

DBMS_JOB RIP – Introducing The Oracle 10g Scheduler

Howard Ong

Principal Consultant

Aurora Consulting Pty Ltd

Abstract

Like trusted old friends, many Oracle database tools have served developers and DBAs well for many years. Alas, as surely as flat-screen TVs are replacing the old CRT TVs in our living rooms, many tried-and-tested database tools are being replaced by newer and greater technology. This paper describes the new Oracle 10g Scheduler and explains why it is a more superior tool to the good-old DBMS_JOB. While there are some changes in the syntax, the author argues that the new features will justify the upgrade effort. Besides, you just never know how long more will Oracle let DBMS_JOB hangs around ...

About The Author

A frequent presenter at Oracle Conferences, Howard has been an expert user of Oracle database and tools for the past 14 years and has been working on database technology since 1991. In particular, Howard possesses in-depth experience in the planning and development of Business Intelligence and E-Business Applications. Harboring a keen interest in the application of Oracle Technology in these areas, Howard and his team have helped many organisations deploy BI and E-Business solutions using various Oracle and third-party technology. For the past 10 years, Aurora Consulting has been delivering Data Warehouses and E-Business solutions to a wide variety of organisations including large government departments and ASX-listed companies.

For more information on Howard or Aurora Consulting, visit <http://www.aurora-consult.com.au> or email info@aurora-consult.com.au.

Introduction

Introduced in Oracle 10g, DBMS_SCHEDULER is a vastly superior replacement to DBMS_JOBS. This paper explains the basic concept and commonly used features of DBMS_SCHEDULER.

DBMS_JOB Recap

DBMS_JOB is a pre-build PLSQL package in Oracle Database, providing job scheduling capabilities. The key procedures in DBMS_JOB are Submit and Change, allowing the users to submit and modify scheduled jobs.

Their syntax are:

```
DBMS_JOB.SUBMIT (
  job OUT BINARY_INTEGER,
  what IN VARCHAR2,
  next_date IN DATE DEFAULT sysdate,
  interval IN VARCHAR2 DEFAULT 'null',
  no_parse IN BOOLEAN DEFAULT FALSE,
  instance IN BINARY_INTEGER DEFAULT any_instance,
  force IN BOOLEAN DEFAULT FALSE);
```

```
DBMS_JOB.CHANGE (
  job IN BINARY_INTEGER,
  what IN VARCHAR2,
  next_date IN DATE,
  interval IN VARCHAR2,
  instance IN BINARY_INTEGER DEFAULT NULL,
  force IN BOOLEAN DEFAULT FALSE);
```

where

- job – job number
- what – PLSQL procedure to run
- next_date – the next run date/time for the job
- interval – a PLSQL function that accepts the current run date/time and return the next run date/time
- no_parse – for deferring parsing of the PLSQL procedure until the first time the job is run
- instance –(RAC environment)for designating which instance can run the job
- force – (RAC environment)for enforcing the rule that an instance must be running before a job can be designated to it

To view the status of jobs, the data dictionary table DBA_JOBS, USER_JOBS or ALL_JOBS are queried. The commonly used columns in these tables are :

- Job – job number
- Log_User – User ID who submitted the job
- Priv_User –User ID whose privileges are used to execute the job
- Schema User – Default schema in which the job runs
- Last_Date – The last time this job is successfully executed
- This_Date – The time this job started executing (for running job)
- Next_Date – The next time this job is scheduled to execute.
- What – PLSQL procedure to run

For example

```
SQL> select job,log_user,priv_user,schema_user
 2   ,to_char(last_date,'YYYYMMDD HH24MI') last_date
 3   ,to_char(next_date,'YYYYMMDD HH24MI') next_date
```

```

4  from dba_jobs
5  /

```

JOB LOG_USER		PRIV_USER	
SCHEMA_USER	LAST_DATE	NEXT_DATE	
2 META		META	
META	20071109 1759	20071109 1814	

DBMS_JOB Limitations

In spite of its tireless service to developers and DBAs for many years, DBMS_JOB does have its limitations. Some of these are:

- Depending on the server resources, the actual run time is almost always some time after the exact time it is scheduled to run. The next scheduled run time is then computed based on the actual run time of the current run. Hence, additional logic in the interval function is required if you want the job to run at the fixed time each time, e.g. 5 am every day.
- Complex logic in the interval function is required to implement complex scheduling. For example, consider how you would schedule a job to run at 5 am on every Monday, Wednesday and Friday, excluding public holidays.
- PLSQL block does not accept parameters.
- Does not support the display of the full schedule (i.e. a series of future execution date/time) for verification purposes.
- Does not support allocating different level of resources and priorities to different jobs.
- Does not support event-driven jobs.

A Simple DBMS_SCHEDULER Job

DBMS_SCHEDULER is a vastly improved scheduler to DBMS_JOB, and overcome many of DBMS_JOB's limitations. In its simplest form, you can use the Create_Job procedure to schedule a job, in a manner similar to DBMS_JOB.Submit:

```

BEGIN
  /* A simple schedule job */

  DBMS_SCHEDULER.CREATE_JOB
    (job_name => 'JOB_SCHEMA_STATS_HR'
    ,job_type => 'plsql_block'
    ,job_action =>
      'begin
        dbms_stats.gather_schema_stats(''HR'');
      end;'
    ,start_date => to_timestamp_tz('01/12/2007 05:00:00', 'dd/mm/yyyy hh24:mi:ss')
    ,repeat_interval => 'freq=MONTHLY'
    ,enabled => true
    ,comments => 'Generate HR schema stats every month'
    );
END;
/

```

Used in the above form, other than a more intuitive to specify repeat interval, there are not too many advantages over DBMS_JOB.

Programs, Schedules & Jobs

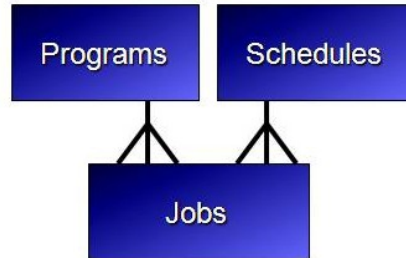
To fully unleash the power of DBMS_SCHEDULER, one needs to understand 3 new objects:

Program. A Program is what the scheduler executes, usually a PLSQL block or a stored procedure.

Schedule. A Schedule defines when and how often DBMS_SCHEDULER performs the execution.

Job. Finally, a Job is a scheduled program.

The diagram below defines the interrelationship among Programs, Schedules and Jobs.



Example:

```

BEGIN
  /* Using Schedule, Program and Job */

  DBMS_SCHEDULER.CREATE_SCHEDULE
  (schedule_name => 'SCHD_FIRST_DAY_EACH_MTH '
  ,start_date => to_timestamp_tz('01/12/2007 05:00:00', 'dd/mm/yyyy hh24:mi:ss')
  ,repeat_interval => 'freq=MONTHLY'
  ,comments => 'Run first day each month'
  );

  DBMS_SCHEDULER.CREATE_PROGRAM
  (program_name => 'PROG_SCHEMA_STATS_HR'
  ,program_type => 'plsqli_block'
  ,program_action =>
  'begin
  dbms_stats.gather_schema_stats(''HR'');
  end;'
  ,enabled => true
  ,comments => 'Generate HR schema stats'
  );

  DBMS_SCHEDULER.DROP_JOB
  ('JOB_SCHEMA_STATS_HR'
  ,true
  );

  DBMS_SCHEDULER.CREATE_JOB
  (job_name => 'JOB_SCHEMA_STATS_HR'
  ,program_name => 'PROG_SCHEMA_STATS_HR'
  ,schedule_name => 'SCHD_FIRST_DAY_EACH_MTH'
  ,enabled => true
  ,comments => 'Generate HR schema stats every month'
  );
END;
/

```

The main advantage of this setup is object reusability. For example, you can set up a complex schedule and use it in multiple jobs by associating it with many different programs. To change the timing of all these jobs, you only have to change that one schedule.

Program reusability is further enhanced by the ability to create parameterised programs:

```

CREATE OR REPLACE PROCEDURE schema_stats
  (p_schema in varchar2)
IS
BEGIN
  DBMS_STATS.GATHER_SCHEMA_STATS (p_schema);
END;
/

BEGIN
  /* Using program arguments */

  DBMS_SCHEDULER.CREATE_PROGRAM
  (program_name => 'PROG_SCHEMA_STATS'
  ,program_type => 'stored_procedure'

```

```

,program_action => 'schema_stats'
,number_of_arguments => 1
,enabled => false
,comments => 'Generate schema stats'
);
DBMS_SCHEDULER.DEFINE_PROGRAM_ARGUMENT
(program_name => 'PROG_SCHEMA_STATS'
,argument_position => 1
,argument_name => 'schema'
,argument_type => 'VARCHAR2'
);
DBMS_SCHEDULER.ENABLE ('PROG_SCHEMA_STATS');

DBMS_SCHEDULER.DROP_JOB
('JOB_SCHEMA_STATS_HR'
,true
);

DBMS_SCHEDULER.CREATE_JOB
(job_name => 'JOB_SCHEMA_STATS_HR'
,program_name => 'PROG_SCHEMA_STATS'
,schedule_name => 'SCHD_FIRST_DAY_EACH_MTH'
,enabled => false
,comments => 'Generate HR schema stats every month'
);
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE
(job_name => 'JOB_SCHEMA_STATS_HR'
,argument_position => 1
,argument_value => 'HR'
);
DBMS_SCHEDULER.ENABLE ('JOB_SCHEMA_STATS_HR');

DBMS_SCHEDULER.CREATE_JOB
(job_name => 'JOB_SCHEMA_STATS_SCOTT'
,program_name => 'PROG_SCHEMA_STATS'
,schedule_name => 'SCHD_FIRST_DAY_EACH_MTH'
,enabled => false
,comments => 'Generate SCOTT schema stats every month'
);
DBMS_SCHEDULER.SET_JOB_ARGUMENT_VALUE
(job_name => 'JOB_SCHEMA_STATS_SCOTT'
,argument_position => 1
,argument_value => 'SCOTT'
);
DBMS_SCHEDULER.ENABLE ('JOB_SCHEMA_STATS_SCOTT');
END;
/

```

In the example above, rather than setting up 2 separate programs for generating the schema statistics for HR and SCOTT, a single program with parameter support is suffice.

Complex Schedules

One of the most powerful features of DBMS_SCHEDULER is its comprehensive repeat interval settings. Some useful repeat_interval attributes and values are:

Attributes	Values
freq	YEARLY, MONTHLY, WEEKLY, DAILY, HOURLY, MINUTELY, SECONDLY.
interval	How many periods before the schedule executes? 1 to 999. Default 1.
bymonth	JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC
bymonthday	1 to 31. Use minus (-) sign to denote counting back from the last day of the month.
bydate	Date in YYYYMMDD format.
byday	MON, TUE, WED, THU, FRI, SAT, SUN. For freq=YEARLY or MONTHLY, a number may be added to the front to denote the nth occurrence of that day in that period. Use minus (-) sign to denote counting back from the end of the period.
bysetpos	Used in conjunction with another “by” attribute, specifying the nth occurrence of the “by”

Attributes	Values
	interval. Use minus (-) sign to denote counting back from the last interval.
exclude	Specific the name of one or more excluded schedule. All date/time indicated by the excluded schedules would be not activated.

Note:

- Always start with the “freq” clause.
- Separate clauses by semi-colons (;).
- Insert white spaces between syntax elements to improve readability.
- Syntax elements are case-insensitive.

The following example creates a schedule consisting of all 2008 public holidays:

```
BEGIN
  /* Complex scheduling example 1: Public Holidays */

  DBMS_SCHEDULER.CREATE_SCHEDULE
    (schedule_name => 'SCHD_2008_PUBLIC_HOLIDAYS'
    ,start_date => to_timestamp_tz('01/01/2008 00:00:00', 'dd/mm/yyyy hh24:mi:ss')
    ,end_date => to_timestamp_tz('31/12/2008 23:59:59', 'dd/mm/yyyy hh24:mi:ss')
    ,repeat_interval =>
      'freq=DAILY;
      bydate=20080101,20080128,20080303,20080321,20080324,20080425,
      20080929,20081225,20081226'
    ,comments => 'Public Holidays 2008'
    );
END;
/
```

The following example creates a schedule consisting of all 2008 working days:

```
BEGIN
  /* Complex scheduling example 2: Working Days */

  DBMS_SCHEDULER.CREATE_SCHEDULE
    (schedule_name => 'SCHD_2008_WORKING_DAYS'
    ,start_date => to_timestamp_tz('01/01/2008 00:00:00', 'dd/mm/yyyy hh24:mi:ss')
    ,end_date => to_timestamp_tz('31/12/2008 23:59:59', 'dd/mm/yyyy hh24:mi:ss')
    ,repeat_interval =>
      'freq=DAILY;
      byday=MON,TUE,WED,THU,FRI;
      exclude=SCHD_2008_PUBLIC_HOLIDAYS'
    ,comments => 'Run on every working day'
    );
END;
/
```

The following example creates a schedule consisting of the first working day of every month in 2008:

```
BEGIN
  /* Complex scheduling example 3: 1st Working Day in a Month */

  DBMS_SCHEDULER.CREATE_SCHEDULE
    (schedule_name => 'SCHD_2008_1ST_WK_DAY_EACH_MTH'
    ,start_date => to_timestamp_tz('01/01/2008 00:00:00', 'dd/mm/yyyy hh24:mi:ss')
    ,end_date => to_timestamp_tz('31/12/2008 23:59:59', 'dd/mm/yyyy hh24:mi:ss')
    ,repeat_interval =>
      'freq=MONTHLY;
      byday=MON,TUE,WED,THU,FRI;
      exclude=SCHD_2008_PUBLIC_HOLIDAYS;
      bysetpos=1'
    ,comments => 'Run on 1st working day every month'
    );
END;
/
```

The following example creates a schedule consisting of the first and third Monday of each month:

```
BEGIN
  /* Complex scheduling example 4: 1st and 3rd Monday each month */

  DBMS_SCHEDULER.CREATE_SCHEDULE
    (schedule_name => 'SCHD_1ST_3RD_MON_EACH_MTH'
    ,start_date => to_timestamp_tz('03/12/2007 05:00:00', 'dd/mm/yyyy hh24:mi:ss')
    ,repeat_interval =>
      'freq=MONTHLY;
      byday=1MON,3MON'
    ,comments => 'Run on every 1st and 3rd Monday each month'
    );
END;
/
```

The following example creates a schedule consisting of the last day of each quarter:

```
BEGIN
  /* Complex scheduling example 5: Quarter */

  DBMS_SCHEDULER.CREATE_SCHEDULE
    (schedule_name => 'SCHD_LAST_DAY_EACH_QTR'
    ,start_date => to_timestamp_tz('31/12/2008 05:00:00', 'dd/mm/yyyy hh24:mi:ss')
    ,repeat_interval =>
      'freq=YEARLY;
      bymonth=MAR,JUN,SEP,DEC;
      bymonthday=-1'
    ,comments => 'Run on every last day each quarter'
    );
END;
/
```

Once a schedule is setup, how do you know if it has been defined correctly? In DBMS_JOBS, you may only view the next scheduled time, make it difficult to properly evaluate whether schedule has been set up properly. DBMS_SCHEDULER introduces the Evaluate_Calendar_String procedure to overcome this problem. Example:

```
CREATE OR REPLACE PROCEDURE show_schedule
  (schedule_name in varchar2
  ,start_date in timestamp
  ,repeat_interval in varchar2
  ,iteration in number)
AS
  v_start_date timestamp;
  v_after_date timestamp;
  v_next_date timestamp;

BEGIN
  dbms_output.put_line (schedule_name);
  v_start_date := start_date;
  v_after_date := v_start_date - (1/86400);

  for i in 1..iteration loop
    dbms_scheduler.evaluate_calendar_string
      (repeat_interval
      ,v_start_date
      ,v_after_date
      ,v_next_date
      );

    dbms_output.put_line (v_next_date);
    v_after_date := v_next_date;
  end loop;
  dbms_output.put_line ('=====');
END;
/

set serveroutput on
exec dbms_output.enable (1000000);

begin
  show_schedule
    (schedule_name => 'SCHD_2008_PUBLIC_HOLIDAYS'
    ,start_date => to_timestamp_tz('01/01/2008 00:00:00', 'dd/mm/yyyy hh24:mi:ss')
    ,repeat_interval =>
      'freq=DAILY;
      bydate=20080101,20080128,20080303,20080321,20080324,20080425,
      20080929,20081225,20081226'
    ,iteration => 9);
```

```

show_schedule
(schedule_name => 'SCHD_2008_WORKING_DAYS'
,start_date => to_timestamp_tz('01/01/2008 00:00:00', 'dd/mm/yyyy hh24:mi:ss')
,repeat_interval =>
'freq=DAILY;
byday=MON,TUE,WED,THU,FRI;
exclude=SCHD_2008_PUBLIC_HOLIDAYS'
,iteration => 21);

show_schedule
(schedule_name => 'SCHD_2008_1ST_WK_DAY_EACH_MTH'
,start_date => to_timestamp_tz('01/01/2008 00:00:00', 'dd/mm/yyyy hh24:mi:ss')
,repeat_interval =>
'freq=MONTHLY;
byday=MON,TUE,WED,THU,FRI;
exclude=SCHD_2008_PUBLIC_HOLIDAYS;
bysetpos=1'
,iteration => 12);

show_schedule
(schedule_name => 'SCHD_1ST_3RD_MON_EACH_MTH'
,start_date => to_timestamp_tz('03/12/2007 05:00:00', 'dd/mm/yyyy hh24:mi:ss')
,repeat_interval =>
'freq=MONTHLY;
byday=1MON,3MON'
,iteration => 26);

show_schedule
(schedule_name => 'SCHD_LAST_DAY_EACH_QTR'
,start_date => to_timestamp_tz('31/12/2008 05:00:00', 'dd/mm/yyyy hh24:mi:ss')
,repeat_interval =>
'freq=YEARLY;
bymonth=MAR,JUN,SEP,DEC;
bymonthday=-1'
,iteration => 5);

end;
/

set serverout off

```

Data Dictionary Tables

Whereas DBMS_JOBS effectively only provides one data dictionary table (DBA_JOBS) for administration purposes, DBMS_SCHEDULER provides a large number of data dictionary tables for the administration of various aspects of the tool.

DBA_SCHEDULER data dictionary tables:

Table Name	Description
DBA_OBJECTS	All objects in the database
DBA_SCHEDULER_PROGRAMS	All scheduler programs in the database
DBA_SCHEDULER_JOBS	All scheduler jobs in the database
DBA_SCHEDULER_JOB_CLASSES	All scheduler classes in the database
DBA_SCHEDULER_WINDOWS	All scheduler windows in the database
DBA_SCHEDULER_PROGRAM_ARGS	All arguments of all scheduler programs in the database
DBA_SCHEDULER_JOB_ARGS	All arguments with set values of all scheduler jobs in the database
DBA_SCHEDULER_JOB_LOG	Logged information for all scheduler jobs
DBA_SCHEDULER_JOB_RUN_DETAILS	The details of a job run
DBA_SCHEDULER_WINDOW_LOG	Logged information for all scheduler windows
DBA_SCHEDULER_WINDOW_DETAILS	The details of a window
DBA_SCHEDULER_WINDOW_GROUPS	All scheduler window groups in the database

Table Name	Description
DBA_SCHEDULER_WINDOW_MEMBERS	Members of all scheduler window groups in the database
DBA_SCHEDULER_SCHEDULES	All schedules in the database
DBA_SCHEDULER_GLOBAL_ATTRIBUTE	All scheduler global attributes
DBA_SCHEDULER_CHAINS	All scheduler chains in the database
DBA_SCHEDULER_CHAIN_RULES	All rules from scheduler chains in the database
DBA_SCHEDULER_CHAIN_STEPS	All steps of scheduler chains in the database
DBA_SCHEDULER_RUNNING_CHAINS	All steps of all running chains in the database

Corresponding ALL_ objects exist for all the above tables. However, only some of the above tables have corresponding USER_ objects.

Example 1

```
col object_type      format a15
col object_name      format a30
col status           format a10

select
  object_type
,object_name
,status
from user_objects
where object_type in ('PROGRAM', 'JOB', 'SCHEDULE')
order by object_type, object_name
/
```

Output:

OBJECT_TYPE	OBJECT_NAME	STATUS
JOB	JOB_SCHEMA_STATS_HR	VALID
JOB	JOB_SCHEMA_STATS_SCOTT	VALID
PROGRAM	PROG_SCHEMA_STATS	VALID
PROGRAM	PROG_SCHEMA_STATS_HR	VALID
SCHEDULE	SCHD_1ST_3RD_MON_EACH_MTH	VALID
SCHEDULE	SCHD_2008_1ST_WK_DAY_EACH_MTH	VALID
SCHEDULE	SCHD_2008_PUBLIC_HOLIDAYS	VALID
SCHEDULE	SCHD_2008_WORKING_DAYS	VALID
SCHEDULE	SCHD_FIRST_DAY_EACH_MTH	VALID
SCHEDULE	SCHD_LAST_DAY_EACH_QTR	VALID

Example 2

```
set linesize 100
col schedule_name    format a30
col start_date       format a20
col end_date         format a20
col repeat_interval  format a15

select
  schedule_name
, to_char(start_date, 'dd/mm/yyyy hh24:mi:ss') start_date
, to_char(end_date, 'dd/mm/yyyy hh24:mi:ss') end_date
, substr(repeat_interval,1,10) || '...' repeat_interval
from user_scheduler_schedules
order by schedule_name
/
```

Output:

SCHEDULE_NAME	START_DATE	END_DATE	REPEAT_INTERVAL
SCHD_1ST_3RD_MON_EACH_MTH	03/12/2007 05:00:00		freq=MONTH...
SCHD_2008_1ST_WK_DAY_EACH_MTH	01/01/2008 00:00:00	31/12/2008 23:59:59	freq=MONTH...
SCHD_2008_PUBLIC_HOLIDAYS	01/01/2008 00:00:00	31/12/2008 23:59:59	freq=DAILY...
SCHD_2008_WORKING_DAYS	01/01/2008 00:00:00	31/12/2008 23:59:59	freq=DAILY...
SCHD_FIRST_DAY_EACH_MTH	01/12/2007 05:00:00		freq=MONTH...
SCHD_LAST_DAY_EACH_QTR	31/12/2008 05:00:00		freq=YEARL...

Example 3

```

set linesize 100
col program_name      format a20
col program_type      format a20
col program_action    format a50

select
  program_name
, program_type
, program_action
from user_scheduler_programs
order by program_name
/

```

Output:

```

PROGRAM_NAME          PROGRAM_TYPE          PROGRAM_ACTION
-----
PROG_SCHEMA_STATS    STORED_PROCEDURE     schema_stats
PROG_SCHEMA_STATS_HR PLSQL_BLOCK          begin
                                                                dbms_stats.gather_schema_stats('HR');
                                                                end;

```

Example 4

```

col job_name          format a25
col state             format a10
col program_name      format a20
col job_class         format a25
col job_type          format a10
col job_action        format a20

select
  job_name
, state
, program_name
, job_type
, job_action
from user_scheduler_jobs
order by job_name
/

```

Output:

```

JOB_NAME              STATE                PROGRAM_NAME          JOB_TYPE  JOB_ACTION
-----
JOB_SCHEMA_STATS_HR   SCHEDULED           PROG_SCHEMA_STATS
JOB_SCHEMA_STATS_SCOTT SCHEDULED           PROG_SCHEMA_STATS

```

System Privileges

Common DBA_SCHEDULER privileges:

Privileges	Description
CREATE JOB	Ability to create/alter/drop jobs, schedules and programs
CREATE ANY JOB	Ability to create/alter/drop jobs, schedules and programs belonging to any user
EXECUTE ANY PROGRAM	Use programs belonging to any user
EXECUTE ANY CLASS	Ability to create/alter/drop job classes
MANAGE SCHEDULER	Ability to create/alter/drop job classes, windows and window groups belonging to any user

Advanced Features

Beyond defining jobs, schedules and programs, there exists advanced features that allow you to control how much resources a job gets:

- Job Classes, Windows, Window Groups
- DBMS_RESOURCE_MANAGER

In addition, there are also facilities to create event-driven and inter-dependent jobs. Thus, you are able to set up a series of jobs that are kicked off upon the occurrence of an event, e.g. upon the appearance of a file in a folder.

- DBMS_SCHEDULER.CREATE_CHAIN
- DBMS_SCHEDULER.DEFINE_CHAIN_EVENT_STEP
- DBMS_SCHEDULER.DEFINE_CHAIN_STEP
- DBMS_SCHEDULER.DEFINE_CHAIN_RULE
- DBMS_SCHEDULER.RUN_CHAIN
- DBMS_SCHEDULER.CREATE_EVENT_SCHEDULE
- Create event queue, subscriber and agents (DBMS_AQADM)

Conclusion

Owing to the differences in syntax, conversion from DBMS_JOB to DBMS_SCHEDULER takes some effort. However, given DBMS_SCHEDULER's significantly superior functionality, and Oracle's continual enhancement of its features, the conversion effort may very well be worth the while.