Practice

Automating Tasks with the Scheduler

Practice Target

In this practice, you will implement the common tasks included in using the Scheduler in Oracle databases.

Practice Overview

In this practice, you will perform the following tasks:

- Create a Scheduler job that prepares a dashboard report
- Re-create the same Scheduler job using a program and a schedule
- Monitor running jobs and Scheduler job logs

Assumptions

- This practice assumes that `srv1` is up and running from the **CDB** snapshot.

A. Creating a Scheduler Job

In this section of the practice, you will create a Scheduler job that reads the ORDERS table, calculates the sales figures for the last month, and saves the results in a table. This table is to be read by a dashboard report in a BI system.

1. Open Putty and connect to srv1 as oracle.

2. Invoke SQL*Plus and login to PDB1 as SOE

```
sqlplus soe/ABcd##1234@PDB1
```

3. Verify that SOE has the privilege to create a scheduler job.

   If the privilege is not granted, you need to grant it before the user can create a Scheduler job.

```
SELECT COUNT(*) CNT FROM USER_SYS_PRIVS WHERE PRIVILEGE='CREATE JOB';
```

4. Create the table where the dashboard report data is to be stored in.

```
CREATE TABLE SOE.SALES_RPT
( ROW_ID NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY PRIMARY KEY,
  INSERT_DATE DATE,
  RMONTH NUMBER(2),
  RYEAR NUMBER(4),
  AMOUNT NUMBER(12,2),
  TOTAL NUMBER(7)
);
```

5. Create a PL/SQL procedure that populates the created table.

   For easy system maintenance, it is recommended to create PL/SQL procedures and functions in packages.

```
CREATE OR REPLACE PACKAGE MYPKG AS
 PROCEDURE POPULATE_SALES_RPT;
END;
/

CREATE OR REPLACE PACKAGE BODY MYPKG AS
PROCEDURE POPULATE_SALES_RPT
AS
BEGIN
 INSERT INTO SALES_RPT
 SELECT NULL, SYSDATE,
  EXTRACT ( MONTH FROM SYSDATE-1 ),
  EXTRACT ( YEAR FROM SYSDATE-1 ),
  SUM(ORDER_TOTAL), COUNT(ORDER_ID)
 FROM ORDERS
 WHERE ORDER_STATUS < 5
   AND TO_CHAR(ORDER_DATE,'MM-YYYY') = TO_CHAR(SYSDATE-1,'MM-YYYY');
COMMIT;
END;
END;
/
```

6. Create a Scheduler job that populates the created job in the beginning of each month. It does its calculation for the past month.

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB(
 JOB_NAME => 'POPULATE_SALES_RPT',
 JOB_TYPE => 'PLSQL_BLOCK',
 JOB_ACTION => 'MYPKG.POPULATE_SALES_RPT;',
 START_DATE => SYSDATE,
 REPEAT_INTERVAL => 'FREQ=MONTHLY;INTERVAL=1;BYHOUR=0;BYMINUTE=0;BYSECOND=10',
 END_DATE => NULL,
 COMMENTS => 'Populate the monthly sales into the sales dashboard report');
END;
/
```

7. Submit the following query to retrieve some basic information about the created job.

   Observe that the job is disabled. By default, newly created Scheduler jobs are disabled. We can change this behavior by setting the parameter ENABLED to TRUE in the DBMS_SCHEDULER.CREATE_JOB procedure.

```
col NEXT_RUN_DATE for a20
SELECT ENABLED, STATE, RUN_COUNT, NEXT_RUN_DATE FROM USER_SCHEDULER_JOBS WHERE
JOB_NAME='POPULATE_SALES_RPT';
```

8. Enable the job.

```
exec DBMS_SCHEDULER.ENABLE('POPULATE_SALES_RPT')
```

9. Submit the following query to retrieve information about the job after enabling it.

   Observe that the job is enabled now and is scheduled to run at midnight in the same day in the next month. But this is not what we are after. We want to run the job in the beginning of the month.

   Furthermore, the job did not start because the SYSDATE is already passed at the time of enabling it.

```
SELECT ENABLED, STATE, RUN_COUNT, NEXT_RUN_DATE FROM USER_SCHEDULER_JOBS WHERE
JOB_NAME='POPULATE_SALES_RPT';
```

10. Run the following code to fix the frequency interval of the job.

```
BEGIN
 DBMS_SCHEDULER.SET_ATTRIBUTE (
  NAME => 'POPULATE_SALES_RPT',
  ATTRIBUTE => 'REPEAT_INTERVAL',
  VALUE => 'FREQ=MONTHLY;INTERVAL=1;BYMONTHDAY=1;BYHOUR=0;BYMINUTE=0;BYSECOND=10');
END;
/
```

11. Submit the following query.

    Observe that the job is scheduled to run at midnight in the first day of the next month.

```
SELECT ENABLED, STATE, RUN_COUNT, NEXT_RUN_DATE FROM USER_SCHEDULER_JOBS WHERE
JOB_NAME='POPULATE_SALES_RPT';
```

We do not want to wait for the next month. Let's manually start the Scheduler job.

12. Manually start the job.

```
exec DBMS_SCHEDULER.RUN_JOB('POPULATE_SALES_RPT')
```

13. Verify that the report table is populated.

    You should see the table is populated for the summation and total orders for the previous month.

```
SELECT * FROM SALES_RPT;
```

14. Drop the job.

```
exec DBMS_SCHEDULER.DROP_JOB('POPULATE_SALES_RPT')
```

B. Creating a Scheduler Job using a Schedule and a Program

In this section of the practice, you will create a Scheduler job that performs the same task as the task of the job created in the previous section. The difference is that this time you will use the Schedule and Program objects for creating the job.

15. Create a Scheduler program that runs the procedure

    Observe that the ENABLED parameter is set so that the program becomes enabled.

```
BEGIN
DBMS_SCHEDULER.CREATE_PROGRAM(
 PROGRAM_NAME => 'POPULATE_SALES_PROG',
 PROGRAM_ACTION => 'MYPKG.POPULATE_SALES_RPT;',
 PROGRAM_TYPE => 'PLSQL_BLOCK',
 ENABLED => TRUE);
END;
/
```

16. Create a Schedule for the beginning of each month

```
BEGIN
 DBMS_SCHEDULER.CREATE_SCHEDULE (
  SCHEDULE_NAME => 'MONTH_BEGIN_SCH',
  START_DATE => SYSTIMESTAMP,
  END_DATE => NULL,
  REPEAT_INTERVAL =>
'FREQ=MONTHLY;INTERVAL=1;BYMONTHDAY=1;BYHOUR=0;BYMINUTE=0;BYSECOND=10',
  COMMENTS => 'A schedule that represents the first day of every month at midnight');
END;
/
```

17. Create a job that uses the created program and schedule.

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB(
 JOB_NAME => 'POPULATE_SALES_JOB',
 PROGRAM_NAME => 'POPULATE_SALES_PROG',
 SCHEDULE_NAME => 'MONTH_BEGIN_SCH',
 ENABLED => TRUE);
END;
/
```

18. Retrieve information about the job.

```
col STATE for a13
col NEXT_RUN_DATE for a35
SELECT ENABLED, STATE, RUN_COUNT, NEXT_RUN_DATE
FROM USER_SCHEDULER_JOBS WHERE JOB_NAME='POPULATE_SALES_JOB';
```

19. Run the job manually to test it.

```
exec DBMS_SCHEDULER.RUN_JOB('POPULATE_SALES_JOB')
```

20. Verify that the report table is populated.

```
SELECT * FROM SALES_RPT;
```

Note: Do not drop the job now. You will still use it in the next practice section.

C. Monitor Running jobs and Scheduler Job Logs

In this section of the practice, you will monitor Scheduler running job and retrieve job logging information.

21. Open SQL Developer and connect to PDB1 as **SYSTEM**. In the rest of this practice, this session will be referred to as the **monitoring** session.

22. In the SQL*Plus session, modify the procedure as follows. The change makes the procedure takes nearly 3 minutes to finish.

```
CREATE OR REPLACE PACKAGE BODY MYPKG AS
PROCEDURE POPULATE_SALES_RPT
AS
 N DATE;
 X NUMBER;
BEGIN
 N := SYSDATE;
 -- loop for 3 minutes
 WHILE ((SYSDATE- N)*24*60) < 3 LOOP
  -- dummy computation
  X := SQRT(DBMS_RANDOM.VALUE(1,1000));
 END LOOP;
INSERT INTO SALES_RPT
 SELECT NULL, SYSDATE,
  EXTRACT ( MONTH FROM SYSDATE-1 ),
  EXTRACT ( YEAR FROM SYSDATE-1 ),
  SUM(ORDER_TOTAL), COUNT(ORDER_ID)
 FROM ORDERS
 WHERE ORDER_STATUS < 5
   AND TO_CHAR(ORDER_DATE,'MM-YYYY') = TO_CHAR(SYSDATE-1,'MM-YYYY');
 COMMIT;
END;
END;
/
```

23. In SQL*Plus session, Retrieve the logging level for the job.

The logging level for the job class ('RUNS') is higher than the logging level of the job ('OFF'). In this case, the effective logging level for the job will be 'RUNS'.

```
SELECT JOB_CLASS, LOGGING_LEVEL JOB_LOG_LEVEL,
(SELECT C.LOGGING_LEVEL FROM ALL_SCHEDULER_JOB_CLASSES C WHERE
C.JOB_CLASS_NAME=J.JOB_CLASS) CLASS_LOG_LEVEL
FROM USER_SCHEDULER_JOBS J
WHERE JOB_NAME='POPULATE_SALES_JOB';
```

24. Run the job.

Tip: set USE_CURRENT_SESSION to FALSE so that the job progress is tracked by the Scheduler performance views.

If the USE_CURRENT_SESSION is kept to its default value (TRUE), the command line does not return to the prompt until the job is finished.

```
exec DBMS_SCHEDULER.RUN_JOB(JOB_NAME => 'POPULATE_SALES_JOB', USE_CURRENT_SESSION=>FALSE)
```

25. In the **monitoring** session, submit the following query to retrieve information about the running job.

    SESSION_ID is used to link the job process with V$SESSION.

    SLAVE_OS_PROCESS_ID is used to link the job process with the OS processes (can be retrieved from the OS command ps) and V$PROCESS.

```
SELECT SESSION_ID, SLAVE_OS_PROCESS_ID, ELAPSED_TIME, CPU_USED
FROM DBA_SCHEDULER_RUNNING_JOBS WHERE JOB_NAME='POPULATE_SALES_JOB';
```

26. Retrieve more information about the process that runs the job from the V$SESSION.

    In Oracle database, V$SESSION is the most important source of information about current sessions. DBAs always refer to it to obtain details about any current client sessions, including user application sessions, background running processes, and processes that run Scheduler Jobs.

    Observe that the sessions that represent running Scheduler jobs have their PROGRAM column values with the format (J*n*), where n is an integer number, and their MODULE column values with the value 'DBMS_SCHEDULER'. The ACTION column is set to the job name.

```
SELECT S.USERNAME, S.STATUS, S.PROCESS, S.PROGRAM, S.CON_ID, SQL_ID, MODULE, ACTION,
S.STATE, EVENT#, EVENT, SECONDS_IN_WAIT
FROM DBA_SCHEDULER_RUNNING_JOBS J, V$SESSION S
WHERE J.SESSION_ID = S.SID
AND J.SESSION_ID = (SELECT SESSION_ID FROM DBA_SCHEDULER_RUNNING_JOBS WHERE
                     JOB_NAME='POPULATE_SALES_JOB') ;
```

27. Retrieve the log information for the Scheduler job.

```
SELECT LOG_ID, LOG_DATE, OPERATION, STATUS, DESTINATION
FROM DBA_SCHEDULER_JOB_LOG
WHERE JOB_NAME='POPULATE_SALES_JOB' ORDER BY LOG_DATE DESC;
```

28. Retrieve the details log information for the Scheduler job.

```
SELECT LOG_DATE, JOB_NAME, STATUS, ADDITIONAL_INFO
FROM DBA_SCHEDULER_JOB_RUN_DETAILS D
WHERE D.LOG_ID = (SELECT MAX(P.LOG_ID) FROM DBA_SCHEDULER_JOB_LOG P
                   WHERE JOB_NAME='POPULATE_SALES_JOB')
ORDER BY LOG_DATE;
```

Cleanup

29. Remove the created Scheduler job, schedule, and program.

    The procedure fails if the job is still running.

```
exec DBMS_SCHEDULER.DROP_JOB(JOB_NAME => 'POPULATE_SALES_JOB')
exec DBMS_SCHEDULER.DROP_PROGRAM( PROGRAM_NAME => 'POPULATE_SALES_PROG')
exec DBMS_SCHEDULER.DROP_SCHEDULE ( SCHEDULE_NAME => 'MONTH_BEGIN_SCH')
```

30. Drop the report table.

```
DROP TABLE SOE.SALES_RPT PURGE;
```

31. Exit from SQL Developer.

Summary

- The Scheduler job can be created with a single command (`DBMS_SCHEDULER.CREATE_JOB`) where we can provide all the job's attributes. Alternatively, we can create Scheduler Programs and Schedules and then use them to create the Scheduler jobs.

- The Scheduler performance views can be used to obtain information about the running jobs and the job history. They can be linked with other views to obtain more details on the Scheduler Jobs.