contains the value 3).

As for the transaction part of the script, it throws an ERROR stating The COMMIT TRANSACTION request has no corresponding BEGIN TRANSACTION. The reason being that the COMMIT this time was the first and last, although no rows were updated.

11. What do the nocount and statistics setting do? Why was it helpful to turn them on and off at specific points in the script.

Answer:
The nocount setting allows us to view the number of rows updated after every statement is processed, while statistics setting gives the time stats for every statement. They were helpful for the DML processing but turning them off before the transaction started was wise to avoid numerous updates being thrown for every run.

12. Explain the sequence of the record counts in the Transaction part of both runs of the scripts.

Answer:

The sequence of record counts are as follows:

a. **Number of Cursor rows after Declare:** They are always 0 as no logical cursor table is built during or after declaration.
b. **Number of Cursor rows:** In the first run all 25,869 rows were selected in the logical cursor table, but for the second run 17,500 updated rows were not selected as their JM234_Last_Update dates were equal to current date and thus we had only 8,376 rows.
c. **Count_Cursor_Dates:** It gives the number of rows updated with date in JM234_Last_Update column without NULL value – the first time 17,500 rows being updated and in the second time all 25,869 rows being updated.
d. **Count_SQL_Dates:** Every run has this column updated for all 25,869 rows. Hence its value is 25,869 all the time.
e. **Sum_SQL_Counter:** In the first run since the value for each row was updated first time, all values were 1 and thus total was 25,869 * 1. Whereas the second updated raised the value by 1 to 2 in each row and thus sum was (25,869 * 2) = 51,738.
f. **Sum_Cursor_Counter:** It gives number of rows updated with value in JM234_Cursor_Count column – the first time 17,500 rows being updated and in the second time all 25,869 rows being updated.
g. **Rollback started For Transaction at Record:** This message is encountered in the first run, due to the forceful Abend set at 17,543rd record. Thus, all records updated after 17,500 records being committed in the first run were rolled back. There was no final commit in this run.
h. **Final Commit Transaction for Record:** This value was encountered in the second run as the cursor table had only 8,376 rows. Thus, after the commit of 8,300 rows the final commit occurs when the final 8,376th row is updated in the second run.

- By pg395