

## **EXAM OBJECTIVE : SIMPLIFY MANAGEMENT TASKS WITH THE SCHEDULER**

### OVERVIEW OF THE SCHEDULER

Organizations may have tasks to be both performed periodically and repetitively and there is a need for an effective scheduling solution that will allow organizations to automate the business process so that they can take place without manual intervention.

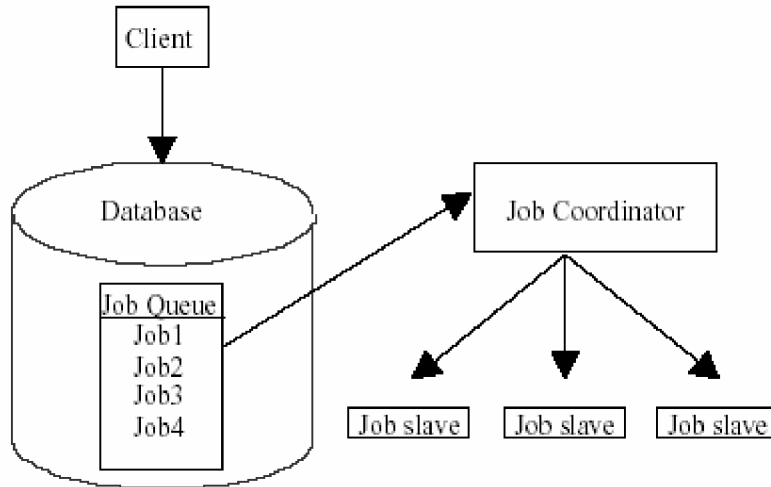
Oracle 10g provides a solution to this requirement by means of the new feature called the **Oracle Scheduler**. The Scheduler enables limited computing resources to be allocated appropriately among competing jobs, thus aligning job processing with business needs.

The scheduler is a collection of functions and procedures that can be executed from any PL/SQL program. The collection of procedures and functions are part of the DBMS\_SCHEDULER package. Because the scheduler is part of the database it is platform independent.

The scheduler can be used to:

- Schedule job execution based on time: This is the ability to schedule a job to run at a particular date and time. For e.g. you can schedule the application of a certain patch during non-peak hours. This can be done automatically by the scheduler without any manual intervention.
- Reuse existing programs and schedules: An entity called a program that consists of metadata about what will be run by the scheduler can be created and stored in the database. Once created and stored as an object in the database the program can be re-used whenever required.

## SCHEDULER ARCHITECTURE



When a job is created, information about the job is placed in a job table in the data dictionary. The Scheduler uses one job table for each database and one job coordinator for each instance. The job coordinator is a **background process (cjqNnn)** that is automatically started when jobs must be run, or windows must be opened. It goes to sleep automatically after a sustained period of Scheduler inactivity.

When it is time for a job to be executed, the job coordinator notifies an available job slave process, which then executes the job. The job slave gathers metadata about how to run the job from the job table and then starts a database session and transaction as the job owner to execute the job. You do not need to configure when the job coordinator checks the job table; the system chooses the time frame automatically.

There is one job coordinator process per instance.

The job coordinator:

- Controls and spawns the job slaves
- Queries the job table
- Picks up jobs from the job table on a regular basis and places them in a memory cache
- Improves performance by avoiding going to the disk
- Takes jobs from the memory cache and passes them to job slaves for execution
- Cleans up the job slave pool when slaves are no longer needed
- Goes to sleep when no jobs are scheduled
- Wakes up when a job needs executing or a new job is created.

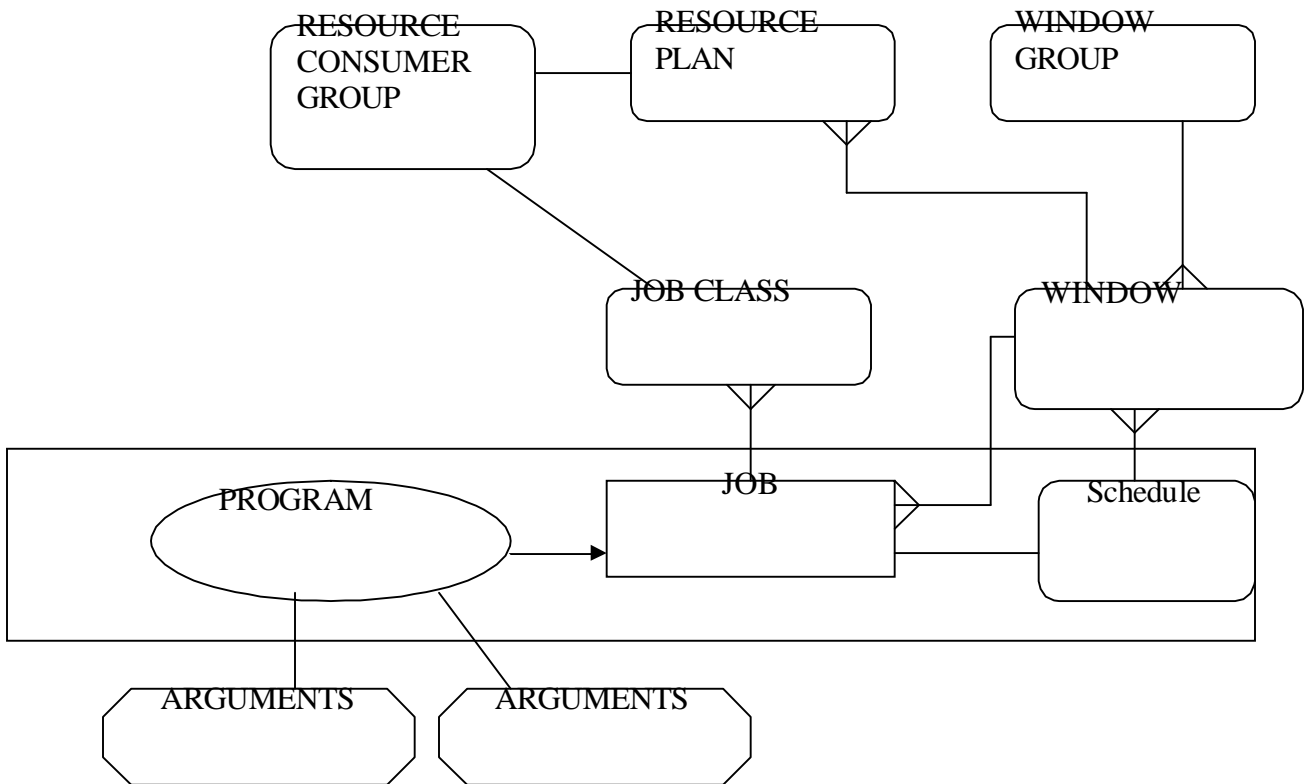
When a job is done, the job slaves:

- Update the entry in the job table to the COMPLETED status
- Insert an entry into the job log table
- Update the run count
- Clean up and then look for new work (if none, they go to sleep)

There are two ways to control the number of spawned job slave processes:

- Use the default Oracle database control mechanisms
- Set the Scheduler attribute `max_job_slave_processes` to the desired value

**EXAM OBJECTIVE: CREATE A JOB, PROGRAM, SCHEDULE AND WINDOW**



**BASIC SCHEDULER COMPONENTS**

The scheduler uses the following basic components:

- Schedule
- Program
- Jobs
  
- **A schedule** specifies when and how many times a job is to be executed. Jobs can be scheduled for processing immediately or at a later time. You can clearly

indicate when a job is to be executed in terms of month, day, hours etc. Jobs can be made to repeat over a period of time and you can also indicate when a schedule expires. Schedules have a time limit associated with them. Schedules can be created as database entities and saved in the database to be used for multiple jobs. For e.g. an end of quarter may be a common schedule for many jobs. An example of a schedule can be 'Run every Friday, at 5 pm, starting on July 1<sup>st</sup>, 2004 and ending on July 31<sup>st</sup>, 2004'.

- **A program** is a collection of metadata or control information about what will be run by the scheduler. Every automated job involves a particular executable such as PL/SQL stored procedure, a native binary executable or even a shell script. For the scheduler, a program is a collection of metadata about the particular executable. It includes information such as the program name, the type of program (e.g. PL/SQL or Java stored procedure) and arguments that are passed to the program.

A program is a separate entity. Different user can use the same program without having to redefine it.

- **A job** is a user-defined task that is scheduled to run one or more times. Job is a combination of what needs to be executed (action) and when (the schedule). An example of a job is 'Gather statistics on certain highly volatile tables, every 3 hours'.

A job instance represents a specific run of a job. Jobs that are scheduled to run only once will have only one instance; however jobs that have a repeating schedule will have multiple instances, with each run representing an instance.

#### How programs, jobs, and schedules are related

To determine when, where and what will be executed in the database, you need to assign relationships among programs, jobs and schedules.

A program called P1 that 'analyzes a table using the DBMS\_STATS package'. P1 can receive a tablename as an argument.

- S1 is a schedule that takes the value of 3 p.m.
- Two jobs called J1 and J2 both point to the program P1 however each job supplies a different tablename to the argument received by P1. The jobs J1 and J2 are scheduled to run at S1.

The tables will be analyzed at 3 p.m.

## **OTHER SCHEDULER COMPONENTS**

Here are mentioned some advanced components that can be configured for the scheduler.

- JOB CLASSES
- WINDOWS
- WINDOWS GROUPS
- **A job class:** Is a way of grouping jobs into larger entities. It defines a category of jobs that have the same resource usage requirements. A job can belong to only job class at a time. It enables you to prioritize access to job slaves among the job classes. For e.g. a certain manager's request will always begin before routine tasks. A job class can also be used to set job characteristics or behavior for a group of jobs. Jobs can be classified based on functionality. For e.g jobs that access similar data such as Marketing, production, sales and so on. You can specify attributes at a class level, such as a purging policy for all payroll jobs. You can also specify the order in which a job is started within a job class.
- **A window:** represents an interval of time with well defined beginning and end times. A window is used to change resource allocation during a time period. You create a window to control which groups of users have what level of priority. A window can have the following attributes: Name that uniquely identifies the window, Time interval and an optional resource manager plan which automatically becomes active when the window opens. When the window is closed, the plan that was active at the beginning gets restored. An example of a window is "from 9AM – 5PM". You can assign a priority for each window. If windows overlap, the window with the highest priority is chosen over other windows with lower priorities.
- **A window group:** represents a group of windows , created for ease of use. An example would be to combine individual windows called weekends, nights and holidays into a group called "AFTERHOURS".

#### AN EXAMPLE OF AUTOMATIC GATHER STATISTICS JOB

The GATHER\_STATS\_JOB job is created at database creation time. This task executes the DBMS\_STATS.GATHER\_DATABASE\_STATS\_JOB\_PROC procedure and uses the scheduler.

Two windows that exist are the WEEKNIGHT\_WINDOW and a WEEKEND\_WINDOW.

WEEKNIGHT\_WINDOW – predefined between 10 p.m to 6.a.m everyday from Monday to Friday.

WEEKEND\_WINDOW – is predefined between 12 a.m on Saturday and 12 a.m on Monday.

A default window group called MAINTENANCE\_WINDOW\_GROUP is predefined to contain the above windows.

The GATHER\_STATS\_JOB uses a specific scheduler class, called AUTO\_TASKS\_JOB\_CLASS.

This class is automatically created by the Oracle database and is associated with a specific resource consumer group, called the AUTO\_TASKS\_CONSUMER\_GROUP. This ensures that the task uses the AUTO\_TASKS\_CONSUMER\_GROUP.

If you want to control resources used by GATHER\_STATS\_JOB then all you need to do is define a resource manager plan for MAINTENANCE\_WINDOW\_GROUP that allocates resources for AUTO\_TASKS\_CONSUMER\_GROUP.

## **CREATING SCHEDULER COMPONENTS**

You can use either Oracle-supplied PL/SQL packages or the Enterprise Manager (EM) console to create and manage scheduler components.

You can add your own customized tasks to existing ones in Enterprise Manager by adding a new job using the Scheduler. Select **the Database Home Page -> Administration tab -> Jobs -> Create button.**

Enter the name of the new task. Specify the program to execute and choose the schedule for the new task.

## CREATING A PROGRAM

Programs are created using the CREATE\_PROGRAM procedure. The metadata can contain information such as:

- A program name
- The type of program. For example, SQL script, stored procedure, C executable, shell script, and so on. The program\_type argument must be one of the following supported types:
  - PLSQL\_BLOCK
  - STORED\_PROCEDURE
  - EXECUTABLE
- Program arguments – You can define program arguments with the DEFINE\_PROGRAM\_ARGUMENT procedure. You can set a job argument values with the SET\_JOB\_ARGUMENT\_VALUE procedure.  
The scheduler allows you to optionally create programs which hold metadata about a task, but no schedule information. A program may relate to a PL/SQL block, a stored procedure or an OS executable file.

The following example displays the creation of a program that contains metadata about the execution of an anonymous PL/SQL block. Within the block statistics for all objects of a particular schema are gathered.

```
BEGIN
  DBMS_SCHEDULER.create_program (
    program_name => 'plsql_program',
    program_type => 'PLSQL_BLOCK',
    program_action => 'BEGIN
      DBMS_STATS.gather_schema_stats(''schemaname'');
    END;';
    enabled => TRUE,
    comments => 'Program to gather statistics using a PL/SQL block. ');
END;
/
```

To display the programs created in the database you can query the DBA\_SCHEDULER\_PROGRAMS data dictionary view.

```
SQL> SELECT owner, program_name, enabled
       FROM dba_scheduler_programs;
```

To create a program using the EM console. Select the Database Home Page -> Administration page -> Scheduler section -> Programs

### CREATING A SCHEDULE

Schedules are created using the CREATE\_SCHEDULE procedure. Given below is a creation of a schedule called *hourly\_schedule*, that repeats hourly. The schedule starts immediately.

```
BEGIN
  DBMS_SCHEDULER.create_schedule (
    schedule_name => 'hourly_schedule',
    start_date    => SYSTIMESTAMP,
    repeat_interval => 'freq=hourly; byminute=0',
    end_date      => NULL,
    comments      => 'Repeats hourly, on the hour, for ever. ');
END;
/
```

To display the programs created in the database you can query the DBA\_SCHEDULER\_SCHEDULES data dictionary view.

```
SQL> SELECT owner, schedule_name  
FROM dba_scheduler_schedules;
```

```
OWNER                SCHEDULE_NAME  
-----  
SYS                  HOURLY_SCHEDULE
```

### CREATING JOBS

Jobs can be created using the CREATE\_JOB procedure. They can either be made up of predefined programs and schedules or completely self contained.

There are many attributes that you can set for a job. They include:

The job\_type parameter indicates the type of task to be performed by the job. The possible values are:

- PLSQL\_BLOCK: An anonymous PL/SQL block.
- STORED\_PROCEDURE: A named PL/SQL, Java, or external procedure.
- EXECUTABLE: A command that can be executed from the operating system command line.

You can specify a repeat interval for a job. When scheduling repeat intervals for a job, you can specify either a calendaring expression or a datetime expression. The primary method of setting how often a job will repeat is by setting the repeat\_interval attribute using an Oracle calendaring. A calendaring expression has three main parts:

- The frequency (mandatory)
- The repeat interval (optional)
- Specifiers that provide detailed information about when the job should be run (optional).

Examples of repeating intervals are:

- `FREQ=HOURLY;INTERVAL=4` indicates a repeat interval of every four hours
- `FREQ=DAILY` indicates a repeat interval of every day, at the same time as the start date of the schedule
- `FREQ=MINUTELY;INTERVAL=15` indicates a repeat interval of every 15 minutes
- `FREQ=YEARLY;BYMONTH=MAR,JUN,SEP,DEC;BYMONTHDAY=15` indicates a repeat interval of every year on March 15, June 15, September 15, and December 15

If you need to specify more complex repeat intervals, such as every 15th day of the month, every fourth week on Monday, or 6:23 a.m. every Tuesday, then you will need to use the BY\* clauses to provide this additional information. For example:

- Every 15th day of the month: `FREQ=MONTHLY; BYMONTHDAY=15`
- Every fourth week on Monday: `FREQ=YEARLY; BYWEEKNO=4, 8, 12, 16, 20, 24, 28, 32, 36, 40, 44, 48, 52; BYDAY=MON`
- 6:23 a.m. every Tuesday: `FREQ=WEEKLY; BYDAY=TUE; BYHOUR=6; BYMINUTE=23`

An example of a PL/SQL expression could be  
`REPEAT_INTERVAL=>SYSDATE+1;`  
`REPEAT_INTERVAL=>SYSDATE + 15/ (24*60);`

This is an example of the creation of a job called `FULL_JOB_DEFINITION` that gathers statistics for a certain schema on an hourly basis.

```
BEGIN
    DBMS_SCHEDULER.create_job (
        job_name      => 'full_job_definition',
        job_type      => 'PLSQL_BLOCK',
        job_action    => 'BEGIN
            DBMS_STATS.gather_schema_stats('schemaname');
            END;';
        start_date    => SYSTIMESTAMP,
        repeat_interval => 'freq=hourly; byminute=0',
        end_date      => NULL,
        enabled       => TRUE,
        comments      => 'Self contained job');
END;
/
```

This is an example of the creation of a job called `program_sc_job` that references a program and a schedule.

```
BEGIN
    DBMS_SCHEDULER.create_job (
        job_name      => 'program_sc_job',
        program_name  => 'plsql_program',
        schedule_name => 'hourly_schedule',
        enabled       => TRUE,
```

```
    comments    => 'Job defined by an existing program and schedule.');"
END;
/
SELECT owner, job_name, enabled FROM dba_scheduler_jobs;
```

OWNER	JOB_NAME	ENABLED
-----	-----	-----
SYS	FULL_JOB_DEFINITION	TRUE
SYS	PROGRAM_SC_JOB	TRUE

### CREATING A WINDOW

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

Given here is an example of the creation of window called W1 that is associated with a predefined schedule. This window is assigned a low priority.

```
BEGIN
    DBMS_SCHEDULER.create_window (
        window_name    => 'W1',
        resource_plan   => NULL,
        schedule_name   => 'hourly_schedule',
        duration        => INTERVAL '60' MINUTE,
        window_priority => 'LOW',
        comments        => 'Window with a predefined schedule.');"
END;
/
```

This is an example of the creation of a window called W2 with an inline schedule.

```
BEGIN
DBMS_SCHEDULER.create_window (
    window_name    => 'W2',
    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;
/
```

To display the windows created and stored in the data dictionary you can query the DBA\_SCHEDULER\_WINDOWS data dictionary view.

```
SQL> 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
W1		TRUE	FALSE
W2		TRUE	FALSE

### **EXAM OBJECTIVE: REUSE SCHEDULER COMPONENTS FOR TASKS**

The scheduler you can reuse existing programs and schedules.

Once a program has been created and stored in the database, it can be reused while creating another object such as a job. Similarly a schedule can also be created as a separate entity and store in the database, for use while creating a job. You can create a job using existing programs and schedules thus giving a user the ability to use an existing object without having to repeatedly redefine it. Any parameter values specified when creating the job will override the default values that are specified in the program. This gives user the flexibility to customize the program to meet their needs.

Different jobs can also schedule the same program to run at different times by using different schedules.

This is an example of the creation of a job called program\_sc\_job that references a program and a schedule.

```
BEGIN
  DBMS_SCHEDULER.create_job (
    job_name      => 'program_sc_job',
    program_name => plsql_program',
    schedule_name => 'hourly_schedule',
    enabled       => TRUE,
    comments      => 'Job defined by an existing program and schedule.');
```

END;

/

**EXAM OBJECTIVE: VIEWING INFORMATION ABOUT JOB EXECUTIONS AND JOB INSTANCES**

Scheduler information can be retrieved from many data dictionary views. Given below is a complete list of views related to the scheduler. The \* can substituted by {ALL|USER|DBA}

<b>View</b>	<b>Description</b>
*_SCHEDULER_SCHEDULES	These views show all schedules.
*_SCHEDULER_PROGRAMS	These views show all programs.
*_SCHEDULER_PROGRAM_ARGUMENTS	These views show all arguments registered with all programs as well as the default values if they exist.
*_SCHEDULER_JOBS	These views show all jobs, enabled as well as disabled.
*_SCHEDULER_GLOBAL_ATTRIBUTE	These views show the current values of Scheduler attributes.
*_SCHEDULER_JOB_ARGUMENTS	These views show all arguments for all jobs, assigned and unassigned.
*_SCHEDULER_JOB_CLASSES	These views show all job classes.
*_SCHEDULER_WINDOWS	These views show all windows.
*_SCHEDULER_JOB_RUN_DETAILS	These views show all completed (failed or successful) job runs.
*_SCHEDULER_WINDOW_GROUPS	These views show all window groups.
*_SCHEDULER_WINDOWGROUP_MEMBERS	These views show the members of all window groups, one row for each group member.
*_SCHEDULER_RUNNING_JOBS	These views show state information on all jobs that are currently being run.