Implementing Data Filtering and Manipulation Part II

Practice Overview

In this practice you will use Oracle GoldenGate built-in functions for data filtering and manipulation. Specifically, you will implement examples to do the following:

- Set up a transaction history table in the target database
- Use the Oracle GoldenGate `IF` function, string and date built-in functions
- Use the `SQLEXEC` function

Implement Data Filtering and Manipulation

A. Set up a transaction history table in the target database

In this is section of the practice, you will set up a history table to store the DML changes made on the EMPLOYEES table. This can practically be useful for auditing purposes.

1. Stop the Extract and the Replicat processes.

2. Open the Extract parameter file and add the following parameters.

`LOGALLSUPCOLS` parameter instructs the Extract to fetch the before value of the supplementally logged columns. This is the default behavior in GoldenGate 12.2. GoldenGate of version earlier than 12c uses the parameter `GETUPDATEBEFORES` instead.

In GoldenGate before 12c, by default, an UPDATE operation in the database results in two records in the trail file. One record for the After image and one record for the Before image data. Even the statistics in the GoldenGate utilities considered those records as two UPDATE operations.

In GoldenGate 12c, the database UPDATE statement results in a single unified trail record. This enhances the performance of the GoldenGate processes and reduces the storage needed by the trail files. The parameter `UPDATERECORDFORMAT COMPACT` parameter sets this functionality on and it is the default behavior in GoldenGate release 12.2.

```
LOGALLSUPCOLS
UPDATERECORDFORMAT COMPACT
```

3. In the target database (`db2`), create the transaction history table.

For sake of simplicity, this example keeps track of the changes on only some of the table columns.

```
CREATE TABLE HRTRG.EMP_HISTORY AS
SELECT EMPLOYEE_ID, SALARY, COMMISSION_PCT FROM EMPLOYEES;

ALTER TABLE HRTRG.EMP_HISTORY ADD (
       OP_TYPE      VARCHAR2(15),
       TRAN_TIME    TIMESTAMP,
       BEFORE_AFTER VARCHAR2(8));
```

4. In the Replicat parameter file, add the following code.

You will enhance this functionality in a future practice lecture.

Notice that for MAP parameter, when its value span multiple lines, you do not need to terminate every line with an ampersand symbol. The same is true about TABLE parameter. All other parameters require to terminate their lines with ampersand when they are assigned multi-line values.

```
Map HR.EMPLOYEES, Target HRTRG.EMP_HISTORY, INSERTALLRECORDS,
 ColMap (USEDEFAULTS,
         OP_TYPE = @GetEnv('GGHEADER', 'OPTYPE'),
         TRAN_TIME = @GetEnv('GGHEADER', 'COMMITTIMESTAMP'),
         BEFORE_AFTER = @GETENV ('GGHEADER', 'BEFOREAFTERINDICATOR'));
```

5. Start the Extract and Replicat processes.

6. Test the functionality for an insert operation.

   a. In db1, insert a new record in the EMPLOYEES table

```
INSERT INTO EMPLOYEES (EMPLOYEE_ID, FIRST_NAME, LAST_NAME, EMAIL, HIRE_DATE,
SALARY, JOB_ID) VALUES (222, 'Rrd Name', 'Family Name', 'test@ahmedbaraka.com',
TO_DATE('01-01-2000','DD-MM-YYYY'),10000, 'IT_PROG');
COMMIT;
```

   b. In db2, verify that the insert has been replicated.

```
SELECT count(*) FROM HRTRG.EMPLOYEES WHERE EMPLOYEE_ID=222;
```

   c. In db2, verify that the history table has been updated.

```
SELECT SALARY, to_char(TRAN_TIME,'DD-MON-RR HH24:MI:SS') TRAN_TIME, OP_TYPE,
BEFORE_AFTER from HRTRG.EMP_HISTORY
WHERE EMPLOYEE_ID=222 ORDER BY TRAN_TIME DESC;
```

7. Test the functionality for an update operation.

   a. In db1, update a record in the EMPLOYEES table

```
UPDATE EMPLOYEES SET SALARY=25000 WHERE EMPLOYEE_ID=222;
COMMIT;
```

   b. In db2, verify that the update has been replicated.

```
SELECT SALARY FROM HRTRG.EMPLOYEES WHERE EMPLOYEE_ID=222;
```

   c. In db2, verify that the history table has been updated.

```
SELECT SALARY, to_char(TRAN_TIME,'DD-MON-RR HH24:MI:SS') TRAN_TIME, OP_TYPE,
BEFORE_AFTER from HRTRG.EMP_HISTORY
WHERE EMPLOYEE_ID=222 ORDER BY TRAN_TIME DESC;
```

8. Test the functionality for a delete operation.

   a. In db1, delete the inserted record from the EMPLOYEES table

```
DELETE EMPLOYEES WHERE EMPLOYEE_ID=222;
COMMIT;
```

   b. In db2, verify that the record has been deleted.

```
SELECT * FROM HRTRG.EMPLOYEES WHERE EMPLOYEE_ID=222;
```

   c. In db2, verify that the history table has been updated.

   Notice that there is no After values for delete operations. Therefore, the value of BEFORE_AFTER is BEFORE for the delete operation record.

```
SELECT SALARY, to_char(TRAN_TIME,'DD-MON-RR HH24:MI:SS') TRAN_TIME, OP_TYPE,
BEFORE_AFTER from HRTRG.EMP_HISTORY
WHERE EMPLOYEE_ID=222 ORDER BY TRAN_TIME DESC;
```

B. Using IF function

In this section of the practice, you will add an indicator column to the target EMPLOYEES table called IT_FLAG. This column takes the value ‘Y’, if the title of the employee is IT related, otherwise it will take the ‘N’ value.

9. Stop the Extract and the Replicat.

10. In `db2`, add a column to the EMPLOYEES table, as follows.

```
ALTER TABLE HRTRG.EMPLOYEES ADD ( IT_JOB_FLAG  VARCHAR2(1));
```

11. Alter the MAP statement in the Replicat as follows.

Observe that because the condition in the IF function is string comparison, you use a string function. Using the equal sign is not acceptable.

```
Map HR.EMPLOYEES, Target HRTRG.EMPLOYEES, ColMap (USEDEFAULTS, IT_JOB_FLAG = @IF
(@STREQ (JOB_ID, 'IT_PROG'), 'Y', 'N') );
```

12. Start the Extract and the Replicat

13. Test the new added mapping.

a. In `db1`, execute the following update statement.

```
UPDATE HR.EMPLOYEES SET JOB_ID='IT_PROG' WHERE EMPLOYEE_ID=103;
COMMIT;
```

b. Verify the flag column has been successfully updated.

It may take a few seconds before you see the update reflected in the target database. If you still do not see the record updated, it is usually an indication that the Replicat has an issue. As always is the case in troubleshooting scenarios, check the status of the Replicat and the `ggserr.log` file.

```
SELECT JOB_ID, IT_JOB_FLAG FROM HRTRG.EMPLOYEES WHERE EMPLOYEE_ID=103;
```

C. Using Date functions

In this section of the practice, you will add a column to the EMPLOYEES table called WORKING_DAYS. This column stores the number of days since the employee join date.

14. Stop the Replicat.

15. In db2, add a column to the EMPLOYEES table, as follows.

```
ALTER TABLE HRTRG.EMPLOYEES ADD ( WORKING_DAYS NUMBER );
```

16. Modify the MAP statement in the Replicat as follows.

```
Map HR.EMPLOYEES, Target HRTRG.EMPLOYEES, ColMap (USEDEFAULTS, IT_JOB_FLAG = @IF
(@STREQ (JOB_ID, 'IT_PROG'), 'Y', 'N'), working_days = @DATEDIFF ('DD', HIRE_DATE,
@DATENOW ()) );
```

17. Start the Replicat process

18. Test the new added mapping.

a. In db1, execute the following update statement.

```
UPDATE EMPLOYEES SET HIRE_DATE='01-JAN-07' WHERE EMPLOYEE_ID=103;
COMMIT;
```

b. Verify the new column has been successfully updated.

```
SELECT  HIRE_DATE , ROUND(WORKING_DAYS/365,3) YEARS
FROM EMPLOYEES WHERE EMPLOYEE_ID=103;
```

D. Using the `SQLEXEC` functions

In this section of the practice, you will add a column to the target EMPLOYEES table called TITLE. This column stores the title of the employee based on the value of JOB_ID. You will use the `SQLExec` function to get the Title from the JOBS table in the target database.

19. Stop the Replicat.

20. In `db2`, add a column to the EMPLOYEES table, as follows.

```
ALTER TABLE HRTRG.EMPLOYEES ADD ( TITLE VARCHAR2(35));
```

21. Modify the MAP statement of the EMPLOYEES table in the Replicat as follows.

```
Map HR.EMPLOYEES, Target HRTRG.EMPLOYEES,
 SQLEXEC (id GET_TITLE, QUERY ' SELECT JOB_TITLE FROM HRTRG.JOBS
                                 WHERE JOB_ID = :V_JOB_ID ',
          PARAMS (V_JOB_ID = JOB_ID)),
 COLMAP (USEDEFAULTS, IT_JOB_FLAG = @IF (@STREQ (JOB_ID, 'IT_PROG'), 'Y', 'N'),
WORKING_DAYS = @DATEDIFF ('DD', HIRE_DATE, @DATENOW ()),
 TITLE = @GETVAL(GET_TITLE.JOB_TITLE));
```

22. Start the Replicat process

23. Test the new added mapping.

a. In `db1`, execute the following update statement.

```
UPDATE HR.EMPLOYEES SET JOB_ID='IT_PROG' WHERE EMPLOYEE_ID=105 ;
COMMIT;
```

b. Verify the new column has been successfully updated.

```
SELECT JOB_ID, IT_JOB_FLAG, TITLE FROM EMPLOYEES WHERE EMPLOYEE_ID=105 ;
```

E. Summary

Oracle GoldenGate offers a reliable filtering and manipulation tools to keep a history of the transactions applied on a source table. It also offers wide range of built-in functions to implement complex filtering and manipulation scenarios.

In this practice, you gained experience on creating a functionality to keep records of the transactions made on a source table. You also used the `IF` function, some date and string functions, and the `SQLEXEC` function.

There are plenty of functions and which one to use depends on your business needs. The technical details of using all the functions can be obtained from the documentation "*Reference for Oracle GoldenGate for Windows and UNIX*".

By the end of this practice, following are the code in each process parameter file:

```
Extract esrv1
REPORTROLLOVER AT 03:00 ON sunday
DISCARDROLLOVER AT 03:00 ON sunday
USERID ogg, PASSWORD oracle
ExtTrail ./dirdat/es
Table HR.JOB_HISTORY;
Table HR.EMPLOYEES;
Table HR.JOBS;
Table HR.DEPARTMENTS;
Table HR.LOCATIONS;
Table HR.REGIONS;
Table HR.SAMPLE;
```

```
Extract psrv1
RmtHost ggsrv2, MgrPort 7810
RmtTrail ./dirdat/rt
Passthru
Table HR.*;
```

```
Replicat rsrv2
REPORTROLLOVER AT 03:00 ON sunday
DISCARDROLLOVER AT 03:00 ON sunday
DiscardFile ./dirrpt/rsrv2.dsc, Purge
USERID ogg, PASSWORD oracle
Map HR.EMPLOYEES, Target HRTRG.EMPLOYEES,
SQLEXEC (id GET_TITLE, QUERY ' SELECT JOB_TITLE FROM HRTRG.JOBS
WHERE JOB_ID = :V_JOB_ID ',
PARAMS (V_JOB_ID = JOB_ID)),
COLMAP (USEDEFAULTS, IT_JOB_FLAG = @IF (@STREQ (JOB_ID, 'IT_PROG'), 'Y', 'N'),
WORKING_DAYS = @DATEDIFF ('D
D', HIRE_DATE, @DATENOW ()),
TITLE = @GETVAL(GET_TITLE.JOB_TITLE))
);
Map HR.EMPLOYEES, Target HRTRG.EMP_HISTORY, INSERTALLRECORDS,
ColMap (USEDEFAULTS,
OP_TYPE = @GetEnv('GGHEADER', 'OPTYPE'),
TRAN_TIME = @GetEnv('GGHEADER', 'COMMITTIMESTAMP'),
BEFORE_AFTER = @GETENV ('GGHEADER', 'BEFOREAFTERINDICATOR'));
Map HR.*, Target HRTRG.*;
```