**Scheduler (DBMS\_SCHEDULER) in Oracle Database 10g**

**Permissions……….**

GRANT CREATE JOB TO SCOTT

begin

DBMS\_SCHEDULER.CREATE\_JOB('SCOTT.XML\_FILE\_GEN\_JOB2','STORED\_PROCEDURE',

job\_action => 'SCOTT.proc\_xml\_fgen',

start\_date => to\_date('10-11-2013 00:00:00', 'dd-mm-yyyy hh24:mi:ss'),

repeat\_interval => 'Freq=Minutely;Interval=5;ByDay=SUN;ByHour=00;ByMinute=0;BySecond=0',

enabled => FALSE);

DBMS\_SCHEDULER.ENABLE('SCOTT.XML\_FILE\_GEN\_JOB2');

DBMS\_SCHEDULER.RUN\_JOB('SCOTT.XML\_FILE\_GEN\_JOB2');

end;

/

--Working below

begin

sys.dbms\_scheduler.create\_job(job\_name => 'SCOTT.JOB\_XML\_GEN',

job\_type => 'STORED\_PROCEDURE',

job\_action => 'proc\_xml\_fgen',

start\_date => to\_date('01-12-2013 00:00:00', 'dd-mm-yyyy hh24:mi:ss'),

repeat\_interval => 'Freq=Hourly;Interval=10;ByHour=13;ByMinute=05',

end\_date => to\_date(null),

job\_class => 'DEFAULT\_JOB\_CLASS',

enabled => true,

auto\_drop => false,

comments => '');

end;

/

Oracle 10g includes a comprehensive scheduler (DBMS\_SCHEDULER) to replace and extend the functionality provided by the DBMS\_JOB package. Jobs form the core of the functionality, but there are several other components available.

* [Simple Example](http://www.oracle-base.com/articles/10g/scheduler-10g.php#simple)
* [Programs](http://www.oracle-base.com/articles/10g/scheduler-10g.php#programs)
* [Schedules](http://www.oracle-base.com/articles/10g/scheduler-10g.php#schedules)
* [Jobs](http://www.oracle-base.com/articles/10g/scheduler-10g.php#jobs)
* [Job Classes](http://www.oracle-base.com/articles/10g/scheduler-10g.php#job_classes)
* [Windows](http://www.oracle-base.com/articles/10g/scheduler-10g.php#windows)
* [Windows Groups](http://www.oracle-base.com/articles/10g/scheduler-10g.php#window_groups)
* [Enable, Disable and Attributes](http://www.oracle-base.com/articles/10g/scheduler-10g.php#enable_disable_and_attributes)
* [Configuring The Scheduler](http://www.oracle-base.com/articles/10g/scheduler-10g.php#configuring_the_scheduler)
* [Calendar Syntax Examples](http://www.oracle-base.com/articles/10g/scheduler-10g.php#calendar_syntax_examples)
* [Extracting DDL](http://www.oracle-base.com/articles/10g/scheduler-10g.php#ddl)
* [Scheduler Views](http://www.oracle-base.com/articles/10g/scheduler-10g.php#views)

Related articles.

* [Scheduler Enhancements in Oracle Database 10g Release 2](http://www.oracle-base.com/articles/10g/scheduler-enhancements-10gr2.php)
* [Scheduler Enhancements in Oracle Database 11g Release 1](http://www.oracle-base.com/articles/11g/scheduler-enhancements-11gr1.php)
* [Scheduler Enhancements in Oracle Database 11g Release 2](http://www.oracle-base.com/articles/11g/scheduler-enhancements-11gr2.php)
* [Remote Scheduler Agent Installation for Oracle 11g Release 2](http://www.oracle-base.com/articles/11g/scheduler-agent-installation-11gr2.php)
* [Services in Oracle Database 10g](http://www.oracle-base.com/articles/10g/database-services-10g.php)
* [SQL Developer 3.1 Scheduler (DBMS\_SCHEDULER) Support](http://www.oracle-base.com/articles/misc/sql-developer-31-scheduler-support.php)

**Simple Example**

Although the scheduler is capable of very complicated schedules, on many occasions you just want to create a simple job with everything defined inline. If that's all you want, the following example is for you.

**BEGIN**

**DBMS\_SCHEDULER.create\_job (**

**job\_name => 'test\_full\_job\_definition',**

**job\_type => 'PLSQL\_BLOCK',**

**job\_action => 'BEGIN my\_job\_procedure; END;',**

**Start\_date => SYSTIMESTAMP,**

**repeat\_interval => 'freq=hourly; byminute=0; bysecond=0;',**

**end\_date => NULL,**

**Enabled => TRUE,**

**Comments => 'Job defined entirely by the CREATE JOB procedure.');**

**END;**

**/**

**Programs**

The scheduler allows you to optionally create programs which hold metadata about a task, but no schedule information. A program may related to a PL/SQL block, a stored procedure or an OS executable file. Programs are created using the CREATE\_PROGRAM procedure.

**-- Create the test programs.**

**BEGIN**

**-- PL/SQL Block.**

**DBMS\_SCHEDULER.create\_program (**

**program\_name => 'test\_plsql\_block\_prog',**

**program\_type => 'PLSQL\_BLOCK',**

**program\_action => 'BEGIN DBMS\_STATS.gather\_schema\_stats(''SCOTT''); END;',**

**enabled => TRUE,**

**comments => 'Program to gather SCOTT''s statistics using a PL/SQL block.');**

**-- Shell Script.**

**DBMS\_SCHEDULER.create\_program (**

**program\_name => 'test\_executable\_prog',**

**program\_type => 'EXECUTABLE',**

**program\_action => '/u01/app/oracle/dba/gather\_scott\_stats.sh',**

**number\_of\_arguments => 0,**

**enabled => TRUE,**

**comments => 'Program to gather SCOTT''s statistics us a shell script.');**

**-- Stored Procedure with Arguments.**

**DBMS\_SCHEDULER.create\_program (**

**program\_name => 'test\_stored\_procedure\_prog',**

**program\_type => 'STORED\_PROCEDURE',**

**program\_action => 'DBMS\_STATS.gather\_schema\_stats',**

**number\_of\_arguments => 1,**

**enabled => FALSE,**

**comments => 'Program to gather SCOTT''s statistics using a stored procedure.');**

**DBMS\_SCHEDULER.define\_program\_argument (**

**program\_name => 'test\_stored\_procedure\_prog',**

**argument\_name => 'ownname',**

**argument\_position => 1,**

**argument\_type => 'VARCHAR2',**

**default\_value => 'SCOTT');**

**DBMS\_SCHEDULER.enable (name => 'test\_stored\_procedure\_prog');**

**END;**

**/**

**PL/SQL procedure successfully completed.**

**-- Display the program details.**

**SELECT owner, program\_name, enabled FROM dba\_scheduler\_programs;**

OWNER PROGRAM\_NAME ENABL

------------------------------ ------------------------------ -----

SYS PURGE\_LOG\_PROG TRUE

SYS GATHER\_STATS\_PROG TRUE

SYS TEST\_PLSQL\_BLOCK\_PROG TRUE

SYS TEST\_EXECUTABLE\_PROG TRUE

SYS TEST\_STORED\_PROCEDURE\_PROG TRUE

5 rows selected.

Notice how programs that accept arguments must have the arguments defined before they can be enabled.

Programs can be deleted using the DROP\_PROGRAM procedure.

**BEGIN**

**DBMS\_SCHEDULER.drop\_program (program\_name => 'test\_plsql\_block\_prog');**

**DBMS\_SCHEDULER.drop\_program (program\_name => 'test\_stored\_procedure\_prog');**

**DBMS\_SCHEDULER.drop\_program (program\_name => 'test\_executable\_prog');**

**END;**

**/**

PL/SQL procedure successfully completed.

-- Display the program details.

**SELECT owner, program\_name, enabled FROM dba\_scheduler\_programs;**

OWNER PROGRAM\_NAME ENABL

------------------------------ ------------------------------ -----

SYS PURGE\_LOG\_PROG TRUE

SYS GATHER\_STATS\_PROG TRUE

2 rows selected.

**Schedules**

Schedules optionally define the start time, end time and interval related to a job. Schedules are created using the CREATE\_SCHEDULE procedure.

-- Create the schedule.

**BEGIN**

**DBMS\_SCHEDULER.create\_schedule (**

**schedule\_name => 'test\_hourly\_schedule',**

**start\_date => SYSTIMESTAMP,**

**repeat\_interval => 'freq=hourly; byminute=0',**

**end\_date => NULL,**

**comments => 'Repeats hourly, on the hour, for ever.');**

**END;**

**/**

PL/SQL procedure successfully completed.

-- Display the schedule details.

**SELECT owner, schedule\_name FROM dba\_scheduler\_schedules;**

OWNER SCHEDULE\_NAME

------------------------------ ------------------------------

SYS DAILY\_PURGE\_SCHEDULE

SYS TEST\_HOURLY\_SCHEDULE

2 rows selected.

Notice how the interval is defined using the calendaring syntax. This is explained [below](http://www.oracle-base.com/articles/10g/scheduler-10g.php#calendar_syntax_examples).

A schedule can be dropped using the DROP\_SCHEDULE procedure.

**BEGIN**

**DBMS\_SCHEDULER.drop\_schedule (schedule\_name => 'TEST\_HOURLY\_SCHEDULE');**

**END;**

**/**

PL/SQL procedure successfully completed.

-- Display the schedule details.

**SELECT owner, schedule\_name FROM dba\_scheduler\_schedules;**

OWNER SCHEDULE\_NAME

------------------------------ ------------------------------

SYS DAILY\_PURGE\_SCHEDULE

1 row selected.

Schedules don't have to be created as separate objects. They can be defined using the REPEAT\_INTERVAL parameter of the CREATE\_JOB procedure.

**Jobs**

Jobs are what the scheduler is all about. They can either be made up of predefined parts (programs and schedules) or completely self contained depending on which overload of the CREATE\_JOB procedure is used to create them.

**-- Create jobs.**

**BEGIN**

**-- Job defined entirely by the CREATE JOB procedure.**

**DBMS\_SCHEDULER.create\_job (**

**job\_name => 'test\_full\_job\_definition',**

**job\_type => 'PLSQL\_BLOCK',**

**job\_action => 'BEGIN DBMS\_STATS.gather\_schema\_stats(''SCOTT''); END;',**

**start\_date => SYSTIMESTAMP,**

**repeat\_interval => 'freq=hourly; byminute=0',**

**end\_date => NULL,**

**enabled => TRUE,**

**comments => 'Job defined entirely by the CREATE JOB procedure.');**

**-- Job defined by an existing program and schedule.**

**DBMS\_SCHEDULER.create\_job (**

**job\_name => 'test\_prog\_sched\_job\_definition',**

**program\_name => 'test\_plsql\_block\_prog',**

**schedule\_name => 'test\_hourly\_schedule',**

**enabled => TRUE,**

**comments => 'Job defined by an existing program and schedule.');**

**-- Job defined by existing program and inline schedule.**

**DBMS\_SCHEDULER.create\_job (**

**job\_name => 'test\_prog\_job\_definition',**

**program\_name => 'test\_plsql\_block\_prog',**

**start\_date => SYSTIMESTAMP,**

**repeat\_interval => 'freq=hourly; byminute=0',**

**end\_date => NULL,**

**enabled => TRUE,**

**Comments => 'Job defined by existing program and inline schedule.');**

**-- Job defined by existing schedule and inline program.**

**DBMS\_SCHEDULER.create\_job (**

**job\_name => 'test\_sched\_job\_definition',**

**schedule\_name => 'test\_hourly\_schedule',**

**job\_type => 'PLSQL\_BLOCK',**

**job\_action => 'BEGIN DBMS\_STATS.gather\_schema\_stats(''SCOTT''); END;',**

**enabled => TRUE,**

**comments => 'Job defined by existing schedule and inline program.');**

**END;**

**/**

**PL/SQL procedure successfully completed.**

**-- Display job details.**

**SELECT owner, job\_name, enabled FROM dba\_scheduler\_jobs;**

OWNER JOB\_NAME ENABL

------------------------------ ------------------------------ -----

SYS PURGE\_LOG TRUE

SYS GATHER\_STATS\_JOB TRUE

SYS TEST\_FULL\_JOB\_DEFINITION TRUE

SYS TEST\_PROG\_SCHED\_JOB\_DEFINITION TRUE

SYS TEST\_PROG\_JOB\_DEFINITION TRUE

SYS TEST\_SCHED\_JOB\_DEFINITION TRUE

6 rows selected.

Jobs are normally run asynchronously under the control of the job coordinator, but they can be controlled manually using the RUN\_JOB and STOP\_JOB procedures.

**BEGIN**

**-- Run job synchronously.**

**DBMS\_SCHEDULER.run\_job (job\_name => 'test\_full\_job\_definition',**

**use\_current\_session => FALSE);**

**-- Stop jobs.**

**DBMS\_SCHEDULER.stop\_job (job\_name => 'test\_full\_job\_definition, test\_prog\_sched\_job\_definition');**

**END;**

**/**

Jobs can be deleted using the DROP\_JOB procedure.

**BEGIN**

**DBMS\_SCHEDULER.drop\_job (job\_name => 'test\_full\_job\_definition');**

**DBMS\_SCHEDULER.drop\_job (job\_name => 'test\_prog\_sched\_job\_definition');**

**DBMS\_SCHEDULER.drop\_job (job\_name => 'test\_prog\_job\_definition');**

**DBMS\_SCHEDULER.drop\_job (job\_name => 'test\_sched\_job\_definition');**

**END;**

**/**

**PL/SQL procedure successfully completed.**

**-- Display job details.**

**SELECT owner, job\_name, enabled FROM dba\_scheduler\_jobs;**

OWNER JOB\_NAME ENABL

------------------------------ ------------------------------ -----

SYS PURGE\_LOG TRUE

SYS GATHER\_STATS\_JOB TRUE

2 rows selected.

**Job Classes**

Job classes allow grouping of jobs with similar characteristics and resource requirements which eases administration. If the JOB\_CLASS parameter of theCREATE\_JOB procedure is undefined the job is assigned to the DEFAULT\_JOB\_CLASS. A job class is created using the CREATE\_JOB\_CLASS procedure.

-- Display the current resource consumer groups.

**SELECT consumer\_group FROM dba\_rsrc\_consumer\_groups;**

CONSUMER\_GROUP

------------------------------

OTHER\_GROUPS

DEFAULT\_CONSUMER\_GROUP

SYS\_GROUP

LOW\_GROUP

AUTO\_TASK\_CONSUMER\_GROUP

5 rows selected.

-- Create a job class.

**BEGIN**

**DBMS\_SCHEDULER.create\_job\_class (**

**job\_class\_name => 'test\_job\_class',**

**resource\_consumer\_group => 'low\_group');**

**END;**

**/**

PL/SQL procedure successfully completed.

-- Display job class details.

**SELECT job\_class\_name, resource\_consumer\_group FROM dba\_scheduler\_job\_classes;**

JOB\_CLASS\_NAME RESOURCE\_CONSUMER\_GROUP

------------------------------ ------------------------------

DEFAULT\_JOB\_CLASS

AUTO\_TASKS\_JOB\_CLASS AUTO\_TASK\_CONSUMER\_GROUP

TEST\_JOB\_CLASS LOW\_GROUP

3 rows selected.

Jobs can be assigned to a job class either during or after creation using the SET\_ATTRIBUTE procedure.

**BEGIN**

**-- Job defined by an existing program and schedule and assigned to a job class.**

**DBMS\_SCHEDULER.create\_job (**

**job\_name => 'test\_prog\_sched\_class\_job\_def',**

**program\_name => 'test\_plsql\_block\_prog',**

**schedule\_name => 'test\_hourly\_schedule',**

**job\_class => 'test\_job\_class',**

**enabled => TRUE,**

**comments => 'Job defined by an existing program and schedule and assigned toa job class.');**

**DBMS\_SCHEDULER.set\_attribute (**

**name => 'test\_prog\_sched\_job\_definition',**

**attribute => 'job\_class',**

**value => 'test\_job\_class');**

**END;**

**/**

PL/SQL procedure successfully completed.

-- Display job details.

**SELECT owner, job\_name, job\_class, enabled FROM dba\_scheduler\_jobs;**

OWNER JOB\_NAME JOB\_CLASS ENABL

------------------------------ ------------------------------ ------------------------------ -----

SYS PURGE\_LOG DEFAULT\_JOB\_CLASS TRUE

SYS GATHER\_STATS\_JOB AUTO\_TASKS\_JOB\_CLASS TRUE

SYS TEST\_FULL\_JOB\_DEFINITION DEFAULT\_JOB\_CLASS TRUE

SYS TEST\_PROG\_SCHED\_JOB\_DEFINITION TEST\_JOB\_CLASS TRUE

SYS TEST\_PROG\_JOB\_DEFINITION DEFAULT\_JOB\_CLASS TRUE

SYS TEST\_SCHED\_JOB\_DEFINITION DEFAULT\_JOB\_CLASS TRUE

SYS TEST\_PROG\_SCHED\_CLASS\_JOB\_DEF TEST\_JOB\_CLASS TRUE

7 rows selected.

Job classes can be dropped using DROP\_JOB\_CLASS procedure.

**BEGIN**

**DBMS\_SCHEDULER.drop\_job\_class (**

**job\_class\_name => 'test\_job\_class',**

**force => TRUE);**

**END;**

**/**

PL/SQL procedure successfully completed.

-- Display job class details.

**SELECT job\_class\_name, resource\_consumer\_group FROM dba\_scheduler\_job\_classes;**

JOB\_CLASS\_NAME RESOURCE\_CONSUMER\_GROUP

------------------------------ ------------------------------

DEFAULT\_JOB\_CLASS

AUTO\_TASKS\_JOB\_CLASS AUTO\_TASK\_CONSUMER\_GROUP

2 rows selected.

The force option disables any dependent jobs and sets their job class to the default value. If the job class has no dependents the force option is not necessary.

**Windows**

Windows provide the link between the scheduler and the resource manager, allowing different resource plans to be activated at different times. Since job classes point to resource consumer groups, and therefore resource plans, this mechanism allows control over the resources allocated to job classes and their jobs during specific time periods.

Only one window can be active (open) at any time, with one resource plan assigned to the window. The affect of resource plan switches is instantly visible to running jobs which are assigned to job classes. The interaction between the resource manager and the scheduler is beyond the scope of this document.

A window can be created using the CREATE\_WINDOW procedure with a predefined or an inline schedule.

**BEGIN**

**-- Window with a predefined schedule.**

**DBMS\_SCHEDULER.create\_window (**

**window\_name => 'test\_window\_1',**

**resource\_plan => NULL,**

**schedule\_name => 'test\_hourly\_schedule',**

**duration => INTERVAL '60' MINUTE,**

**window\_priority => 'LOW',**

**comments => 'Window with a predefined schedule.');**

**-- Window with an inline schedule.**

**DBMS\_SCHEDULER.create\_window (**

**window\_name => 'test\_window\_2',**

**resource\_plan => NULL,**

**start\_date => SYSTIMESTAMP,**

**repeat\_interval => 'freq=hourly; byminute=0',**

**end\_date => NULL,**

**duration => INTERVAL '60' MINUTE,**

**window\_priority => 'LOW',**

**comments => 'Window with an inline schedule.');**

**END;**

**/**

PL/SQL procedure successfully completed.

-- Display window group details.

**SELECT window\_name, resource\_plan, enabled, active**

**FROM dba\_scheduler\_windows;**

WINDOW\_NAME RESOURCE\_PLAN ENABL ACTIV

------------------------------ ------------------------------ ----- -----

WEEKNIGHT\_WINDOW TRUE FALSE

WEEKEND\_WINDOW TRUE FALSE

TEST\_WINDOW\_1 TRUE FALSE

TEST\_WINDOW\_2 TRUE FALSE

4 rows selected.

Windows can be opened and closed manually using the OPEN\_WINDOW and CLOSE\_WINDOW procedures.

**BEGIN**

**-- Open window.**

**DBMS\_SCHEDULER.open\_window (**

**window\_name => 'test\_window\_2',**

**duration => INTERVAL '1' MINUTE,**

**force => TRUE);**

**END;**

**/**

PL/SQL procedure successfully completed.

-- Display window group details.

**SELECT window\_name, resource\_plan, enabled, active**

**FROM dba\_scheduler\_windows;**

WINDOW\_NAME RESOURCE\_PLAN ENABL ACTIV

------------------------------ ------------------------------ ----- -----

WEEKNIGHT\_WINDOW TRUE FALSE

WEEKEND\_WINDOW TRUE FALSE

TEST\_WINDOW\_1 TRUE FALSE

TEST\_WINDOW\_2 TRUE TRUE

4 rows selected.

**BEGIN**

**-- Close window.**

**DBMS\_SCHEDULER.close\_window (**

**window\_name => 'test\_window\_2');**

**END;**

**/**

PL/SQL procedure successfully completed.

-- Display window group details.

**SELECT window\_name, resource\_plan, enabled, active**

**FROM dba\_scheduler\_windows;**

WINDOW\_NAME RESOURCE\_PLAN ENABL ACTIV

------------------------------ ------------------------------ ----- -----

WEEKNIGHT\_WINDOW TRUE FALSE

WEEKEND\_WINDOW TRUE FALSE

TEST\_WINDOW\_1 TRUE FALSE

TEST\_WINDOW\_2 TRUE FALSE

4 rows selected.

Windows can be dropped using the DROP\_WINDOW procedure.

**BEGIN**

**DBMS\_SCHEDULER.drop\_window (**

**window\_name => 'test\_window\_1',**

**force => TRUE);**

**DBMS\_SCHEDULER.drop\_window (**

**window\_name => 'test\_window\_2',**

**force => TRUE);**

**END;**

**/**

**PL/SQL procedure successfully completed.**

**-- Display window group details.**

**SELECT window\_name, resource\_plan, enabled, active**

**FROM dba\_scheduler\_windows;**

WINDOW\_NAME RESOURCE\_PLAN ENABL ACTIV

------------------------------ ------------------------------ ----- -----

WEEKNIGHT\_WINDOW TRUE FALSE

WEEKEND\_WINDOW TRUE FALSE

2 rows selected.

**Windows Groups**

A window group is a collection of related windows. It can be created with 0, 1 or many windows as group members using the CREATE\_WINDOW\_GROUP procedure.

**BEGIN**

**DBMS\_SCHEDULER.create\_window\_group (**

**group\_name => 'test\_window\_group',**

**window\_list => 'test\_window\_1, test\_window\_2',**

**comments => 'A test window group');**

**END;**

**/**

PL/SQL procedure successfully completed.

-- Display window group details.

**SELECT window\_group\_name, enabled, number\_of\_windowS**

**FROM dba\_scheduler\_window\_groups;**

WINDOW\_GROUP\_NAME ENABL NUMBER\_OF\_WINDOWS

------------------------------ ----- -----------------

MAINTENANCE\_WINDOW\_GROUP TRUE 2

TEST\_WINDOW\_GROUP TRUE 2

2 rows selected.

Windows can be added and removed from a group using the ADD\_WINDOW\_GROUP\_MEMBER and REMOVE\_WINDOW\_GROUP\_MEMBER procedures.

**BEGIN**

**-- Create a new window.**

**DBMS\_SCHEDULER.create\_window (**

**window\_name => 'test\_window\_3',**

**resource\_plan => NULL,**

**schedule\_name => 'test\_hourly\_schedule',**

**duration => INTERVAL '60' MINUTE,**

**window\_priority => 'LOW',**

**comments => 'Window with a predefined schedule.');**

**DBMS\_SCHEDULER.add\_window\_group\_member (**

**group\_name => 'test\_window\_group',**

**window\_list => 'test\_window\_3');**

**END;**

**/**

**PL/SQL procedure successfully completed.**

**-- Display window group members.**

**SELECT window\_group\_name, window\_name**

**FROM dba\_scheduler\_wingroup\_members;**

WINDOW\_GROUP\_NAME WINDOW\_NAME

------------------------------ ------------------------------

MAINTENANCE\_WINDOW\_GROUP WEEKNIGHT\_WINDOW

MAINTENANCE\_WINDOW\_GROUP WEEKEND\_WINDOW

TEST\_WINDOW\_GROUP TEST\_WINDOW\_1

TEST\_WINDOW\_GROUP TEST\_WINDOW\_2

TEST\_WINDOW\_GROUP TEST\_WINDOW\_3

5 rows selected.

**BEGIN**

**DBMS\_SCHEDULER.remove\_window\_group\_member (**

**group\_name => 'test\_window\_group',**

**window\_list => 'test\_window\_3');**

**END;**

**/**

PL/SQL procedure successfully completed.

-- Display window group members.

**SELECT window\_group\_name, window\_name**

**FROM dba\_scheduler\_wingroup\_members;**

WINDOW\_GROUP\_NAME WINDOW\_NAME

------------------------------ ------------------------------

MAINTENANCE\_WINDOW\_GROUP WEEKNIGHT\_WINDOW

MAINTENANCE\_WINDOW\_GROUP WEEKEND\_WINDOW

TEST\_WINDOW\_GROUP TEST\_WINDOW\_1

TEST\_WINDOW\_GROUP TEST\_WINDOW\_2

4 rows selected.

Window groups can be dropped using the DROP\_WINDOW\_GROUP procedure.

**BEGIN**

**DBMS\_SCHEDULER.drop\_window\_group (**

**group\_name => 'test\_window\_group',**

**force => TRUE);**

**END;**

**/**

PL/SQL procedure successfully completed.

-- Display window group details.

**SELECT window\_group\_name, enabled, number\_of\_windowS**

**FROM dba\_scheduler\_window\_groups;**

WINDOW\_GROUP\_NAME ENABL NUMBER\_OF\_WINDOWS

------------------------------ ----- -----------------

MAINTENANCE\_WINDOW\_GROUP TRUE 2

1 row selected.

The force option must be used if the window group currently has members.

**Enable, Disable and Attributes**

All applicable scheduler objects can be enabled and disabled using the overloaded ENABLE and DISABLE procedures.

**BEGIN**

**-- Enable programs and jobs.**

**DBMS\_SCHEDULER.enable (name => 'test\_stored\_procedure\_prog');**

**DBMS\_SCHEDULER.enable (name => 'test\_full\_job\_definition');**

**-- Disable programs and jobs.**

**DBMS\_SCHEDULER.disable (name => 'test\_stored\_procedure\_prog');**

**DBMS\_SCHEDULER.disable (name => 'test\_full\_job\_definition');**

**END;**

**/**

The values for individual attributes of all scheduler objects can be altered using one of the SET\_ATTRIBUTE overloads.

**BEGIN**

**DBMS\_SCHEDULER.set\_attribute (**

**name => 'hourly\_schedule',**

**attribute => 'repeat\_interval',**

**value => 'freq=hourly; byminute=30');**

**END;**

**/**

The values can be set to NULL using the SET\_ATTRIBUTE\_NULL procedure.

**BEGIN**

**DBMS\_SCHEDULER.set\_attribute\_null (**

**name => 'hourly\_schedule',**

**attribute => 'repeat\_interval');**

**END;**

**/**

**Configuring The Scheduler**

The SCHEDULER\_ADMIN role gives a user the ability to control every aspect of the scheduler, as well as generating jobs to run as any other user. For this reason you should avoid granting it to anyone other than trusted DBAs.

For the majority of users, the CREATE JOB privilege will be sufficient.

For users requiring some level of scheduler administrative privileges, the MANAGE SCHEDULER privilege allows them to create additional scheduler objects, as well as allowing them to set and retrieve scheduler attributes using the SET\_SCHEDULER\_ATTRIBUTE and GET\_SCHEDULER\_ATTRIBUTE procedures.

**Calendar Syntax Examples**

The basic calendaring syntax is shown below.

repeat\_interval = frequency\_clause

[; interval=?] [; bymonth=?] [; byweekno=?]

[; byyearday=?] [; bymonthday=?] [; byday=?]

[; byhour=?] [; byminute=?] [; bysecond=?]

frequency\_clause = "FREQ" "=" frequency

frequency = "YEARLY" | "MONTHLY" | "WEEKLY" | "DAILY" |

"HOURLY" | "MINUTELY" | "SECONDLY"

For a full syntax breakdown, see [this](http://docs.oracle.com/cd/B13789_01/appdev.101/b10802/d_sched.htm#1010077) and [this](http://docs.oracle.com/cd/B13789_01/server.101/b10739/scheduse.htm#i1023132).

The easiest way to get to grips with the calendaring syntax is by example, so this section presents several examples of how the syntax is used to schedule jobs at different intervals. The date and timestamp intervals are also listed for the sake of comparison.

The [test\_calendar\_string.sql](http://www.oracle-base.com/dba/script.php?category=10g&file=test_calendar_string.sql) script is quite useful for testing calendaring syntax strings.

* [Every day](http://www.oracle-base.com/articles/10g/scheduler-10g.php#every_day)
* [Every day, at midnight](http://www.oracle-base.com/articles/10g/scheduler-10g.php#every_day_midnight)
* [Every day, at 06:00](http://www.oracle-base.com/articles/10g/scheduler-10g.php#every_day_600)
* [Every hour](http://www.oracle-base.com/articles/10g/scheduler-10g.php#every_hour)
* [Every hour, on the hour](http://www.oracle-base.com/articles/10g/scheduler-10g.php#every_hour_exact)
* [Every minute](http://www.oracle-base.com/articles/10g/scheduler-10g.php#every_minute)
* [Every minute, on the minute](http://www.oracle-base.com/articles/10g/scheduler-10g.php#every_minute_exact)
* [Every 5 minutes](http://www.oracle-base.com/articles/10g/scheduler-10g.php#every_5_minutes)
* [Every Monday at 09:00](http://www.oracle-base.com/articles/10g/scheduler-10g.php#every_monday_900)
* [Every Monday, Wednesday and Friday at 06:00](http://www.oracle-base.com/articles/10g/scheduler-10g.php#every_mon_wed_fri_600)
* [First Monday of each quarter](http://www.oracle-base.com/articles/10g/scheduler-10g.php#first_mon_in_quarter)

**Every day**

Repeat interval using calendaring syntax.

'freq=daily;'

Repeat interval using dates and timestamps.

'sysdate + 1'

'systimestamp + 1'

'sysdate + interval ''1'' day'

'systimestamp + interval ''1'' day'

**Every day, at midnight**

Repeat interval using calendaring syntax.

'freq=daily; byhour=0; byminute=0; bysecond=0;'

Repeat interval using dates and timestamps.

'trunc(sysdate) + 1'

'trunc(systimestamp) + 1'

'trunc(sysdate) + interval ''1'' day'

'trunc(systimestamp) + interval ''1'' day'

**Every day, at 06:00**

Repeat interval using calendaring syntax.

'freq=daily; byhour=6; byminute=0; bysecond=0;'

Repeat interval using dates and timestamps.

'trunc(sysdate) + 1 + 6/24'

'trunc(systimestamp) + 1 + 6/24'

'trunc(sysdate) + interval ''1 6'' day to hour '

'trunc(systimestamp) + interval ''1 6'' day to hour'

**Every hour**

Repeat interval using calendaring syntax.

'freq=hourly;'

Repeat interval using dates and timestamps.

'sysdate + 1/24'

'systimestamp + 1/24'

'sysdate + interval ''1'' hour'

'systimestamp + interval ''1'' hour'

**Every hour, on the hour**

Repeat interval using calendaring syntax.

'freq=hourly; byminute=0; bysecond=0;'

Repeat interval using dates and timestamps.

'trunc(sysdate, ''HH24'') + 1/24'

'trunc(systimestamp, ''HH24'') + 1/24'

'trunc(sysdate, ''HH24'') + interval ''1'' hour'

'trunc(systimestamp, ''HH24'') + interval ''1'' hour'

**Every minute**

Repeat interval using calendaring syntax.

'freq=minutely;'

Repeat interval using dates and timestamps.

'sysdate + 1/24/60'

'systimestamp + 1/24/60'

'sysdate + interval ''1'' minute'

'systimestamp + interval ''1'' minute'

**Every minute, on the minute**

Repeat interval using calendaring syntax.

'freq=minutely; bysecond=0;'

Repeat interval using dates and timestamps.

'trunc(sysdate, ''MI'') + 1/24/60'

'trunc(systimestamp, ''MI'') + 1/24/60'

'trunc(sysdate, ''MI'') + interval ''1'' minute'

'trunc(systimestamp, ''MI'') + interval ''1'' minute'

**Every 5 minutes**

Repeat interval using calendaring syntax.

'freq=minutely; interval=5; bysecond=0;'

Repeat interval using dates and timestamps.

'trunc(sysdate, ''MI'') + 5/24/60'

'trunc(systimestamp, ''MI'') + 5/24/60'

'trunc(sysdate, ''MI'') + interval ''5'' minute'

'trunc(systimestamp, ''MI'') + interval ''5'' minute'

**Every Monday at 09:00**

Repeat interval using calendaring syntax.

'freq=weekly; byday=mon; byhour=9; byminute=0; bysecond=0;'

Repeat interval using dates and timestamps.

'trunc(next\_day(sysdate, ''MONDAY'')) + 9/24'

'trunc(next\_day(systimestamp, ''MONDAY'')) + 9/24'

'trunc(next\_day(sysdate, ''MONDAY'')) + interval ''9'' hour'

'trunc(next\_day(systimestamp, ''MONDAY'')) + interval ''9''hour'

**Every Monday, Wednesday and Friday at 06:00**

Repeat interval using calendaring syntax.

'freq=weekly; byday=mon,wed,fri; byhour=6; byminute=0; bysecond=0;'

Repeat interval using dates and timestamps.

'trunc(least(next\_day(sysdate, ''monday''), next\_day(sysdate, ''wednesday''), next\_day(sysdate, ''friday''))) + (6/24)'

'trunc(least(next\_day(systimestamp, ''monday''), next\_day(systimestamp, ''wednesday''), next\_day(systimestamp, ''friday''))) + (6/24)'

'trunc(least(next\_day(sysdate,''monday''), next\_day(sysdate, ''wednesday''), next\_day(sysdate, ''friday''))) + interval ''6'' hour'

'trunc(least(next\_day(systimestamp, ''monday''), next\_day(systimestamp, ''wednesday''), next\_day(systimestamp, ''friday''))) + interval ''6'' hour'

**First Monday of each quarter**

Repeat interval using calendaring syntax.

'freq=monthly; bymonth=1,4,7,10; byday=1mon'

Repeat interval using dates and timestamps.

'next\_day(add\_months(trunc(sysdate, ''q''), 3), ''monday'')'

'next\_day(add\_months(trunc(systimestamp, ''q''), 3), ''monday'')'

**Extracting DDL**

The script used to create scheduler objects can be extracted using the [DBMS\_METADATA](http://www.oracle-base.com/articles/9i/dbms_metadata.php) package, as shown in the following example.

CONN test/test

BEGIN

DBMS\_SCHEDULER.CREATE\_JOB (

job\_name => 'MY\_TEST\_JOB',

job\_type => 'PLSQL\_BLOCK',

job\_action => 'BEGIN NULL; END;',

start\_date => TRUNC(SYSDATE),

repeat\_interval => 'FREQ=monthly;BYDAY=SUN;BYHOUR=22;BYMINUTE=0;BYSECOND=0');

END;

/

SET LONG 100000

SELECT DBMS\_METADATA.get\_ddl('PROCOBJ','MY\_TEST\_JOB', 'TEST') AS job\_def FROM dual;

JOB\_DEF

--------------------------------------------------------------------------------

BEGIN

dbms\_scheduler.create\_job('"MY\_TEST\_JOB"',

job\_type=>'PLSQL\_BLOCK', job\_action=>

'BEGIN NULL; END;'

, number\_of\_arguments=>0,

start\_date=>TO\_TIMESTAMP\_TZ('04-APR-2012 12.00.00.000000000 AM +01:00','DD-MON-R

RRR HH.MI.SSXFF AM TZR','NLS\_DATE\_LANGUAGE=english'), repeat\_interval=>

'FREQ=monthly;BYDAY=SUN;BYHOUR=22;BYMINUTE=0;BYSECOND=0'

, end\_date=>NULL,

JOB\_DEF

--------------------------------------------------------------------------------

job\_class=>'"DEFAULT\_JOB\_CLASS"', enabled=>FALSE, auto\_drop=>TRUE,comments=>

NULL

);

COMMIT;

END;

SQL>

Clean up the test job using the floowing.

EXEC DBMS\_SCHEDULER.drop\_job('MY\_TEST\_JOB');

**Scheduler Views**

A number of DBA\_SCHEDULER\_%, ALL\_SCHEDULER\_% and USER\_SCHEDULER\_% views are available to display information about scheduler objects. In Oracle 10g, the following views are present.

* [DBA\_SCHEDULER\_JOB\_ARGS](http://docs.oracle.com/cd/E11882_01/server.112/e25513/statviews_2041.htm#i1586875)
* [DBA\_SCHEDULER\_JOB\_CLASSES](http://docs.oracle.com/cd/E11882_01/server.112/e25513/statviews_2042.htm#i1586977)
* [DBA\_SCHEDULER\_JOB\_LOG](http://docs.oracle.com/cd/E11882_01/server.112/e25513/statviews_2044.htm#i1587038)
* [DBA\_SCHEDULER\_JOB\_RUN\_DETAILS](http://docs.oracle.com/cd/E11882_01/server.112/e25513/statviews_2045.htm#i1587156)
* [DBA\_SCHEDULER\_JOBS](http://docs.oracle.com/cd/E11882_01/server.112/e25513/statviews_2046.htm#i1587306)
* [DBA\_SCHEDULER\_PROGRAM\_ARGS](http://docs.oracle.com/cd/E11882_01/server.112/e25513/statviews_2048.htm#i1587654)
* [DBA\_SCHEDULER\_PROGRAMS](http://docs.oracle.com/cd/E11882_01/server.112/e25513/statviews_2049.htm#i1587769)
* [DBA\_SCHEDULER\_RUNNING\_JOBS](http://docs.oracle.com/cd/E11882_01/server.112/e25513/statviews_2053.htm#I1020387)
* [DBA\_SCHEDULER\_SCHEDULES](http://docs.oracle.com/cd/E11882_01/server.112/e25513/statviews_2054.htm#i1587874)
* [DBA\_SCHEDULER\_WINDOW\_DETAILS](http://docs.oracle.com/cd/E11882_01/server.112/e25513/statviews_2055.htm#i1588021)
* [DBA\_SCHEDULER\_WINDOW\_GROUPS](http://docs.oracle.com/cd/E11882_01/server.112/e25513/statviews_2056.htm#i1587960)
* [DBA\_SCHEDULER\_WINDOW\_LOG](http://docs.oracle.com/cd/E11882_01/server.112/e25513/statviews_2057.htm#i1588122)
* [DBA\_SCHEDULER\_WINDOWS](http://docs.oracle.com/cd/E11882_01/server.112/e25513/statviews_2058.htm#i1588223)
* [DBA\_SCHEDULER\_WINGROUP\_MEMBERS](http://docs.oracle.com/cd/E11882_01/server.112/e25513/statviews_2059.htm#i1588366)

The DBA\_SCHEDULER\_JOB\_RUN\_DETAILS view is especially interesting as it provides a history of the job runs, including the status of the run and error messages associated with failed runs.

Later database releases include additional views to support new functionality.

--------------------------<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>--------------------------------------------

**Another Example**

v\_jobnam := v\_objnam;

v\_jobnam := DBMS\_SCHEDULER.generate\_job\_name (v\_jobnam);

v\_startdate := to\_timestamp(v\_startdate);

select sysdate + (v\_delhrs/1440) - - -v\_delhrs is Number

into v\_startdate

from dual;

DBMS\_SCHEDULER.CREATE\_JOB(job\_name => v\_jobnam, job\_type => 'PLSQL\_BLOCK', JOB\_ACTION => 'BEGIN DROP\_OBJ1(' || v\_objnam|| ', ' || v\_objtyp || ', '|| v\_schema || ',' || v\_objid ||'); END;', start\_date => SYSTIMESTAMP, repeat\_interval => 'freq=secondly; bysecond=0', end\_date => NULL, enabled => TRUE, comments => 'Calls PLSQL once');

DBMS\_SCHEDULER.CREATE\_JOB(job\_name => v\_jobnam,

job\_type => 'PLSQL\_BLOCK',

JOB\_ACTION => 'BEGIN DROP\_OBJ1(''' || v\_objnam

|| ''', ''' || v\_objtyp || ''', '''

|| v\_schema || ''',' || v\_objid

|| '); END;',

start\_date => SYSTIMESTAMP,

repeat\_interval => 'freq=secondly; bysecond=0',

end\_date => NULL,

enabled => TRUE,

comments => 'Calls PLSQL once');

------------------------<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>>>>>>>>>-------------------------------------

**DBMS\_SCHEDULER: Learn with Example**

Published September 29, 2009 | By [*admin*](http://www.oraclecity.com/author/admin/)

This is very useful Oracle supplied package which is used to run scheduled job in Oracle databases. Released with Oracle 10g this package replaces DBMS\_JOB which was available with previous version of Oracle databases. DBMS\_JOB still runs in Oracle 10g and even in 11g but that is used only for backward compatibility and should not be used in developing new solutions. Where possible it is advisable to convert legacy jobs which uses DBMS\_JOB to use more powerful and robust DBMS\_SCHEDULER.

Like DBMS\_JOB package, the DBMS\_SCHEDULER is a collection of functions and procedures intended to make job scheduling in Oracle more robust and easy to use.

PRIVILEGES:  
===========================

To create and runs job in Oracle database you will need CREATE JOB role. To perform administrative tasks you will need SCHEDULER\_ADMIN role.

Depending on the requirement and your installation environment you may like to grant the following privileges:

CREATE ANY JOBS  
CREATE EXTERNAL JOBS

CREATING A SCHEDULED JOB:  
===========================

The package is supplied to help perform some database tasks such as running a stored procedure or package at some given time and at some given interval.

And example is here below:

dbms\_scheduler.create\_job( job\_name=>’Myschema.Daily\_Emp\_Report’,  
job\_type=>’STORED\_PROCEDURE’,  
job\_action=>’Myschema.p\_daily\_emp\_teport’,  
number\_of\_arguments=>0,  
start\_date=>TRUNC(SYSDATE),         repeat\_interval=>’FREQ=WEEKLY;BYDAY=SUN;BYHOUR=07;BYMINUTE=00;  BYSECOND=00′,  
end\_date=>NULL,  
job\_class=> ‘DEFAULT\_JOB\_CLASS’,  
enabled=>TRUE,  
auto\_drop=>FALSE,  
comments=>NULL);

In this example I am creating a scheduled Oracle job named Myschema.Daily\_Emp\_Report. When run, this job will execute a STORED PROCEDURE called p\_daily\_emp\_teport. The stored procedure p\_daily\_emp\_teport resides in a schema called Myschema. The job will run on every week Sunday at 07:00.

Here it should be noted that the stored procedure to be run can be in any schema as long as the owner of the scheduler job has execute privilege to run the stored procedure.

The most interesting part in the above command is the parameter called repeat\_interval. This will dictate when and how often this job is needed to run. In the above example of it is very clear we want to run the job only once a week which is on Sunday at 7 AM.

Some example parameter repeat\_interval:

1. repeat\_interval=>’FREQ=daily;byhour=20;byminute=0;bysecond=0′.  
The job will runs daily from Monday to Sunday at 20:00 hours.

2. repeat\_interval=>’FREQ=daily;byhour=20;byminute=30;bysecond=0′.  
The job will runs daily from Monday to Sunday at 20:30 hours.

3. repeat\_interval=>’FREQ=monthly;bymonthday=10,20; byminute=0;bysecond=0;’  
The job will on 10th and 20th day of each month.

4. repeat\_interval=>’FREQ=yearly;bymonth=JAN,APR,SEP,DEC;BYMONTHDAY=21;  
byhour=20;byminute=30;bysecond=0′ The job will run on 21st of months  
JAN, APR, SEP and DEC at 20:30 hours.

UPDATING A SCHEDULED JOB  
===========================

Sometime you may find yourself in a situation where you would like to change a parameter value of the scheduled job after you have actually created it. The easiest way to do that might be just drop the job and re-create it with the correct parameter values. But I would say that will be a drastic action to take and in some situation (when it is in production environment) it just might not be the appropriate action and your manager might be really feel uneasy in approving such steps, quite rightly.

That’s why Oracle has provided ways to change a parameter value without actualling dropping anything. An example is given below. Here I am changing my job Myschema.Daily\_Emp\_Report to run a new stored procedure called Myschema.p\_monthly\_emp\_teport instead of running Myschema.p\_daily\_emp\_teport:

DBMS\_SCHEDULER.SET\_ATTRIBUTE (  
name=>’Myschema.Daily\_Emp\_Report’,  
attribute=>’job\_action’,  
value=>’Myschema.p\_monthly\_emp\_teport’);

STOPPING A SCHEDULED JOB  
===========================

If you need just to stop a scheduler job you can do so my executing the package

DBMS\_SCHEDULER.STOP\_JOB (  
job\_name IN VARCHAR2  
force IN BOOLEAN DEFAULT FALSE  
commit\_semantics IN VARCHAR2 DEFAULT ‘STOP\_ON\_FIRST\_ERROR’);

where  
job\_name -> is the name of your job to be stopped  
force -> is either TRUE (terminate immediately) or FALSE (stops the job gracefully)  
commit\_semantics -> accepted value STOP\_ON\_FIRST\_ERROR or ABSORB\_ERRORS

Example: DBMS\_SCHEDULER.STOP\_JOB(job\_name=>’Myschema.Daily\_Emp\_Report’,  
force=>’TRUE’,  commit\_semantics=>’ABSORB\_ERRORS’):

The preceding command will stop the job Myschema.Daily\_Emp\_Report immediately and will try to absorbs error and commit all stop operations that were successful.

DROPPING A SCHEDULER JOB  
===============================

The command to drop a scheduler job is

exec dbms\_scheduler.drop\_job(‘myjobname’);

USEFUL DATA DICTIONARY OBJECTS  
========================================

DBA\_SCHEDULER\_PROGRAMS  
DBA\_SCHEDULER\_JOBS  
DBA\_SCHEDULER\_JOB\_ROLES  
DBA\_SCHEDULER\_JOB\_CLASSES  
DBA\_SCHEDULER\_WINDOWS  
DBA\_SCHEDULER\_PROGRAM\_ARGS  
DBA\_SCHEDULER\_JOB\_ARGS  
DBA\_SCHEDULER\_JOB\_RUN\_DETAILS  
DBA\_SCHEDULER\_JOB\_LOG  
DBA\_SCHEDULER\_WINDOW\_LOG  
DBA\_SCHEDULER\_WINDOW\_DETAILS  
DBA\_SCHEDULER\_WINDOW\_GROUPS  
DBA\_SCHEDULER\_WINGROUP\_MEMBERS  
DBA\_SCHEDULER\_WINGROUP\_MEMBERS  
DBA\_SCHEDULER\_SCHEDULES  
DBA\_SCHEDULER\_RUNNING\_JOBS  
DBA\_SCHEDULER\_REMOTE\_DATABASES  
DBA\_SCHEDULER\_REMOTE\_JOBSTATE  
DBA\_SCHEDULER\_GLOBAL\_ATTRIBUTE  
DBA\_SCHEDULER\_CHAINS  
DBA\_SCHEDULER\_CHAIN\_RULES  
DBA\_SCHEDULER\_CHAIN\_STEPS  
DBA\_SCHEDULER\_RUNNING\_CHAINS  
DBA\_SCHEDULER\_CREDENTIALS

----------------------------------------<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>--------------------------------------------

Here is a simple job.

SQL> create table log1 (ts timestamp)

2 /

Table created.

SQL> create or replace procedure printe as

2 begin

3 insert into log1 values (systimestamp);

4 commit;

5 end;

6 /

Procedure created.

SQL>

So the first thing is to submit it with both the start time and interval correctly specified. If you cannot remember how many minutes there are in a day (1440) it is a good idea to use brackets. Let's compare submitting the job with your date specifications ...

SQL> var job\_no number

SQL> BEGIN

2 DBMS\_JOB.SUBMIT

3 (

4 job =>:job\_no,

5 WHAT=>'printe;',--Procedure

6 next\_date=>sysdate+1/24\*60,

7 interval=>'sysdate+1/24\*60'

8 );

9 commit;

10 END;

11 /

PL/SQL procedure successfully completed.

SQL> print job\_no

JOB\_NO

----------

71

SQL>

... with brackets to assert precedence ...

SQL> BEGIN

2 DBMS\_JOB.SUBMIT

3 (

4 job =>:job\_no,

5 WHAT=>'printe;',--Procedure

6 next\_date=>sysdate+1/(24\*60),

7 interval=>'sysdate+1/(24\*60)'

8 );

9 commit;

10 END;

11 /

PL/SQL procedure successfully completed.

SQL> print job\_no

JOB\_NO

----------

72

SQL>

Clearly job 71 has not run and isn't going to run for some time yet:

SQL> select job, what, last\_date, next\_date, interval

2 from user\_jobs

3 where job in (71,72)

4 /

JOB WHAT LAST\_DATE NEXT\_DATE INTERVAL

------ ------------ -------------------- -------------------- -----------------

71 printe; 05-MAY-2010 17:35:34 sysdate+1/24\*60

72 printe; 03-MAY-2010 05:44:42 03-MAY-2010 05:45:34 sysdate+1/(24\*60)

SQL>

Monitoring job 72 ....

SQL> select \* from log1

2 /

TS

-------------------------------------------------------------------

03-MAY-10 05:43:39.250000

03-MAY-10 05:44:42.296000

SQL>

So, if this still isn't working for you, what should you be doing? The first thing is to check whether the database is configured to run jobs at all. You will need DBA access for this.

SQL> select value

2 from v$parameter

3 where name='job\_queue\_processes'

4 /

VALUE

-------------------------

1000

SQL>

If I remember correctly, in Oracle 9i the default value for this parameter is 0. It needs to be set to some non-zero value for jobs to run.

And if that isn't the problem you need to check for error messages in the alert log. Thebackground\_dump\_dest directory may also have some .trc files produced by a failing job.

----------------------------------<<<<<<<<<<<<<<<<<<<>>>>>>>>>>>>>>>>--------------------------------------------

OK, then try the following:  
Change:

,job\_type        => 'STORED\_PROCEDURE'

into:

,job\_type        => 'PLSQL\_BLOCK'

SQL> begin

  2

  3      DBMS\_SCHEDULER.CREATE\_JOB

  4              (

  5                 job\_name        => 'sfcia\_drop\_acq\_txn\_1'

  6                ,start\_date      =>  sysdate

  7                ,repeat\_interval => 'FREQ=DAILY; INTERVAL=1'

  8                ,end\_date        => NULL

  9                ,job\_class       => 'DEFAULT\_JOB\_CLASS'

10                ,job\_type        => 'PLSQL\_BLOCK'

11                ,job\_action      => 'BEGIN PRC\_DROP\_ACQ\_TXN(TO\_DATE(''07-JAN-2012'',''DD-MON-YYYY''),''NFSA'',''NFS''); END;'

12                ,enabled         =>  TRUE

13              );

14

15      end;

16  /

PL/SQL procedure successfully completed.