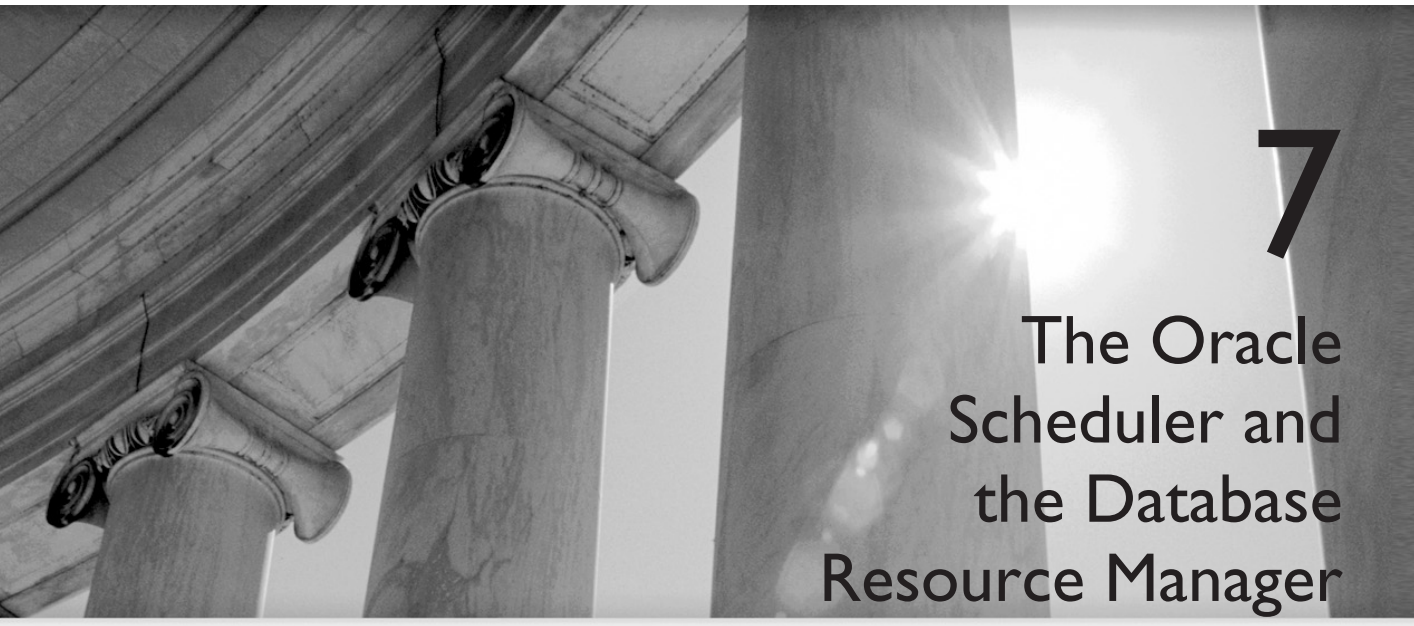

7 The Oracle Scheduler and the Database Resource Manager

CERTIFICATION OBJECTIVES

Oracle DBAs and developers frequently use the DBMS_JOB package to schedule database jobs. The package is a commonly used scheduling utility that has been around for a number of years. However, Oracle databases suffered from not having a real scheduling facility, especially when compared to the Microsoft SQL Server database. Oracle Database 10*g* provides you with the terrific new scheduling feature, aptly named the Scheduler. In this chapter, you'll learn first about the Scheduler framework and then how to manage this brand-new Oracle 10*g* feature.

In the latter part of this chapter, you'll learn about the Oracle Database 10*g* enhancements to the Database Resource Manager. These include using the Resource Manager to impose idle session limits, returning sessions automatically to their initial consumer groups after their top call is over, creating mapping between sessions and resource consumer groups, and using new methods of CPU allocation among resource plans and resource groups.

Let's start with an introduction to the new Scheduler and see how you can simplify management tasks by using it.

CERTIFICATION OBJECTIVE 7.01

Simplifying Management Tasks Using the Scheduler

The set of functions and procedures in the DBMS_SCHEDULER package form the backbone of the new Scheduler facility. The Scheduler helps you schedule jobs within the database or in the server that hosts the database. Using the Scheduler vastly simplifies a database administrator's tasks relating to the running of regular database jobs. The most important architectural feature of the Scheduler is its modular approach to managing tasks: it breaks down each task into location, time, and database object, thus simplifying management. The modular approach helps different users to reuse similar jobs, with minor modifications. The Scheduler works closely with the Database Resource Manager and utilizes the resource consumer group and resource plan concepts not only to schedule jobs according to a timetable, but also to allocate scarce server resources according to organizational needs and priorities. This focus on resource allocation marks the Scheduler as a much more sophisticated and powerful scheduling tool than the DBMS_JOBS package.

A severe limitation of the DBMS_JOBS package is that it can only schedule PL/SQL-based jobs. You can't use it to schedule operating system scripts or an executable. To run these non-database-type jobs, you must use a scheduling feature like crontab in UNIX or the AT facility in Windows servers. You may even have to use a third-party tool to get the job done. The Oracle Database 10*g* Scheduler feature offers you the capability to perform not only PL/SQL jobs, but also operating system shell scripts, Java programs, and native binary executables.

Basic Scheduler Components

The Scheduler consists of three basic components—jobs, programs, and schedules. You are familiar with the concept of an Oracle job from using the DBMS_JOB package in the previous versions. However, the other two components, programs and schedules, are new in the Scheduler. The concepts of a program and a schedule lead to a modular approach to the management of tasks. For example, several users can perform similar tasks, with minor changes in the time and resource usage, by simply using similar schedules and programs.

Let's look at the three main components of the Scheduler in the following sections.

Jobs

A *job* is a task that you schedule to run one or more times. A job contains information about what it should execute and time of execution. A Scheduler job can execute a PL/SQL block of code, a native binary executable, a Java application, or a shell script. You can create a new job by either specifying all the details, such as the code the job should execute and the time and frequency of the execution, or you can simply use saved programs and schedules, to facilitate the creation of a job.

When you create a new job, Oracle adds an entry to the job table. There is one job table for each database. When you execute a job, the database makes an entry in the job log.

Schedules

A *schedule* is a specification of when and how frequently the database should execute a job. The important thing to note here is that you can use the same schedule for several jobs. Schedules are like any other database entities, and you store them just as you do the other objects. Users can share schedules.

on the job

The schedule and program components of the Scheduler are purely optional.

Programs

A *program* contains information about the code, script, or executable that's to be executed as part of a job. The information contains a program name, the program action (PL/SQL code or a UNIX shell script, for example), and other similar items. As with schedules, several jobs can share the same program.

You can combine the three main components of the Scheduler—jobs, schedules, and programs, to create and run scheduled jobs in your database, as you'll see in later sections of this chapter.

Advanced Scheduler Components

In addition to jobs, schedules, and programs, the Scheduler also uses several advanced concepts: job classes, windows, and resource groups. These advanced features set apart the Scheduler from its predecessor, the `DBMS_JOBS` package, in the matter of scheduling database jobs. These advanced concepts enable the prioritizing of jobs in the database and the allocation of resources in accordance with your priorities. Let's look at these components in the following sections.

Job Classes

A *job class* groups similar jobs into one large entity. The jobs that are part of a job class share common characteristics like resource requirements. By helping you to classify several jobs into related groups on the lines of functionality or shared attributes, job classes facilitate the calibration of limited resources among different groups of jobs in your database.

Job classes group jobs with common characteristics. The main purpose of using a job class is to manage resource allocation among jobs. A job can belong to only one job class.

After grouping jobs into classes on the functionality (marketing, administration, finance, and so on), you can use the concept of a job class to do the following:

- Assign the job priority level for an individual job, with the priority values ranging from 1 to 5. A higher-priority job always starts before a lower-priority job.
- Specify common attributes such as the frequency of log purging.

Windows

A *window* represents an interval of time during which you can schedule jobs. The purpose behind using a window is to change resource allocation during a time period. For example, a "maintenance window" may last from Friday night to Sunday evening. Each window is associated with a specific resource plan (created with the help of the Database Resource Manager). By using a window in which to schedule a job, you ensure that the job will only run during a certain time interval, when a certain resource plan associated with that window is active. The concept of resource plan–based windows enables you to ensure optimal use of your scarce resources.

Windows can overlap, and when they do, Oracle chooses the window with the higher priority. The window with the higher priority opens, and the other window will close.

Window Groups

Window groups are simply a collection of similar windows. Once you create a window group, you can then specify that a job will run during a window group. For example, you can create a window for your weekends and a window for your holidays. You can then group both these windows into a single maintenance window, if you wish. The key idea here is that a window group comprises windows with similar characteristics. You have the choice of specifying a window or window group as the schedule for a job.

Resource Allocation Among Jobs

The Scheduler and the Database Resource Manager are tightly integrated in Oracle Database 10*g*, thus making it easy to prioritize your jobs and assign the right level of resources to jobs. You can use the Resource Manager feature to control resource allocation in your database. The important thing to note here is that the Scheduler doesn't allocate resources at the individual job level—it does so at the job class level. Oracle assigns each job class to a specific resource consumer group. If you don't specify a resource consumer group for any job class, that job class will by default be assigned to the default consumer group.

The resource consumer group that a job class belongs to will determine the allocation of resources to that job class. Thus, you can use the Database Resource Manager to prioritize Scheduler jobs.

Scheduler Architecture

The Scheduler architecture consists primarily of the job table, job coordinator, and the job workers (or slaves). Let's look at each of these components in the following sections.

The Job Table

The *job table* contains information about the jobs, such as the job name, program name, and job owner. You can examine the job table by using the `DBA_SCHEDULER_JOBS` view.

The Job Coordinator

When you create a new job or execute a job, a background process (cjqNNN) automatically wakes up and starts coordinating the running of the job. The Scheduler sleeps when there aren't any jobs to execute. The *job coordinator* regularly looks in the job table to find out what jobs to execute. The job coordinator creates and manages the job worker processes that execute the job.

The Job Workers

When the job coordinator instructs a *job worker* to execute a job, the worker process starts a database session and starts a transaction. It then starts executing the job, and once the job completes, it commits and ends the transaction and terminates the database session.

It is the job of the job workers to

- Update the job table, to indicate that the job has been completed
- Update the job log table
- Update the run count

Scheduler Privileges

Jobs, schedules, and programs are like any other database object when it comes to ownership and privileges. Oracle creates all jobs, programs, and schedules in the current user's schema. However, Oracle creates all the advanced Scheduler components, like job classes, windows, and window groups, at the database level, not the schema level, and their owner is the SYS schema.

The MANAGE SCHEDULER system privilege lets you do the following:

- Create, drop, and alter job classes, windows, and window groups. (You still can't create a job in a job class you create, unless you have a separate EXECUTE privilege on that job class.)
- Stop any job.
- Start and stop windows prematurely.

By default, all Scheduler objects are in uppercase, unless you wrap the lowercase names in double quotes, as in "test_job".

You must have the CREATE JOB privilege to create the basic job, schedule, and program components. In order to use the advanced Scheduler components like windows, window groups, and job classes, you need the MANAGE SCHEDULER system privilege.

Once you create a job or some other Scheduler component, it will be a part of your own schema. You can assign other users the right to use one of your components like a job, schedule, or program by giving them EXECUTE privileges on that component. For example, you can grant the EXECUTE privileges for a certain program or for a certain job class. Note that

- The EXECUTE ANY PROGRAM privilege lets a user execute any program under any schema.
- The EXECUTE ANY CLASS privilege lets you assign a job to any job class.

In order for a user to modify a Scheduler component, you must grant the user the following privilege (you use the appropriate component):

```
SQL> grant alter on <job, program or schedule> to scott;
```

When you create a job, as you are aware, you can use programs, schedules, job classes, and windows. Once you have the CREATE JOB privilege, you can start creating your job, with the following restrictions:

- To create the job itself, you must have the CREATE JOB privilege.
- If you specify a program that you happen to own, you don't need any privileges to specify the program.
- If you specify a program owned by a different user, you must have the EXECUTE privilege on that program, or the EXECUTE ANY PROGRAM system privilege.

- You can specify a schedule, whether owned by you or a different user, without any privileges.
- You don't need any special privileges to specify a window or windows group. However, since all of these objects are in the SYS schema, you must fully qualify these components when you use them in your `CREATE JOB` or any other statement.
- If you wish to assign your new job to a job class, you must either have the EXECUTE privilege on that class, or the more powerful EXECUTE ANY CLASS system privilege.

on the job

The SCHEDULER_ADMIN role contains all the Scheduler system privileges, with the `WITH ADMIN OPTION` clause. The DBA role contains the SCHEDULER_ADMIN role.

CERTIFICATION OBJECTIVE 7.02

Managing the Basic Scheduler Components

The basic Scheduler components—jobs, programs, and schedules—have several common manageability features. For example, you follow a similar procedure to create, alter, and drop all three components. Similarly, you can use the `SET_ATTRIBUTES` procedure of the `DBMS_SCHEDULER` package to change the attributes of all three components. Let's start with managing Scheduler jobs in the next section.

on the job

All Scheduler objects are of the form [schema].name. All Scheduler names are in uppercase, unless you surround them with double quotes.

Managing Jobs

Creating and managing jobs, of course, is at the heart of the Scheduler facility. As you'll learn, you can create and run jobs independently, or you can create a job using schedules and programs. Use of saved programs and schedules enables you to eliminate the redefinition of the program or schedule each time you create a new job. This reusability of the Scheduler components is a highly useful feature, since users can customize schedules and programs to meet their needs. Let's start with the creation of Scheduler jobs.

exam Watch

The parameter values you specify when you create a job will override the default values for the saved programs and schedules.

Creating Jobs

You create a job using the CREATE_JOB procedure of the DBMS_SCHEDULER package. You can specify all the job execution details directly using the CREATE_JOB procedure. Programs and schedules, as you've seen earlier, are saved database objects that contain information pertaining to jobs, like the job type and the start and end times. Later, you'll see how to use programs and schedules to make the creation of jobs easy to manage. Here's a simple example that shows how to create a basic Scheduler job. Note that there aren't any programs or schedules in this job—this is the most straightforward way to specify a job, with all pertinent information specified in the job creation statement itself. More sophisticated examples later on show you how to embed most of the repetitive information within saved schedules and windows.

```
SQL> begin
  2      dbms_scheduler.create_job(
  3      job_name           =>  'test_job1',
  4      job_type           =>  'PLSQL_BLOCK',
  5      job_action         =>  'DELETE FROM persons WHERE sysdate=sysdate-1',
  6      start_date         =>  '28-JUNE-04 07.00.00 PM Australia/Sydney',
  7      repeat_interval    =>  'FREQ=DAILY;INTERVAL=2',
  8      end_date           =>  '20-NOV-04 07.00.00 PM Australia/Sydney',
  9      comments           =>  'TEST JOB');
 10*   end;
SQL> PL/SQL procedure successfully completed.
SQL>
```

on the Job

Your new job, test_job1, isn't ready to be used yet, since by default, all new jobs are disabled when they are created. You need to enable a job in order to use it.

exam Watch

You'll be the owner of a job if you create it in your own schema. However, if you create it in another schema, that schema user will be owner of the job. Thus, the fact that you create a job doesn't mean that you are necessarily the owner of that job.

Note how similar the preceding job creation statement is to using the traditional DBMS_JOB package to create a new job. Incidentally, you can still use the DBMS_JOB package in Oracle Database 10*g*, since it's left there for backward-compatibility purposes. Let's look at the various components of our CREATE_JOB procedure.

job_name This parameter provides a way to specify a name for your job.

job_type This parameter tells the Scheduler what type of job—a PL/SQL block, a stored procedure, a shell script, or a Java program—that the job must execute.

job_action This parameter specifies the exact procedure, command, or script that the job will execute.

start_date and end_date These parameters specify the date that a new job should start and end. (Many jobs may not have an `end_date` parameter, since they are ongoing jobs.)

repeat_interval This attribute tells the Scheduler how often it should execute the job. In our example, the repeat interval is `'FREQ=DAILY;INTERVAL=2'`, which means that you run the job every other day. You can specify a repeat interval in one of two ways:

- Use a PL/SQL date/time expression.
- Use a database calendaring expression.

If you omit the `repeat_interval` attribute, a job will run only once.

Note that a big feature of the Scheduler tool is its ability to run operating system scripts and executables in addition to PL/SQL code. Here's an example that shows how you can schedule a job on your server. The job runs the check_freespace.ksh UNIX script every half an hour.

```
begin
dbms_scheduler.create_job(
job_name        => 'CHECK_FREE_SPC',
job_type        => 'EXECUTABLE',
job_action      => '/u01/app/oracle/admin/dba/scripts/check_freespace.sh',
repeat_interval => 'FREQ=MINUTELY; INTERVAL=30',
enabled         => true,
comments        => 'Check free space in tablespaces');
end;
/
```

The script will run every 30 minutes (`'FREQ=MINUTELY; INTERVAL=30'`), and the `job_type` parameter shows that it's an executable program. The `job_action` parameter tells the Scheduler which OS script to execute.

Let's briefly look at how you can set your repeat interval using calendaring expressions and PL/SQL date/time expressions.

- **Using a calendaring expression** *Calendaring expressions* generate the next date of execution for a job. A calendaring expression is a straightforward, English-like expression consisting of the following three components:
  - **Frequency** A mandatory component of a calendaring expression, represented by the keyword `FREQ`. Possible values are `YEARLY`, `MONTHLY`, `WEEKLY`, `DAILY`, `HOURLY`, `MINUTELY`, and `SECONDLY`.
  - **Repeat interval** Refers to how often the database must repeat the job. The keyword `INTERVAL` shows the repeat interval.
  - **Specifiers** Provides detailed information about when a job should be run. Possible values are `BYMONTH`, `BYWEEKNO`, `BYYEARDAY`, `BYMONTHDAY`, `BYDAY`, `BYHOUR`, `BYMINUTE`, and `BYSECOND`.

  Of these three components, specifiers are optional, but the repeat interval and frequency are mandatory. You'll need specifiers only when you need to specify complex repeat intervals. Here are some examples of using the three components to provide values for the `repeat_interval` attribute of the `CREATE_JOB` procedure:

```
FREQ=DAILY; INTERVAL=10 executes a job every 10 days
FREQ=HOURLY; INTERVAL=2 executes a job every other hour
FREQ=WEEKLY; BYDAY=FRI  executes a job every Friday.
FREQ=WEEKLY; INTERVAL=2; BYDAY=FRI executes a job every other Friday.
FREQ=MONTHLY; BYMONTHDAY=1 executes a job on the last day of the month
FREQ=YEARLY; BYMONTH=DEC; BYMONTHDAY=31 executes a job on the 31st of December.
FREQ=MONTHLY; BYDAY=2FRI executes a job every second Friday of the month
```

  In the preceding set of examples, the `BY*` clause is the optional specifier component, which provides additional information that the frequent and repeat interval components can't provide.

- **Using a PL/SQL expression** You can also create more complex repeat intervals than what the calendaring expressions enable you to do by using PL/SQL expressions, with the proviso that all such expressions must evaluate to a date or a timestamp datatype. When you use a date/time expression for specifying the repeat interval, you end up with a date/time datatype as the value of the interval. Look at the following example:

```
repeat_interval      =>  'SYSTIMESTAMP + INTERVAL '10' MINUTE'
```

  The `repeat_interval` specification in the preceding example shows the use of a PL/SQL expression. The expression states that Oracle will execute the job every ten minutes. Each time a job executes, Oracle evaluates the `repeat_interval` clause to compute the next scheduled execution time of the job—in this case, ten minutes into the future.

Administering Jobs

You can use the DBMS_SCHEDULER package to perform all the administrative tasks associated with jobs. Let's look at some of the main job-related administrative tasks in the following sections.

Enabling and Disabling Jobs All jobs are disabled by default when you create them. You must explicitly enable them in order to activate and schedule them. You can enable a job by issuing the following command:

```
SQL> exec dbms_scheduler.enable ('TEST_JOB1');
PL/SQL procedure successfully completed.
```

You can disable a job by issuing the following command:

```
SQL> exec dbms_scheduler.disable ('TEST_JOB1');
PL/SQL procedure successfully completed.
```

Dropping a Job You can drop a job by using the DROP_JOB procedure, as shown here:

```
begin
dbms_scheduler.drop_job (
  job_name    =>  'test_job1');
end;
/
```

Running and Stopping a Job You can run a job manually (at other than the regularly scheduled times) by using the RUN_JOB procedure, as shown here:

```
SQL> exec dbms_scheduler.run_job('TEST_JOB1');
```

You may want to run a job manually to test it before scheduling on a regular basis, or to correct some errors in a job that already ran. When you run a job manually, it runs in your own session, and the job coordinator and the job slave don't enter the picture.

You can stop a job immediately by using the STOP_JOB procedure, as shown here:

```
SQL> exec dbms_scheduler.stop_job('TEST_JOB1');
```

exam Watch

In both the `STOP_JOB` and `RUN_JOB` procedures, there is a `FORCE` argument, which is set to `false` by default. The `FORCE` attribute determines whether the job can be stopped or dropped if it is currently running. By setting `FORCE=TRUE`, you can stop or drop a job immediately by using the appropriate procedure. You must have the MANAGE SCHEDULER system privilege to use the `FORCE` setting.

EXERCISE 7-1

Creating a Job to Perform an Online Backup

Create a simple job, which will execute a script at a scheduled time. Here's a sample job creation code:

```
begin
    dbms_scheduler.create_job (
    job_name               =>   'EXECUTABLE',
    job_action             =>
    '/U01/APP/ORACLE/DBA/SCRIPT/PROD1_ONLINE_BKP.sh',
    start_date        =>  '05-JUL-04 11.00.00 PM',
    repeat_interval   =>  'FREQ=DAILY;INTERVAL=2', /* every other day */
    comments         =>  Alternate day online backup');
end;
        /
```

Managing Programs

A program contains metadata, including the name and type of the program, and what a job will execute, whether it is a PL/SQL procedure, a script, or an executable. When you use programs, you can alter the job task, or what the job will actually do, without directly modifying the job definition itself. Programs enable you to control the run-time behavior of the job task. Different jobs can pick the same program from a library of programs, thus helping you avoid having to specify all the details of a program in your job creation statement.

Let's start by looking at how you can create programs. Later you'll learn about program-related administrative tasks.

Creating a Program

To create a new Scheduler program, use the CREATE_PROGRAM procedure of the DBMS_SCHEDULER package, as shown here:

```
SQL> begin
  2  dbms_scheduler.create_program(
  3  program_name => 'TEST_PROGRAM',
  4  program_action => 'SCOTT.UPDATE_SCHEMA_STATS',
  5  program_type => 'STORED_PROCEDURE',
  6  enabled => TRUE);
  7* end;
SQL> /
PL/SQL procedure successfully completed.
SQL>
```

By default, Oracle creates a program in the creator's schema. If you want to create the program in a different user's schema, you must qualify the program name with the schema name.

The program component has all the job details. Once you create a program in the manner shown here, you can simplify your job creation statement for the job TEST_JOB1 by replacing the job_type and job_action attributes with the name of the program (TEST_PROGRAM) that already contains the specification of these attributes. The program_type and program_action will provide the information that the three job attributes provided in your first job creation statement. You can begin to see why this type of a modular approach is beneficial—different jobs can use the same program, thus simplifying the specification and creation of jobs.

Let's re-create our TEST_JOB1 job, using the program component this time. Here's the much simpler job creation statement:

```
SQL> begin
  2  dbms_scheduler.create_job(
  3  job_name => 'TEST_JOB1',
  4  program_name => 'TEST_PROGRAM',
  5  repeat_interval=> 'FREQ=DAILY;BYHOUR=12',
  6  ENABLED => TRUE;
ENABLED => TRUE);
  7* end;
SQL> /
PL/SQL procedure successfully completed.
SQL>
```

In the preceding example, the use of a program (TEST_PROGRAM) lets you avoid specifying the job_type and job_action parameters in the CREATE_JOB

statement. You thus have a choice between specifying all job attributes directly in the CREATE_JOB statement itself, or using a stored program to supply some of the job attributes.

Administering Programs

You can enable, disable, drop, and modify Scheduler programs using various procedures from the DBMS_SCHEDULER package.

Enabling and Disabling Programs You can enable a program in the following manner, using the ENABLE procedure:

```
SQL> exec dbms_scheduler.enable('TEST_PROGRAM');
PL/SQL procedure successfully completed.
```

You can disable a program in the following manner, using the DISABLE procedure:

```
SQL> exec dbms_scheduler.disable('TEST_PROGRAM');
PL/SQL procedure successfully completed.
```

Dropping a Program You can drop a saved program by using the DROP_PROGRAM procedure, as shown here:

```
SQL> exec dbms_scheduler.drop_program('TEST_PROGRAM');
PL/SQL procedure successfully completed.
SQL>
```

Managing Schedules

Let's say you have a number of jobs, all of which use similar execution time schedules. In cases like this, the Scheduler makes it easy for you to use a common schedule. If you need to modify some aspect of the schedule, all the jobs that use the schedule can automatically inherit those changes. Let's start by looking at how you create a schedule.

Creating a Schedule

You use the CREATE_SCHEDULE procedure of the DBMS_SCHEDULER package to create a schedule, as shown here:

```
SQL> begin
  2  dbms_scheduler.create_schedule(
  3  schedule_name =>  'TEST_SCHEDULE',
  4  start_date => SYSTIMESTAMP,
```

```
  5  end_date  =>  SYSTIMESTAMP + 30,
  6  repeat_interval  => 'FREQ=HOURLY;INTERVAL= 12',
  7  comments => 'Every 12 hours');
  8* end;
SQL> /
PL/SQL procedure successfully completed
SQL>
```

The TEST_SCHEDULE schedule states that a job with this schedule will be executed immediately and then be reexecuted every 12 hours, for a period of 30 days. Note the following things about the creation of the TEST_SCHEDULE schedule:

- The CREATE_SCHEDULE procedure has three important parameters (there is also an optional comment parameter).
- The start and end times are always precise to a second.
- You specify the start and end times using the TIMESTAMP WITH TIME ZONE datatype. The Scheduler also supports all NLS_TIMESTAMP_TZ_FORMAT settings.
- You *must* use a calendering expression to create the repeat interval.

Once you create the schedule TEST_SCHEDULE, you can simplify the job creation process even further. Now you can use both TEST_PROGRAM and TEST_SCHEDULE to simplify job creation, as shown here:

```
SQL> begin
  2  dbms_scheduler.create_job(
  3  job_name       =>  'TEST_JOB02',
  4  program_name   =>  'TEST_PROGRAM',
  5  schedule_name  => 'TEST_SCHEDULE');
  6  end;
  7  /
PL/SQL procedure successfully completed.
SQL>
```

As you can see, TEST_JOB02 is really easy to create, since you are using both a saved program as well as a saved schedule. The use of the saved schedule means that you don't have to specify the start_date, end_date, and the repeat_interval attributes while creating TEST_JOB02, since TEST_SCHEDULE already has all that information.

Administering Schedules

A schedule states when and how often a job should be run. You can save a schedule as a database object and share it with other users. Let's look at some of the basic administrative tasks concerning schedules.

Creating a Schedule You create a schedule using the `CREATE_SCHEDULE` procedure. You can create a schedule in your own or some other schema. You've learned how to use the `CREATE_SCHEDULE` procedure in the previous section. The most important attributes of a schedule are the start and end dates and the frequency of execution, which is set by the `repeat_interval` attribute. You can alter the various attributes of a schedule by using the `SET_ATTRIBUTE` procedure of the `DBMS_SCHEDULER` package. You may alter all attributes except `schedule_name`.

exam Watch

When you create a schedule, Oracle provides access to PUBLIC. Thus, all users can use your schedule, without any explicit grant of privileges to do so.

EXERCISE 7-2

Creating a Schedule

Create a simple schedule on the lines of the following example. In the example, the `repeat_interval` attribute specifies that a job will execute every five minutes.

```
SQL> begin
       dbms_scheduler.create_schedule (
       scheduler_name      => 'TEST_SCHEDULE',
       start_date          => SYSTIMESTAMP,
       repeat_interval     => 'FREQ=MINUTELY;INTERVAL=5',
       comments            => 'A test schedule.');
     end;
         /
```

Dropping a Schedule You can drop a schedule with the `DROP_SCHEDULE` procedure, but there are some interesting things to note in this regard. For example,

if you have any jobs or windows that use the schedule you want to drop, your procedure will fail, as shown here:

```
SQL> begin
  2  dbms_scheduler.drop_schedule(schedule_name => 'TEST_SCHEDULE');
  3  end;
  4  /
begin
ERROR at line 1:
ORA-27479: Cannot drop "SAM.TEST_SCHEDULE" because other objects depend on it
ORA-06512: at "SYS.DBMS_ISCHED", line 615
ORA-06512: at "SYS.DBMS_SCHEDULER", line 1070
ORA-06512: at line 2
SQL>
```

In our example, `TEST_JOB03` uses the schedule `TEST_SCHEDULE`, and therefore, your attempt to drop the schedule results in an error. You can override this default behavior by using the `FORCE` attribute and setting it to `true`, as shown here:

```
SQL> begin
  2  dbms_scheduler.drop_schedule(schedule_name  =>  'TEST_SCHEDULE',
  3  force => TRUE);
  4  end;
  5  /
PL/SQL procedure successfully completed.
SQL>
```

When you drop a schedule by using the `FORCE=TRUE` attribute, you'll drop the schedule, even if there are jobs and windows that use the schedule. The Scheduler first disables the dependent jobs/windows before dropping the schedule itself.

CERTIFICATION OBJECTIVE 7.03

Managing Advanced Scheduler Components

In the previous section, you learned how to manage the basic Scheduler components—jobs, programs, and schedules. In this section, let's look at how to manage the advanced Scheduler components—job classes and windows (and window groups). You'll also learn how the Scheduler makes good use of the Database Resource Manager features, such as resource consumer groups and resource plans, to efficiently allocate scarce OS and database resources. Too often, heavy batch jobs run past their window of opportunity

and spill over into the daytime, when OLTP transactions demand the lion's share of the resources. Prioritizing jobs to ensure that they are guaranteed adequate resources to perform along accepted lines is an essential requirement in production databases. The Scheduler uses the concepts of job classes and windows to prioritize jobs.

Job Classes

You can create and schedule a job all by itself, or you can manage it as part of a job class. Although both types of jobs will perform the same actions, you can't really allocate resources when you run a job as an entity by itself. Job classes group all jobs with similar characteristics and common resource requirements together. Using job classes helps you prioritize jobs by allocating resources differently among the various jobs.

The Scheduler associates each job class with a *resource consumer group*, which lets the Scheduler group jobs according to common resource requirements. Depending on the resource consumer group a job class belongs to, the Scheduler will determine the appropriate resource allocation. This ability to associate job classes with resource consumer groups provides an efficient way to prioritize jobs in the database.

You use the `CREATE_JOB_CLASS` procedure of the `DBMS_SCHEDULER` to create a job class. Unlike a job, a job class can't be created in a user's schema—no matter who creates it, all job classes are created in the SYS schema. You can create a job class only if you have the MANAGE SCHEDULER privilege.

exam Watch

All jobs must belong to a job class. There is a default job class, `DEFAULT_JOB_CLASS`, to which all jobs will belong if you don't explicitly assign them to a job class. Similarly, there is a default resource consumer group, the `DEFAULT_CONSUMER_GROUP`, to which a job class will map if you don't expressly assign that job class to a user-created resource consumer group.

Creating a Job Class

You create a job class by using the `CREATE_JOB_CLASS` procedure of the `DBMS_SCHEDULER` package, as shown here:

```
begin
dbms_scheduler.create_job_class (
   job_class_name             =>  'admin_jobs',
```

```
   resource_consumer_group    =>  'admin_group',
   logging_level           => dbms_scheduler.logging_off
   logging_history         =>  30);
);
end;
```

Let's look at the parameters of the CREATE_JOB_CLASS procedure in the preceding example:

- Job_class_name is the name of the job class.
- Resource_consumer_group tells us that all the jobs that are members of our new class will have the admin_group as their resource consumer group.
- The logging_level parameter, which is set to LOGGING_OFF, can take the following three values:
  - **Logging_off** There is no logging of any kind for the jobs.
  - **Logging_runs** For each run of a job in the job class, there will be a detailed entry in the job log.
  - **Logging_full** For each run of a job in the job class, there will be a detailed entry in the job log. In addition, the Scheduler logs all operations performed on the other jobs in the job class. These operations include the creation, dropping, altering, enabling, or disabling of the other jobs.

The default logging level is to log all job runs (LOGGING_RUNS option). The LOGGING_FULL option gives you the most information about jobs.

- Logging_history specifies the number of days (30 in this case) that the database will retain the logs before purging them. Oracle will automatically create a daily job called the PURGE_LOG, which cleans the log entries. The following query shows that this job exists:

```
SQL> select job_name
  2  from dba_Scheduler_jobs;
JOB_NAME
------------------------------
GATHER_STATS_JOB
PURGE_LOG
…
SQL>
```

If you want to clear the logs manually, you can do so by using the DBMS_SCHEDULER.PURGE_LOG procedure.

When you create the ADMIN_JOBS job class, Oracle will automatically store it in the SYS schema, and the same is true of any other job class that you might create in the database. Any user in the database can create a job in the ADMIN_JOBS job class, provided they have the EXECUTE privilege on that specific job class, or they have the EXECUTE ANY JOB CLASS privilege.

exam Watch

The PURGE_LOG procedure of the DBMS_SCHEDULER package takes two important parameters—log_history and which_log. You use the log_history parameter to specify the number of days to keep logs before the Scheduler purges them. The which_log parameter enables you to specify whether you want to purge job or window logs. For example, to purge all job logs more than seven days old, you use the statement:

```
exec dbms_scheduler.purge_log(log_history=7,
which_log =>'JOB_LOG')
```

Altering a Job Class

After you create a job class, you can alter all of its attributes except the job_name attribute. You can use the ALTER_ATTRIBUTES procedure to change a job class's attributes, as shown in the following code. You can change the START_DATE, END_DATE, and other logging-related attributes as well. In the example, the attribute you are changing is START_DATE. The VALUE clause holds the new value of the START_DATE attribute.

```
SQL> begin
  2  dbms_scheduler.set_attribute(
  3  name => 'ADMIN_JOBS',
  4  attribute => 'START_DATE',
  5  value => '01-JAN-2005 9:00:00 PM US/Pacific');
  6* end;
SQL>
```

on the Job

You can't change any attributes of the default job class, which is named, well, the DEFAULT_JOB_CLASS.

Dropping a Job Class

You can drop a job class by executing the DROP_JOB_CLASS procedure, as shown here:

```
begin
dbms_scheduler.drop_job_class('TEST_CLASS');
end;
```

exam Watch

If you want to drop a job class with jobs in it, you must specify the FORCE=TRUE option in your DROP_JOB_CLASS procedure. When you do this, the jobs in the dropped job class are disabled and moved to the default job class in your database. If the job is already running when you drop its job class, the job will run to completion anyway.

Review of the Database Resource Manager

Much of the Scheduler's power comes from its ability to interact with the Database Resource Manager to allocate resources among the various scheduled jobs. In order to fully comprehend this interaction, it pays to quickly review the concepts of the Database Resource Manager. Let's take a brief detour to do that in this section.

Using the Database Resource Manager, you can allocate percentages of your server CPU to contending applications and users. You can also limit operations that you think are going to run too long, as well as control the degree of parallelism used by various jobs. You may also terminate long-running sessions and guarantee minimum processing resources to certain users, regardless of the system load.

You can fully use the resource controlling power of the Database Resource Manager by creating the basic components and modifying them over time. The four main components are the resource consumer group, resource plan, resource allocation method, and resource plan directive.

Resource Consumer Group A group of users or sessions that have identical processing and resource requirements is known as a resource consumer group. You can manually assign users to resource consumer groups or let the database assign the user sessions to specific groups based on the value of various user session attributes. You may dynamically reassign a session to a different consumer group.

Resource Plan How you allocate resources among the various resource consumer groups is your resource plan. For example, a resource plan may allocate 50 percent of the CPU resources to the Finance Group, 30 percent to the Sales Group, and 20 percent to the Administration Group of users.

Resource Allocation Method You can choose the method, or policy, of allocation of resources among the various resource consumer groups.

Resource Plan Directive You can assign a resource consumer group to a resource plan, thus deciding what resources each resource consumer group gets. Plan directives also enable you to specify the parameters for each resource allocation method. For example, you may limit the degree of parallelism of all users in the Administrative Group to 2, instead of using the default value, which is an unlimited number of parallel sessions.

With this brief review of the Database Resource Manager behind us, let's move on to see how you can use the concept of a time window to activate different resource plans at different times.

Working with Scheduler Windows

Although the Scheduler works in tandem with the Database Resource Manager to allocate resources in an optimal way to all the jobs scheduled for running, you will want to allocate different amounts of resources at different times. This is where the concept of the Scheduler window serves as a handy device. Windows enable the automatic changing of resource plans based on a schedule.

The basic purpose of a window is to switch the active resource plan during a certain time frame. All jobs that run during this window will be controlled by the resource plan that's in effect during that window. Without windows, you'll have to manually switch the resource manager plans.

A window is an interval with a specific begin and end time, for example, "from 12 midnight to 6:00 A.M." However, a window is not merely a chronological device like a schedule, specifying when a job will run; every window is associated with a resource plan. When you create a window, you specify a resource plan as a parameter. This ability to activate different resource plans at different times is what makes a window a special scheduling device that enables you to set priorities.

exam Watch

All windows, like job classes, are created in the SYS schema, no matter which user creates them. To create a window, you must have the MANAGE SCHEDULER system privilege.

A Scheduler window consists of the following three major attributes:

- *Schedules*, which are the times when the window is in effect
- *Duration*, which determines the length of time a window stays open
- *Resource plan*, which determines the resource priorities among the job classes

Note that schedules and duration are part of all Scheduler schedules; the resource plan sets a Scheduler window apart from a simple schedule. Each time a window is open, a specific active resource plan is associated with it. Thus, the same job will be allocated different resources if it runs under different windows.

You can specify what resources you want to allocate to various job classes during a certain time period (say midnight to 6:00 A.M.) by associating a resource plan with the window you create for this period. When the window opens, the database automatically switches to the associated resource plan, which becomes the active resource plan. The systemwide resource plan associated with the window will control the resource allocation for all jobs and sessions that are scheduled to run within this window.When the window finishes, there will be another switch to the orginal resource plan that was in effect, provided no other window is in effect at that time.

If a window is open and the Resource Manager is not turned on, the window won't be able to switch the resource plans. In this case, the window will continue to open as scheduled and run the jobs that have been assigned for that window. Thus, the window in this case serves only as a basic schedule component.

You can see which window is currently active and the resource plan associated with that window by using the following query:

```
SQL select window_name, resource_plan from dba_scheduler_windows
    where active='TRUE';
   WINDOW_NAME                    RESOURCE_PLAN
  ------------------------------ --------------------------
  TEST_WINDOW                    TEST_RESOURCEPLAN
SQL>
```

The V$RESOURCE_PLAN provides information on currently active resource plans in your database. If you want to view the active resource plan when there is no window open, you can use the following query:

```
SQL> select * from v$resource_plan;
```

Creating a Window

You create a window by using the CREATE_WINDOW procedure. Let's look at two examples using this procedure, one with an inline specification of the start and end times and the repeat interval, and the other where you use a saved schedule instead to provide these three scheduling attributes. In the first example, the window creation statement specifies the schedule for the window.

```
begin
dbms_scheduler.create_window (
   window_name       =>  'TEST_WINDOW',
   start_date        =>  '01-JAN-05 12:00:00AM',
   repeat_interval   =>  'FREQ=DAILY',
   resource_plan     =>  'TEST_RESOURCEPLAN',
   duration          =>  interval '60' minute,
   end_date          =>  '31-DEC-05 12:00:00AM',
   window_priority   =>  'HIGH',
   comments          =>  'Test Window');
end;
/
```

Let's look at the individual attributes of a window in TEST_WINDOW:

- Resource_plan tells us that while this window is open, resource allocation to all the jobs that run in this window will be guided by the resource plan directives in the resource plan TEST_RESOURCEPLAN.
- Window_priority is set to HIGH. The default priority level is LOW. These are the only two values possible. If two windows overlap, the window with the high priority level has precedence. Since only one window can be open at a given time, when they overlap, the high-priority window will open and the lower-priority window doesn't open.
- Start_date indicates that your window first becomes active at 12:00 A.M. on January 1, 2005. You can also say that the window will *open* at this time.
- The duration attribute shows that the window will remain open for a period of 60 minutes, after which it will close.
- Repeat_interval tells us the next time the window will open again. In our case, it is 12:00 A.M. on January 2, 2005.
- End_date tells us that this window will open for the last time on December 31, 2005, after which it will be disabled and closed.

on the Job

Since the Scheduler doesn't check to make sure that there are prior windows for any given schedule, you have the possibility of overlapping windows. Note that Oracle doesn't recommend overlapping windows.

In the second example, shown here, you see how you can create a window using a saved schedule. Obviously, it is much simpler to create a window this way.

```
begin
dbms_scheduler.create_window (
   window_name          =>  'TEST_WINDOW',
   schedule_name        =>  'TEST_SCHEDULE',
   resource_plan        =>  'TEST_RESOURCEPLAN',
   duration             =>  interval '160' minute,
   comments             =>  'Test Window');
end;
/
```

In the preceding `CREATE_WINDOW` procedure, the use of the schedule `TEST_SCHEDULE` lets you avoid specifying the `start_date`, `end_date`, and `repeat_interval` parameters.

on the Job

A window is automatically enabled upon its creation.

Once you create a window, you must associate it with a job or job class, so the jobs can take advantage of the automatic switching of the active resource plans.

Managing Windows

You can open, close, alter, enable/disable, or drop a window using the appropriate procedure of the `DBMS_SCHEDULER` package. You need the MANAGE SCHEDULER privilege to perform any of these tasks. Note that since all windows are created in the SYS schema, you must always use the [SYS].*window_name* syntax when you reference any window.

Opening a Window When a window opens, there will be a switch in the resource plan, and all jobs that are currently running will see a change in their resource allocation, as specified by the particular resource plan associated with the window that just opened. A window is designed to open at a certain time (`start_time`) and stay open for a fixed length of time (`duration`). The rest of the time, the window stays closed.

A window will automatically open at a time specified by its `start_time` attribute. Only one window can be open at any given time.You can also open a window manually anytime you wish, by using the `OPEN_WINDOW` procedure. Even when you manually open a window prematurely, that window will still open at its regular open time specified by its interval. Here's an example that shows how you can open a window manually:

```
dbms_scheduler.open_window (
window_name =>'BACKUP_WINDOW',
duration => '0 12:00:00');
```

Look at the `duration` attribute in the preceding statement. When you specify the duration, you can specify days, hours, minutes, seconds, in that order. Thus, the setting means 0 days, 12 hours, 0 minutes, and 0 seconds.

Manually opening a window doesn't affect the regular schedule of the window.

Note that you can open an already open window. If you do this, the duration of the window will last a time period equal to its `duration` attribute. That is, if you open a window that has been running for 30 minutes and its duration is 60 minutes, your window will last a total of the initial 30 minutes plus an additional 60 minutes, for a total of 90 minutes.

You must have the MANAGE SCHEDULER system privilege in order to open, close, disable, or drop a window.

Closing a Window To close a window, you use the `CLOSE_WINDOW` procedure, as illustrated by the following example:

```
dbms_scheduler.close_window('BACKUP_WINDOW');
```

If a job is running when you close a window, the job will continue to run to its completion. However, a running job may close upon the closing of its window if you create a job with the attribute `STOP_ON_WINDOW_CLOSE` set to `TRUE`.

You can alter all attributes except the `window_name` attribute by using the `SET_ATTRIBUTES` procedure.

Disabling a Window To disable a window, you use the `DISABLE` procedure, as shown here:

```
dbms_scheduler.disable (name  =>  'BACKUP_WINDOW');
```

You can only disable a window if no job uses that window or if the window isn't open. If the window is open, you can disable it by using the `DISABLE` program with the `FORCE=TRUE` attribute.

Dropping a Window When you drop a window, you automatically remove it from its window group.You can drop a window by using the `DROP_WINDOW` procedure. By using the `FORCE=TRUE` attribute, you can do the following:

- Disable all the jobs that use that window
- Drop an open window

If a job associated with a window is running, a `DROP_WINDOW` procedure will continue to run through to completion and is disabled after it completes. If you set the `STOP_ON_WINDOW_CLOSE` attribute to `TRUE`, however, the job will immediately stop when you drop an associated window. If you use the `FORCE=TRUE` setting, you'll disable all jobs that use that window.

Prioritizing Jobs

As you are aware, all database jobs are collected into job classes. You then map each job class to a specific resource consumer group. A resource plan is assigned to a resource consumer group and thus indirectly to each job class as well, by the Database Resource Manager. The active resource plan (as determined by the currently open window) will apportion resources to groups, thus giving different levels of resources to different jobs, based on their job class.

The Scheduler works closely with the Resource Manager to ensure proper resource allocation to the jobs. The Scheduler will start a job only if there are enough resources to run.

To see which resource consumer group a session belongs to, you can use the following query:

```
SQL> select sid, serial#,username,resource_consumer_group from v$session;
```

Within each Scheduler window, you can have several jobs running, with varying degrees of priority. You can prioritize jobs at two levels—*class* and *job*. The prioritization

at the class level is based on the resources allocated to each resource consumer group by the currently active resource plan. For example, the FINANCE_JOBS class might rank higher than the ADMIN_JOBS class, based on the resource allocations dictated by its active resource plan.

Within the FINANCE_JOBS and ADMIN_JOBS classes, there will be several individual jobs. Each of these jobs has a job priority, which can range from 1 to 5, with 1 being the highest priority. You can use the SET_ATTRIBUTES procedure to change the job priority of any job, as shown here:

```
begin
      dbms_scheduler.set_attribute (
      name          =>    'test_job',
      attribute     =>    'job_priority',
      value         =>     1);
end;
```

The default job priority for a job is 3, which you can verify with the following query:

```
SQL> select job_name, job_priority from dba_Scheduler_jobs;
JOB_NAME                       JOB_PRIORITY
------------------------------ ------------
ADV_SQLACCESS1523128                      3
ADV_SQLACCESS5858921                      3
GATHER_STATS_JOB                          3
PURGE_LOG                                 3
TEST_JOB03                                3
TEST_JOB1                                 3
6 rows selected.
SQL>
```

When you have more than one job within the same class scheduled for the same time, the job_priority of the individual jobs determines which job starts first.

exam Watch

Please take time to distinguish between interclass priority levels and intraclass job priority levels. Be aware that job priority is only within a class, and therefore, a low-priority job belonging to a high-priority class wll run ahead of a high-priority job in a low-priority class.

Window Priorities

Since windows might have overlapping schedules, you may frequently have more than one window open at the same time, each with its own resource plan. At times like this, the Scheduler will close all windows except one, using certain rules of precedence. Here is a summary of the windows precedence rules:

- If two windows overlap, the window with the higher priority opens and the window with the lower priority closes.
- If two windows of the same priority overlap, the active window remains open.
- If you are at the end of a window and you have other windows defined for the same time period, the window that has the highest percentage of time remaining will open.

Window Groups

A window group is a collection of windows, and is part of the SYS schema. Window groups are optional entities, and you may make a window a part of a window group when you create it, or add windows to the group at a later time.You can specify either a single window or a window group as the schedule for a job.

As explained earlier in this chapter, you may take two or more windows that have similar characteristics—for example, some night windows and a holiday window—and club them together to create a "downtime window group." The concept of a window group is for convenience only, and its use is purely optional.

Managing Scheduler Attributes

In earlier sections in this chapter, you've seen how you can use the procedure `SET_ATTRIBUTE` to modify various components of the Scheduler. Attributes like `JOB_NAME` and `PROGRAM_NAME` are unique to the job and program components.

You can retrieve the attributes of any Scheduler component with the `GET_SCHEDULER_ATTRIBUTE` procedure of the `DBMS_SCHEDULER` package.

Unsetting Component Attributes

Use the `SET_ATTRIBUTE_NULL` procedure to set a Scheduler component's attributes to `NULL`. For example, to unset the `comments` attribute of the program `TEST_PROGRAM`, you can use the following code:

```
execute dbms_scheduler.set_attribute_null('TEST_PROGRAM', 'COMMENTS');
```

Altering Common Component Attributes

There are some attributes that are common to all Scheduler components. The procedure `SET_SCHEDULER_ATTRIBUTE` lets you set these common, or *global level*, attribute values, which affect all Scheduler components. Let's discuss these common attributes in the following sections.

exam Watch

Oracle recommends that you set the `default_timezone` attribute to a region's name instead of absolute time zone offset, in order to ensure that daylight saving adjustments are being taken into account.

default_timezone If jobs and windows specifications use the calendering syntax but omit the start date, the Scheduler derives the time zone from the `default_timezone` attribute.

log_history This attribute refers to the number of days the Scheduler will retain job and window logs. The default retention period is 30 days.

max_job_slave_processes The Scheduler determines the optimal number of job slave processes, based on your processing requirements. However, you can set a limit on the number of job slave processes using the `max_job_slave_processes` attribute, whose default value is `NULL`, and the range is from 1 to 999.

Viewing Information About the Scheduler

You can use a set of data dictionary views to manage the Scheduler. Let's take a brief look at the important Scheduler-related data dictionary views.

DBA_SCHEDULER_JOBS This view provides the status and general information about scheduled jobs in your database. Here's a simple query using the view:

```
SQL> select job_name, program_name from
     dba_scheduler_jobs;
JOB_NAME                        PROGRAM_NAME
------------------------------------------------------
ADV_SQLACCESS1523128            ADV_SQLACCESS5858921
GATHER_STATS_JOB                GATHER_STATS_PROG
PURGE_LOG                       PURGE_LOG_PROG
TEST_JOB03
…
SQL>
```

DBA_SCHEDULER_RUNNING_JOBS This view provides you with information regarding currently running jobs.

DBA_SCHEDULER_JOB_RUN_DETAILS This view provides information about status and the duration of execution for all jobs in your database, as the following example shows.

```
SQL> select job_name, status, run_duration
     from dba_Scheduler_job_run_details;
JOB_NAME                STATUS        RUN_DURATION
-------------------------------------------------
GATHER_STATS_JOB        SUCCEEDED     +000 00:09:37
ADV_SQLACCESS5858921    FAILED        +000 00:00:03
SQL>
```

DBA_SCHEDULER_SCHEDULES This view provides information on all current schedules in your database, as shown in the following query:

```
SQL> select schedule_name, repeat_interval
  2* from dba_Scheduler_schedules;
SCHEDULE_NAME                          REPEAT_INTERVAL
-------------------------------------------------------------------------
DAILY_PURGE_SCHEDULE       freq=daily;byhour=3;byminute=0;bysecond=0
SQL>
```

exam Watch

The DBA_SCHEDULER_JOB_RUN_DETAILS view provides information about the status and duration of execution for all Scheduler jobs.

DBA_SCHEDULER_JOB_LOG This view enables you to audit job management activities in your database. The data that this view will contain depends on how you set the logging parameters for your jobs and job classes.

In order to set the logging level at the job class level, you need to use the DBMS_SCHEDULER procedure's LOGGING_FULL and LOGGING_RUNS when you create your job class, as shown in the earlier section "Creating a Job Class."

In order to set the logging levels at the individual job level, you use the SET_ATTRIBUTE procedure of the DBMS_SCHEDULER. In the SET_ATTRIBUTE procedure, you can set the attribute logging_level to two different values:

```
DBMS_SCHEDULER.LOGGING_FULL
DBMS_SCHEDULER.LOGGING_RUNS
```

The LOGGING_RUNS option will merely record the job runs, while the LOGGING_FULL option turns on full job logging. Here is an example showing how you can turn on full job logging at the job level:

```
execute dbms_scheduler.set_attribute ('TESTJOB',
'LOGGING_LEVEL', dbms_Scheduler.LOGGING_FULL);
```

exam Watch

As a DBA, you can set logging at the job class level in order to audit Scheduler jobs. Once you set the logging at the class level, an individual user can only increase the amount of logging. For example, if you set `LOGGING_RUNS` at the job class level, the creator of a job may turn off logging at the job level, yet the Scheduler will log the job run information. A user, however, can increase the logging level to `LOGGING_FULL` for a job, thus raising the level of logging from the `LOGGING_RUNS` level you set to a higher logging level.

Purging Job Logs

By default, once a day, the Scheduler will purge all window logs and job logs that are older than 30 days. As long as the Scheduler is active, it runs the `PURGE_JOB` job to perform this automatic purging of the logs. You can also manually purge the logs by executing the procedure yourself, as shown here:

```
execute dbms_scheduler.purge_log (
log_history => 1,
job_name =>  'TEST_JOB1');
```

You can modify the retention period (the default is 30days) of the logs for a job class by using the `SET_ATTRIBUTE` procedure, as shown here:

```
dbms_scheduler.set_attribute(
'TEST_JOB_CLASS', 'log_history', '7');
```

In the preceding example, the `log_history` attribute resets the log retention period to seven days.

exam Watch

The `log_history` attribute specifies the number of days to keep the logs. It can take a value between 0 and 999. If you set it to 0, it means that you want to purge all log entries daily.

In order to clear *all* window and job logs, use the following command:

```
execute dbms_scheduler.purge_log();
```

CERTIFICATION OBJECTIVE 7.04

Database Resource Manager Enhancements

The Database Resource Manager helps you to effectively manage your server resources. There are several important enhancements in the Resource Manager tool in Oracle Database 10*g*. Using the `idle_time` resource plan, you can set limits on idle time for user sessions that are blocking other sessions. You can specify that a session is returned to its original consumer group automatically, after a top call. You can now use consumer group mappings to assign priorities to consumer groupings. There is also a new CPU allocation method in the `DBMS_RESOURCE_MANAGER` package.

Let's start our discussion of the Resource Manager enhancements by looking at the new ways you can set idle time-outs in Oracle Database 10*g*.

Setting Idle Time-Outs

You can now limit the maximum idle time for a session as well as the maximum time an idle session can block another session. Until now, the only way to limit session idle time was by setting the value of the `IDLE_TIME` attribute while creating or altering a database profile.

You set the new idle time limits using the `CREATE_PLAN_DIRECTIVE` procedure of the Database Resource Manager. The new argument `MAX_IDLE_TIME` determines the maximum idle time of a session. The other new argument, `MAX_IDLE_BLOCKER_TIME`, determines the maximum amount of time an idle session can block another session. Here is an example:

```
exec dbms_resource_manager.create_plan_directive
(plan => 'new_plan', -
 group_or_subplan => 'sales',
 comment => 'sales group', -
 cpu_p1 => 60,
 parallel_degree_limit_p1_P1 => 4
 max_idle_time => 600,
 max_idle_blocker_time => 300);
```

on the job

The default value for both the `MAX_IDLE_TIME` and `MAX_IDLE_BLOCKER_TIME` is `NULL`, which implies that the idle time is unlimited.

In the preceding example, the maximum time a session can remain idle is capped at 600 seconds by the MAX_IDLE_TIME attribute. After this time, if a session continues to sit idle, without executing or waiting for I/O, the PMON process kills the session and cleans up after it. The MAX_IDLE_BLOCKER_TIME ensures that the PMON also kills any session that is idle for more than 300 seconds and is blocking another session. The PMON process grabs the offending sessions during its regular database checks, which it conducts on a minutely basis.

Automatic Switching Back to Initial Consumer Groups

When you create plan directives for the Database Resource Manager using the CREATE_PLAN_DIRECTIVE procedure of the DBMS_RESOURCE_MANAGER package, you can use several switch parameters to stipulate the action to be taken after a specified time period. The SWITCH_TIME parameter specifies the length of time that a session can continue to execute before a certain action is taken.

In Oracle Database 10g, you now have a new switch parameter called SWITCHTIME_IN_CALL. Like the SWITCH_TIME parameter, this parameter also specifies the time that a session can execute before a certain action is taken. However, the SWITCH_TIME_IN_CALL parameter has only one specific purpose: if you specify a value for this parameter, say 600 seconds; after that interval passes, at the end of the call, Oracle will switch the consumer group of that session to its initial consumer group. The initial consumer group you started with is specified by the SWITCH_GROUP attribute in the CREATE_PLAN_DIRECTIVE procedure that you use to implement the Database Resource Manager. Unlike SWITCH_TIME, the SWITCH_TIME_IN_CALL parameter will only downgrade your session to a lower priority group for that one call, whereas SWITCH_TIME will downgrade you permanently.

You cannot specify both SWITCH_TIME and SWITCH_TIME_IN_CALL plan directives for a single consumer group.

The ability to switch sessions back to their initial consumer group at the end of their top call is a huge enhancement because it means that you can now use the Resource Manager in the web environment, with connection pooling from the middle tier. Previously, if you downgraded a connection because one web client issued a horrific statement, it would stay downgraded for any subsequent jobs, even if they came from other web clients.

Creating Mappings to Assign Priorities to Resource Groups

At any given time, a user session can belong to only one resource consumer group. You can now configure the Database Resource Manager to assign consumer groups to sessions by mapping various session attributes to consumer groups. The ability to map session attributes and consumer groups enables you to do two things:

- Assign sessions to consumer groups
- Assign priorities to indicate which mappings have precedence

Let's first look at how you can assign consumer groups to sessions.

Creating Mapping Between Sessions and Consumer Groups

There are two types of session atrributes—*login attributes* and *run-time attributes*. When a user first logs into the database, the Resource Manager looks at the login attributes in order to evaluate to which consumer group it should assign the session. The run-time attributes are useful when an already logged in user's session needs to be assigned to another consumer group. Let's look at the individual session atrributes in more detail, in the following section.

on the Job

The ability to create session mappings is another enhancement that strengthens the use of the Database Resource Manager in managing web applications. In earlier versions of Oracle, the `ORACLE_USER` attribute was the only mapper available, and all your web clients were likely to use the same `ORACLE_USER` attribute through a connection pool. But in Oracle Database 10g, the Database Resource Manager can map sessions to groups according to what they are doing.

Session Attributes These are the key elements in the mapping of sessions and consumer resource groups. As mentioned in the previous section, there are two types of session attributes—run-time and login. Following are the login attributes:

- **`ORACLE_USER`** The standard Oracle Database username
- **`CLIENT_OS_USER`** The operating system name of the user
- **`CLIENT_MACHINE`** The name of the machine from which the user is connecting to Oracle
- **`CLIENT_PROGRAM`** The name of the program the user is employing to log into the database, for example, a SQL*Plus session
- **`SERVICE_NAME`** The name used by the user to log into the database

There are five session attributes:

- **MODULE_NAME** The module name in the application that is currently executing in the database
- **MODULE_NAME_ACTION** The current module name and action being performed by the session
- **SERVICE_MODULE** A combination of the service and the module name
- **SERVICE_MODULE_ACTION** A combination of the service name, module name, and action name
- **EXPLICIT** An explicit mapping requested by the client

Creating the Mappings You map a session's attributes to a consumer group by using the `SET_CONSUMER_GROUP_MAPPING` procedure of the `DBMS_RESOURCE_MANAGER` package. You create an explict mapping between each session attribute and the resource group pair, as shown here:

```
execute dbms_resource_manager.set_consumer_group_mapping (
DBMS_RESOURCE_MANAGER.ORACLE_USER, 'scott', 'dev group');
```

In the previous listing, a user with the operating system name SAM can log in as the Oracle user SCOTT, and will be assigned to the "dev" consumer resource group. Notice that the `ORACLE_USER` attribute is being mapped to the resource group 'dev group'. In the following example, I map user SAM to the 'prod group', based on the `CLIENT_OS_USER` attribute.

```
execute dbms_resource_manager.set_consumer_group_mapping -
(DBMS_RESOURCE_MANAGER.CLIENT_OS_USER, 'SAM', 'prod_group');
```

Automatic Consumer Group Switching Oracle will follow an automatic consumer group switching policy based on the specific mapping in force at any given time. Here's a summary of the switching policy:

- When you first log in, you are mapped to the initial resource group based on the attribute being mapped.
- If you change a run-time attribute later, you are switched automatically to a different consumer group.
- It is possible to be switched to the same consumer group you are currently in.

Assigning Mapping Priorities As you can surmise, a single session can be mapped to several consumer resource groups, because you can map each session attribute to a consumer group. You may, for example, map a resource group based on the value of a session's MACHINE_NAME, CLIENT_OS_USER, MODULE_NAME, CLIENT_NAME, SERVICE_NAME_ACTION, or some other login or run-time attribute. Naturally, there is room for a conflict or ambiguity regarding which resource group to assign to a session, since there are multiple attributes to each session, and each of them may be assigned to a different resource group. The Database Resource Manager uses priority levels to resolve such ambiguities. Let's look at how you can assign priorities among session attributes.

exam Watch

A session must be a member of a particular resource group in order to be switched to it—a mere mapping between a session attribute and that group won't suffice to switch resource groups.

You set session attribute mapping priorities by using the SET_CONSUMER_GROUP_MAPPING_PRI procedure of the DBMS_RESOURCE_MANAGER package. This procedure helps you prioritize session mappings. In order to establish priorities among the various attributes, you assign each session attribute a priority level that ranges from 1 to 10, with 1 being the highest and 10 the lowest. Here's an example:

```
SQL> dbms_resource_manager.set_consumer_group_mapping_pri (
(explicit  => 1,
client_machine  => 2,
module_name   => 3,
oracle_user   => 4,
service_name  => 5,
client_os_user  => 6,
client_program  => 7,
module_name_action  => 8,
service_module  => 9,
service_module_action  => 10);
```

on the Job

All the priority levels you see in the preceding listing are the default priority levels of each of the session attributes, which you may change with the SET_CONSUMER_GROUP_MAPPING procedure.

In the preceding example, the session attribute ORACLE_USER has a priority level of 4. Thus, even if the ORACLE_USER and the CLIENT_OS_USER attributes are mapped to different consumer resource groups, Oracle will assign the user to the group mapped to the ORACLE_USER session attribute because its priority level (4) is higher than the priority level of the CLIENT_OS_USER attribute (6).

exam Watch

Application developers may also set the MODULE_NAME and MODULE_NAME_ACTION through the use of the DBMS_APPLICATION_INFO package. The SERVICE_NAME attribute is the connect string that you specify in your tnsnames.ora file.

The MODULE_NAME and MODULE_NAME_ACTION attributes offer you powerful capabilities when you are using web applications. Typically, middle-tier applications use a single username for all the users, but use different MODULE_NAME and MODULE_NAME_ACTION attributes to distinguish between the type of work performed by individual users. The resource manager can now detect what application module a session is actually invoking at any given moment, and assign resources appropriately (by associating the session to the appropriate resource consumer group), independently of the DB logon (which is probably pooled from a middle tier).

New Database Resource Manager Allocation Methods

In Oracle Database 10*g*, you have new methods of allocating CPU resources. When you use the CREATE_CONSUMER_GROUP procedure, you can now set the CPU_MTH attribute to a RUN_TO_COMPLETION setting. When you use the CREATE_PLAN procedure to create a resource plan, you have a new setting called RATIO, for the CPU_MTH variable. Let's take a quick look at both of these changes in the following sections.

The RUN_TO_COMPLETION Allocation Method When you create a consumer group using the CREATE_CONSUMER_GROUP procedure, the CPU_MTH option provides the method to distribute your CPU among the sessions in the consumer group. The default value for the CPU_MTH option is ROUND_ROBIN. The new RUN_TO_COMPLETION method specifies that the session with the largest active time should be scheduled ahead of other sessions. Here's an example:

```
exec dbms_resource_manager.create_consumer_group (
consumer_group   => 'sales',
cpu_mth  => 'RUN TO COMPLETION',
comment  => 'this is the new 10g cpu_mth option');
```

on the Job

Usually, you would give batch and large data warehouse-type jobs a low priority to avoid impacting your OLTP operations. But the RUN_TO_COMPLETION CPU allocation method accords top priority to large jobs run by a particular group.

The Ratio Allocation Method Previously, in Oracle9*i*, when you created a new resource plan, the CPU_MTH (the CPU resource allocation method) could only take a single value—EMPHASIS. In Oracle 10*g*, you can now use a new CPU allocation method when you create a resource plan, called RATIO. The RATIO allocation method is meant for single-level resource plans that use *ratios* to specify the allocation of CPU. EMPHASIS remains as the default CPU allocation method, and it is for multilevel plans that use *percentages* to specify how CPU is allocated among various resource groups. Following is an example that shows how to specify RATIO as a resource plan's CPU allocation method.

```
dbms_resource_manager.create_plan
(plan => 'service_level_plan',
cpu_mth -> 'RATIO',
comment => 'service level plan');
```

Once you choose the new RATIO allocation policy by specifying it as the CP_MTH value in the CREATE_PLAN statement, you aren't quite ready to use the ratio method. You must also use the CREATE_PLAN_DIRECTIVE procedure and set the CPU_P1 directive to actually set the ratios for the CPU allocation. Here is an example:

```
SQL> dbms_resource_manager.create_plan_directive
   (plan => 'service_level_plan',
    group_or_subplan => 'GOLD_CG',
    comment => 'Gold service level customers',
    cpu_p1 => 10);
dbms_resource_manager.create_plan_directive
   (plan => 'service_level_plan',
    group_or_subplan => 'SILVER_CG',
    comment => 'Silver service level customers',
    cpu_p1 => 5);
dbms_resource_manager.create_plan_directive
   (plan => 'service_level_plan',
    group_or_subplan => 'BRONZE_CG',
    comment => 'Bronze service level customers',
    cpu_p1 => 2);
dbms_resource_manager.create_plan_directive
    (plan => 'service_level_plan',
    group_or_subplan => 'OTHER_GROUPS',
    comment => 'Lowest priority sessions',
    cpu_p1 => 1);
```

In the preceding example, there are four consumer groups—Gold_CG, Silver_CG, Bronze_CG, and the default OTHER_GROUPS. The CPU is allocated using the RATIO method. For every 18 units, the ratio of CPU allocation would be 10:5:2:1 for

exam Watch

You must understand that the RUN_TO_COMPLETION and the old ROUND_ROBIN allocation methods are part of the CREATE_CONSUMER_GROUP procedure, and they apply to resource consumer groups. The RATIO and the old EMPHASIS allocation methods, on the other hand, are used with the CREATE_PLAN procedure and apply to resource plans.

the GOLD_CG, SILVER_CG, BRONZE_CG, and OTHER_GROUPS consumer groups, respectively. If at times you only have sessions for the first two groups—GOLD_CG and SILVER_CG—then these two groups would split the CPU resources in a 10:5 ratio.

INSIDE THE EXAM

The exam has questions that test your familiarity with the logging and purging of Scheduler jobs. Know the default purging procedure for jobs and the various logging levels. Which logging level gives you the most information?

You must remember the names and functions of the basic and advanced Scheduler components. What is the purpose of using advanced components like job classes, windows, and window classes? What privileges do you need to manage each of these components? The test will most likely probe your knowledge of job priorities. Know the difference between inter-job class and intra-job class priorities. When windows overlap, how does the Scheduler determine window priorities?

You must know how to set repeat intervals for jobs. Pay particular attention to calendaring expressions. Practice with calendaring expressions that set repeat intervals like once a week and every five minutes.

You must know the DBMS_SCHEDULER package very well. Understand the different parameters like start_date, end_date, repeat_interval, and duration. How and which Scheduler attributes can you set at the global level? The exam will test your knowledge of the DBA_SCHEDULER_JOB_RUN_DETAILS and DBA_SCHEDULER_JOB_LOG views.

The exam tests your knowledge of the new resource allocation methods when you use the DBMS_RESOURCE_MANAGER package. What do the RATIO and the RUN_TO_COMPLETION resource allocation methods do? Know how to assign priorities using the DBMS_RESOURCE_MANAGER package, and how mappings are created between session attributes and specific consumer resource groups.

CERTIFICATION SUMMARY

The chapter started with an introduction to the all-new Oracle Scheduler feature. You learned about the basic building blocks of the Scheduler—jobs, programs, and schedules. You also learned how to create and administer these basic components.

You were introduced to the main features of the advanced Scheduler components, including job classes and windows. You saw how you can associate job classes with resource consumer groups. You also learned how you can control resource allocation by associating resource plans with Scheduler windows.You learned how to query the main data dictionary view concerning the Scheduler.

In the last part of this chapter, you learned about the advancements in the Database Resource Manager feature. These include the new idle time-outs for consumer groups, automatic switching of consumer groups at the end of the top call, and the automatic assignment of sessions to consumer groups through the mapping of session attributes. You then learned how to utilize the new `CPU_MTH` methods for allocating CPU—the `RUN_TO_COMPLETION` and `RATIO` methods.

TWO-MINUTE DRILL

Simplifying Management Tasks by Using the Scheduler

- ❑ The Scheduler replaces `DBMS_JOB` as the main scheduling facility in Oracle Database 10*g*.
- ❑ Use the `DBMS_SCHEDULER` package to manage the Scheduler.
- ❑ You can schedule PL/SQL programs, Java programs, native binary executables, and shell scripts.
- ❑ Jobs, programs, and schedules are the basic Scheduler components.
- ❑ A job is a task you run one or more times.
- ❑ You can run a job by itself or with a saved program and/or a saved schedule.
- ❑ A schedule tells you when and how frequently you should run a job.
- ❑ A program contains metadata about a job.
- ❑ A job class groups several similar jobs into one large entity.
- ❑ In a job class, the priority levels range from 1 to 5.
- ❑ A higher-priority job always starts before a lower-priority job.
- ❑ A window represents a time during which you can schedule a job.
- ❑ A window always has a resource plan associated with it.
- ❑ A window group is a collection of windows.
- ❑ The Scheduler assigns each job class to a specific resource consumer group.
- ❑ The default resource consumer group is the `DEFAULT_CONSUMER_GROUP`.
- ❑ The default job class is the `DEFAULT_JOB_CLASS`.
- ❑ The job table records information about all the jobs run by the Scheduler.
- ❑ The background process cjqNNN automatically starts and monitors the running of a job.
- ❑ The job slaves start database sessions and execute the job.
- ❑ Jobs, programs, and schedules are created in the current user's schema.
- ❑ Job classes, windows, and windows groups are created in the SYS schema.
- ❑ To create the basic components, you need the CREATE JOB privilege.
- ❑ To use the advanced Scheduler components, you must have the MANAGE SCHEDULER system privilege.

- The EXECUTE ANY PROGRAM privilege lets a user execute a program under any schema.
- The SCHEDULER_ADMIN role contains all the Scheduler system privileges.

Managing the Basic Scheduler Components

- All Scheduler objects are of the form [*schema*].*name*.
- When you create a job, the parameter values you specify will override the default values for programs and schedules.
- You create a job using the `CREATE_JOB` procedure.
- You can run a PL/SQL block, a Java program, a shell script, or a native binary executable as part of your job.
- You can use a PL/SQL date/time expression or a calendaring expression for setting the `repeat_interval` parameter.
- Calendaring expressions have three components: frequency, repeat interval, and any optional specifiers.
- You must have the MANAGE SCHEDULER privilege in order to use the `FORCE` setting when you stop or drop a job.

Managing Advanced Scheduler Components

- The Scheduler uses job classes, windows, and window classes to prioritize jobs in the database.
- Each job class is associated with a resource consumer group.
- The logging level when you create a job class can take the values `LOGGING_OFF`, `LOGGING_RUNS`, or `LOGGING_FULL`.
- The default logging level is `LOGGING_RUNS`.
- The purpose of a window is to automatically switch resource plans during certain periods.
- Windows consist of three basic components: schedules, duration, and resource plan.
- Window priorities can take the values `LOW` or `HIGH`.
- You can have overlapping windows, since the Scheduler doesn't check before creating new windows.
- You can use the `SET_ATTRIBUTES` procedure to change the job priority of a job.

- ❑ Window priorities are resolved by checking which window has the higher priority level, or whichever window opened first, if the window priorities are the same.
- ❑ You can use the `SET_ATTRIBUTE_NULL` procedure to set a Scheduler component's attributes to `NULL`.

Viewing Information About the Scheduler

- ❑ The `DBA_SCHEDULER_JOBS` view provides information about the status and general information about scheduled jobs.
- ❑ The `DBA_SCHEDULER_RUNNING_JOBS` view provides information about currently running jobs.
- ❑ The `DBA_SCHEDULER_JOB_RUN_DETAILS` view provides information about the status and the duration of execution for all jobs.
- ❑ The `DBA_SCHEDULER_SCHEDULES` view provides information on all saved schedules.

Database Resource Manager Enhancements

- ❑ You can now set idle time limits using the `CREATE_PLAN_DIRECTIVE` procedure of the `DBMS_RESOURCE_MANAGER` package.
- ❑ You can use the `MAX_IDLE_TIME` parameter to limit idle sessions. You can use the `MAX_IDLE_BLOCKER_TIME` to limit idle sessions that are blocking other sessions.
- ❑ The new `SWITCH_TIME_IN_CALL` parameter automatically returns a session to its original resource consumer group, after the database completes the top call.
- ❑ You can create mappings between session attributes and resource consumer groups.
- ❑ You can create as many mappings as there are session attributes.
- ❑ You can assign session attribute mapping priorities by using the `DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING_PRI` procedure.
- ❑ You can now set the `CPU_MTH` attribute to the new `RUN_TO_COMPLETION` setting.
- ❑ You can now use the new CPU allocation method `RATIO`, which uses ratios to specify the allocation of CPU resources among various resource groups.

SELF TEST

The following questions will help you measure your understanding of the material presented in this chapter. Read all the choices carefully because there might be more than one correct answer. Choose all correct answers for each question.

Simplifying Management Tasks by Using the Scheduler

1. Whose job is it to update the job table and the log table?
   - A. The worker (slave) process
   - B. The SYS user
   - C. The Database Resource Manager
   - D. The cjqNNN background process
2. Which of the following do you need to specify a stored schedule that's owned by a different user?
   - A. You must have the EXECUTE ANY SCHEDULE privilege.
   - B. You don't need any privileges to use a stored schedule.
   - C. You need the MANAGE SCHEDULER privilege.
   - D. You need the EXECUTE ANY JOB privilege.
3. Which of the following is true when you are using saved schedules and programs as part of a new job?
   - A. If you explicitly use parameters that are also in the schedules or programs, the parameter values you specify will override the values specified for the parameters in the schedules and programs.
   - B. If you explicitly use parameters that are also in the schedules or programs, the parameter values you specify will be overridden by the values specified for the parameters in the schedules and programs.
   - C. You can't use separate parameter values in a job creation statement if they are already part of a schedule or a program.
   - D. You cannot use saved schedules and programs with a new job.
4. If you create a job in user SCOTT's schema, who will be the owner of the job?
   - A. SCOTT will be the owner of the job.
   - B. SYS will be the owner of the job.
   - C. You will be the owner of the job.
   - D. PUBLIC will be the owner of the job.

Managing the Basic Scheduler Components

5. What is the correct syntax to create a schedule that executes every second?

A. `FREQ=SECONDLY; INTERVAL=1/60`
B. `FREQ=SECONDLY; INTERVAL=1`
C. `FREQ=SECONDLY`
D. `FREQ=HOURLY; INTERVAL=1/60`

6. Regarding your ability to combine the various components of the Scheduler, which of the following are possible?

A. Use a job by itself all the time.
B. Use a job and a program together.
C. Use a job, schedule, and a program together.
D. Use a program and a schedule together.

7. What is the priority level for a job in a job class?

A. From 1 to 5
B. From 1 to 999
C. From 1 to 10
D. A job can only take a value of HIGH or LOW.

8. Scheduler job table information can be seen in which view?

A. `DBA_RESOURCE_MANAGER` view
B. `DBA_JOBS` view
C. `DBA_SCHEDULER_JOBS` view
D. `DBA_SCHEDULER` view

Managing Advanced Scheduler Components

9. You create a new job class and name it `my_job_class`. Where will this job class be saved?

A. In your schema.
B. In the SYS schema.
C. In all schemas that are part of the database.
D. Job classes aren't saved under any one schema.

10. You create a job class and set the `LOGGING_RUNS` option for the `LOGGING_LEVEL` parameter. What will be the result?

A. The database will not perform any logging for individual jobs.
B. The database will not perform a detailed logging of the job runs.
C. The database will perform a detailed logging of the job runs.
D. For each run of a job, there will just be a mention of the job only in the job log.

11. The DBA wishes to drop an existing job class. The DBA notices that a job from the job class she wants to drop is currently running. Which of the following scenarios would be true?

A. You cannot drop the jobs along with the job class definition.
B. If you use the `FORCE=TRUE` option, the running jobs will stop immediately and the job class will be removed as well.
C. Just use the `DROP_JOB_CLASS` with no options to drop the job class and kill any running jobs right away.
D. If you use the `FORCE=TRUE` option, the running jobs will continue to completion anyway, but the job class and its jobs will be removed.

12. What important attribute does a window have, that a schedule doesn't?

A. A program
B. An interval
C. A resource plan
D. A resource consumer group

Viewing Information About the Scheduler

13. To see which Scheduler jobs are currently running, which view would you use?

A. `DBA_SCHEDULER_RUNNING_JOBS` view
B. `DBA_SCHEDULER_JOB_RUN_DETAILS` view
C. `DBA_SCHEDULER_SCHEDULES` view
D. `DBA_SCHEDULER_JOBS` view

14. Using the `SET_ATTRIBUTE` procedure, you can set the `logging_level` attribute to which two values?

A. `LOGGING_NONE` and `LOGGING_RUNS`
B. `LOGGING_DEFAULT` and `LOGGING_RUNS`

C. LOGGING_DETAILED and LOGGING_RUNS
D. LOGGING_FULL and LOGGING_RUNS

15. What will the command EXECUTE DBMS_SCHEDULER.PURGE_LOG(); do?

A. Purge only all the window logs
B. Purge only all the job logs
C. Purge all window and job logs
D. Purge only yesterday's job logs

16. Where can you find the status and duration of execution of all jobs in your database?

A. DBA_SCHEDULER_JOB_RUN_DETAILS
B. DBA_SCHEDULER_RUNNING_JOBS
C. DBA_SCHEDULER_SCHEDULES
D. DBA_SCHEDULER_JOBS

Database Resource Manager Enhancements

17. Which procedure of the DBMS_RESOURCE_MANAGER package enables the DBA to set idle time limits?

A. DBMS_RESOURCE_MANAGER.create_plan_directive
B. DBMS_RESOURCE_MANAGER.create_resource_plan
C. DBMS_RESOURCE_MANAGER.create_resource_group
D. DBMS_RESOURCE_MANAGER.idle_time

18. In the context of using the DBMS_RESOURCE_MANAGER package to set mapping priorities, what kind of attribute is MODULE_NAME?

A. Run-time session attribute
B. Neither a run-time nor a login session attribute
C. Login attribute
D. Could be either a run-time attribute or a login session attribute

19. What is the default value for the CPU_MTH option when you are creating a consumer group?

A. ROUND_ROBIN
B. RATIO
C. RUN_TO_COMPLETION
D. DEFAULT

20. When you create a mapping between consumer groups and session attributes, how many consumer resource groups can a single session map to?

A. Only one consumer resource group
B. At most two resource consumer groups
C. As many resource consumer groups as there are session attributes
D. A unique consumer resource group each time you log in again

LAB QUESTIONS

Lab 1

Use the Database Resource Manager to ensure that all idle sessions will not last more than 20 minutes. Also ensure that you kill all idle sessions that have been around for more than a minute and are blocking other user sessions.

Lab 2

Create a schedule, a program, and a job to gather data at five-minute intervals in your database.

SELF TEST ANSWERS

Simplifying Management Tasks by Using the Scheduler

1. ☑ **A.** The worker (slave) process updates the job table and log table.
   ☒ **B** and **C** are wrong since neither the SYS user nor the Database Resource Manager play any role in updating the log and the job tables. **D** is wrong since the cjqNNN background process identifies the job coordinator, who doesn't execute any chores directly, but rather hands them to the slave (worker) processes. These jobs include updating the Scheduler job and log tables.

2. ☑ **B.** You don't need any privileges to use any of the saved schedules in the database.
   ☒ **A, C,** and **D** are wrong since you don't need any specific privileges to use a saved schedule.

3. ☑ **A.** If you use separate parameter values for any schedule- or program-related parameters while creating a job, those values will override the values for these parameters specified in any schedules or programs that you may be using in the job creation statement.
   ☒ **B** is wrong because it states the opposite of the correct answer. **C** is wrong since you can specify parameter values for the parameters specified in a schedule or a program. **D** is wrong since you can use saved programs and schedules when you create a new job.

4. ☑ **A.** The creator of a job need not be the owner of a job. User Scott will be the owner of the job since you're creating the job in Scott's schema.
   ☒ **B, C,** and **D** are incorrect since a job is owned by the user in whose schema you create the job.

Managing the Basic Scheduler Components

5. ☑ **B** and **C. B** is correct since `FREQ=SECONDLY` will help you execute a schedule that will execute on a minutely basis. Answer **C** will also execute the schedule every second.
   ☒ **A** is wrong since it states that the program will execute 60 times in one second! **D** is wrong since it will execute the job every minute, not every second.

6. ☑ **A, B,** and **C** are all correct. You can use a job all by itself or with either or both of the optional objects—schedule and program.
   ☒ **D** is wrong since you can't use the Scheduler without a job.

7. ☑ **A.** The priority level for a job can range from 1 to 5.
   ☒ **B, C,** and **D** are the wrong priority levels.

8. ☑ **C.** The `DBA_SCHEDULER_JOBS` view captures the data in the Scheduler job table.
   ☒ **A, B,** and **D** refer to the wrong views.

Managing Advanced Scheduler Components

9. ☑ **B** is correct because all job classes are saved in the SYS schema.
☒ **A** and **C** are wrong since they point to the wrong schemas. **D** is wrong since it says that job classes can't be saved in any schema.

10. ☑ **C.** `LOGGING_RUNS` means there will be a detailed log of all job runs.
☒ **A** is wrong since there will be logging. **B** is wrong since there will be detailed logging of all job runs. **D** is wrong since the logging isn't merely limited to the mention of the job—there will be a detailed logging for each job.

11. ☑ **D** is correct since using the `FORCE=TRUE` option will remove the job class, but any currently running jobs will continue to completion.
☒ **A** is wrong since you can drop a job along with its job class. However, note that when you drop a job class, its jobs aren't really dropped, but *disabled*. **B** is wrong since the `FORCE=TRUE` option won't stop a running job before it completes. **C** is wrong since you can't kill running jobs while dropping a job class.

12. ☑ **C.** A window always has a resource plan, but a schedule doesn't have one.
☒ **A** and **B** are wrong since both windows and schedules have a program and interval. **D** is wrong since you don't have to specify the resource consumer group for either a window or a schedule.

Viewing Information About the Scheduler

13. ☑ **A.** The `DBA_SCHEDULER_RUNNING_JOBS` view shows you details about all currently running jobs.
☒ **B** is wrong since the `DBA_SCHEDULER_JOB_RUN_DETAILS` view shows you the status and duration of execution of all jobs in your database. **C** is wrong because the `DBA_SCHEDULER_SCHEDULES` view provides information on all schedules in your database. **D** is wrong because the `DBA_SCHEDULER_JOBS` view shows the status and general information for all your scheduled jobs.

14. ☑ **D.** `LOGGING_FULL` and `LOGGING_RUNS` options are valid.
☒ **A, B,** and **C** are wrong since they contain a nonexistent logging option.

15. ☑ **C.** The command will purge all window and job logs.
☒ **A** is wrong since the command will also purge all job logs. **B** is wrong because the command will also purge all window logs. **D** is wrong because the command will purge all window and job logs, not just yesterday's logs.

16. ☑ **A.** The DBA_SCHEDULER_JOB_RUN_DETAILS view provides the status and duration of execution of all jobs.
☒ **B, C,** and **D** point to the wrong data dictionary views. Please refer to the explanation for Question 13 for more details on why these are incorrect answers.

Database Resource Manager Enhancements

17. ☑ **A.** The CREATE_PLAN_DIRECTIVE procedure lets you specify idle time limits for sessions.
☒ **B** is wrong since the CREATE_RESOURCE_PLAN directive helps you specify resource plans. **C** is a wrong choice since the CREATE_RESOURCE_GROUP is only useful for specifying the members of a resource consumer group. **D** refers to a nonexistent procedure.

18. ☑ **A.** The MODULE_NAME attribute belongs to the group of run-time session attributes.
☒ **B, C,** and **D** are wrong because MODULE_NAME is a run-time attribute.

19. ☑ **A.** The old ROUND_ROBIN method is still the default method.
☒ **B, C,** and **D** aren't default values for the CPU_MTH option.

20. ☑ **C.** You can map a single session to as many resource consumer groups as there are session attributes, since you can map each of the attributes to a separate resource consumer group.
☒ **A, B,** and **D** provide the wrong alternatives.

LAB ANSWERS

Lab 1

To kill idle sessions that go over a set idle time limit, use the DBMS_RESOURCE_MANAGER package in the following way, and set the MAX_IDLE_TIME and MAX_IDLE_BLOCKER_TIME parameters. (Note that you must create the pending area before you can create the following plan.)

```
SQL> begin
       dbms_resource_manager.create_plan_directive (
       plan                          =>       'TEST_PLAN',
       group_or_subplan              =>       'TEST_GROUP',
       comment                       =>       'Limit user idle time',
       max_idle_time                 =>       600,
       max_idle_blocker_time         =>       300);
     end;
```

Lab 2

1. Create the program.

```
SQL> begin
      dbms_scheduler.create_program (
      program_name           =>      'DAILY_GATHERING',
      program_action         =>      'DAILY_DATA_GATHERING',
      program_type           =>      'STORED_PROCEDURE',
      enabled                =>       'TRUE');
    end;
```

2. Create the schedule.

```
SQL> begin
       dbms_scheduler.create_job (
       job_name               =>    'DAILY_GATHERING_JOB',
       program_name           =>    'DAILY_GATHERING',
       start_date             =>    'TRUNC(SYSDATE+1)+22/24',
       repeat_interval        =>    'TRUNC(SYSDATE+1)+22/24',
       comments               =>    'Daily Data Gathering Job.');
      end;
SQL> begin
       dbms_scheduler.create_schedule (
       schedule_name          =>    'TEST_SCHEDULE',
       start_date             =>    SYSTIMESTAMP,
       repeat_interval        =>    'FREQ=MINUTELY;INTERVAL=5',
       comments               =>    'A test schedule.');
     end;
```

3. Create the job using the program and schedule you created in the previous two steps.

```
SQL> begin
       dbms_scheduler.create_job (
       job_name               =>     'DAILY_GATHERING_JOB',
       program_name           =>     'DAILY_GATHERING',
       schedule_name          =>     'TEST_SCHEDULE');
     end;
```