Implementing Data Filtering and Manipulation Part I

Practice Overview

In this practice you will implement the data filtering and manipulation techniques in Oracle GoldenGate.

Specifically, you will perform the following:

- Select rows with a WHERE clause and make the data available
- Select rows with FILTER clause
- Implement column mapping using COLMAP clause

Troubleshooting Procedure

As you will make a lot of modifications in the parameter file, you may face a situation where a process abends or fails to start. The basic troubleshooting procedure is as follows:

1. Check the status of the process (Extract or Replicat).

```
ggsci> info all
```

2. If the status is **not** RUNNING, check out the `ggserr.log` file. You do not have to exit from the ggsci to view the contents of the file. You can issue either of the following commands:

```
ggsci> sh tail ggserr.log
ggsci> View GGSEvt
```

3. If no error in the error log file, view the report of the failing process.

Implement Data Filtering and Manipulation

A. Filter rows with a WHERE Clause

In this section, you will implement a filter on the Extract using the WHERE clause and test it. You will learn on a common mistake made by new GoldenGate learners when they use WHERE clause and learn how to void it.

1. Open the Extract parameter file and modify it as follows. Extract will capture the operations on only the employee of ID 120.

```
edit params esrv1

...
Table HR.EMPLOYEES WHERE (EMPLOYEE_ID=120);
...
```

2. Restart the Extract

```
stop e*
start e*
info all
```

3. Verify that the new modification has taken effect on the Extract process.

This command will display the values of the TABLE parameters taken by the Extract.

```
send esrv1 getparaminfo table
```

4. Test the filter.

a. In `db1`, retrieve the phone number of employee ID 120.

```
SELECT PHONE_NUMBER FROM EMPLOYEES WHERE EMPLOYEE_ID=120;
```

b. Change the phone number of the employee.

```
UPDATE EMPLOYEES SET PHONE_NUMBER='999.999.9999' WHERE EMPLOYEE_ID=120;
COMMIT;
```

c. Check out the same data in `db2`. You should see the phone number updated.

```
SELECT PHONE_NUMBER FROM EMPLOYEES WHERE EMPLOYEE_ID=120;
```

5. Modify the Extract WHERE clause so that it captures the changes only to the employees of DEPARTMENT_ID 50..

a. Open the Extract parameter file of the Extract and modify the following parameter.

```
Table HR.EMPLOYEES WHERE (DEPARTMENT_ID=50);
```

b. Restart the Extract

c. In `db1`, verify that the employee of ID 120 belongs to DEPARTMENT ID 50.

```
SELECT DEPARTMENT_ID FROM EMPLOYEES WHERE EMPLOYEE_ID=120;
```

d. In `db1`, change the phone number of the employee.

```
UPDATE EMPLOYEES SET PHONE_NUMBER='777.777.7777' WHERE EMPLOYEE_ID=120;
COMMIT;
```

e. Check out the same data in `db2`. You will notice the phone number is **not** updated.

```
SELECT PHONE_NUMBER FROM EMPLOYEES WHERE EMPLOYEE_ID=120;
```

f. Check the status of the Extract and Replicat and verify they are in RUNNING status.

g. Check the `ggserr.log` file and make sure no error reported.

h. **Question**: Can you describe why the change have not been applied on the target database?

Answer: because DEPARTMENT_ID value is not included in the redo log. The way you enabled the table-level supplemental logging for the table makes the database logs the primary key values and the updated columns but the not-updated column values will **not** be included. Therefore, the value of the DEPARTMENT_ID in your UPDATE statement will not be saved in the supplemental logging information.

Note: Replicat reacts similarly, if you refer to unavailable column value in your filter condition.

i. Modify the table-level supplemental options of the EMPLOYEES table so that the DEPARTMENT_ID is included in the logging data.

ALLCOLS could be replaced with COLS (DEPARTMENT_ID). ALLCOLS is used in case you may need to refer to values of other columns.

```
DBLOGIN userid ogg, password oracle
Info TranData HR.EMPLOYEES
Add TranData HR.EMPLOYEES ALLCOLS
```

j. Restart the Extract

k. Change the phone number of the employee.

```
UPDATE EMPLOYEES SET PHONE_NUMBER='777.777.7777' WHERE EMPLOYEE_ID=120;
COMMIT;
```

l. Check out the same data in `db2`. You will notice the update has got replicated this time.

```
SELECT PHONE_NUMBER FROM EMPLOYEES WHERE EMPLOYEE_ID=120;
```

B. Filter rows with a FILTER Clause

FILTER clause offers more advanced options for filtering data than the WHERE clause. In this section of the practice, you will gain experience on using the FILTER clause.

6. In the target system (`db2`), open the Replicat parameter file and add the parameter as follows. Replicat will apply the changes on the EMPLOYEES table only if the SALARY value is greater than or equal to 3000.

Note: when setting up explicit and wildcard MAP statements that interfere, the order is important. Always put the wildcard MAP statements in the last. This way, the explicit MAP statements will be excluded from the wildcard MAP statement. In such a case, GoldenGate will report the warning message OGG-02081.

```
edit params rsrv2

...
Map HR.EMPLOYEES, Target HRTRG.EMPLOYEES, Filter ( @COMPUTE (SALARY / 1000) >= 3);
Map HR.*, Target HRTRG.*;
```

7. Restart the Replicat.

8. Remove the WHERE clause option from the EMPLOYEES table setting in the Extract parameter file.

9. Restart the Extract.

10. Verify that the SALARY for the EMPLOYEE_ID 150 is greater than 3000.

```
SELECT SALARY , PHONE_NUMBER FROM HR.EMPLOYEES WHERE EMPLOYEE_ID=150;
```

11. In `db1`, update the PHONE_NUMBER of that employee.

```
UPDATE HR.EMPLOYEES SET PHONE_NUMBER = '123.4567' WHERE EMPLOYEE_ID=150;
COMMIT;
```

12. In `db2`, verify the update has been applied.

Notice that although the SALARY column was not included in the UPDATE statement, the filter found its value and operation got replicated.

```
SELECT SALARY , PHONE_NUMBER FROM HRTRG.EMPLOYEES WHERE EMPLOYEE_ID=150;
```

C. Implement column mapping using COLMAP clause

In this section of the practice, you will implement and test the column mapping functionality. You will add a column to the target table and use the Replicat to populate it using the COLMAP clause.

13. In the target system, add the following column to the EMPLOYEES table.

This modification makes the table structure of the EMPLOYEES table in the target database different from its corresponding table in the source database. If you were working with GoldenGate earlier than release 12.2, you should generate *source definitions* file in the source database and copy it to the target database. As you are in this practice using the *self-describing* trail files, which is the default format in GoldenGate release 12.2, the definition of the table is saved in the trail file itself and, thus, there is no need to generate the source definitions file.

```
ALTER TABLE HRTRG.EMPLOYEES ADD (FULL_NAME VARCHAR2(50));
```

14. Modify the parameter file of the Replicat process as follows:

You will learn about the GoldenGate built-in functions (like the StrCat used in the code underneath) in a future lecture.

```
Map HR.EMPLOYEES, Target HRTRG.EMPLOYEES, ColMap (USEDEFAULTS,  FULL_NAME = @StrCat(
LAST_NAME,',',FIRST_NAME));
```

15. Restart the Extract

16. Restart the Replicat

17. In `db1`, update the PHONE_NUMBER of the employee 150 again.

```
UPDATE HR.EMPLOYEES SET PHONE_NUMBER = '123.4567', LAST_NAME='New last name' WHERE
EMPLOYEE_ID=150;
COMMIT;
```

18. In `db2`, check the value of the new column.

Note: It is noted that in this example, FULL_NAME has actually taken the *before* value of the LAST_NAME and FIRST_NAME columns rather than the updated value. This is not the expected behavior and it should be reported to Oracle support.

Thanks to the student “Amit” who has observed this issue.

```
SELECT LAST_NAME, FULL_NAME FROM EMPLOYEES WHERE EMPLOYEE_ID=150;
```

Summary

In this practice you implemented some data filtering and manipulation techniques in Oracle GoldenGate. Specifically, you learnt how to do the following:

- Select rows with a WHERE clause
- Select rows using the FILTER clause
- Implement column mapping using COLMAP clause

You will learn more filtering options in the next practice lecture.