

17

Automating Tasks with the Scheduler

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Simplify management tasks by using the Scheduler
- Create a job, program, and schedule
- Monitor job execution
- Use a time-based or event-based schedule for executing Scheduler jobs
- Describe the use of windows, window groups, job classes, and consumer groups
- Use email notification
- Use job chains to perform a series of related tasks
- Describe Scheduler jobs on remote systems
- Use advanced Scheduler concepts to prioritize jobs

ORACLE

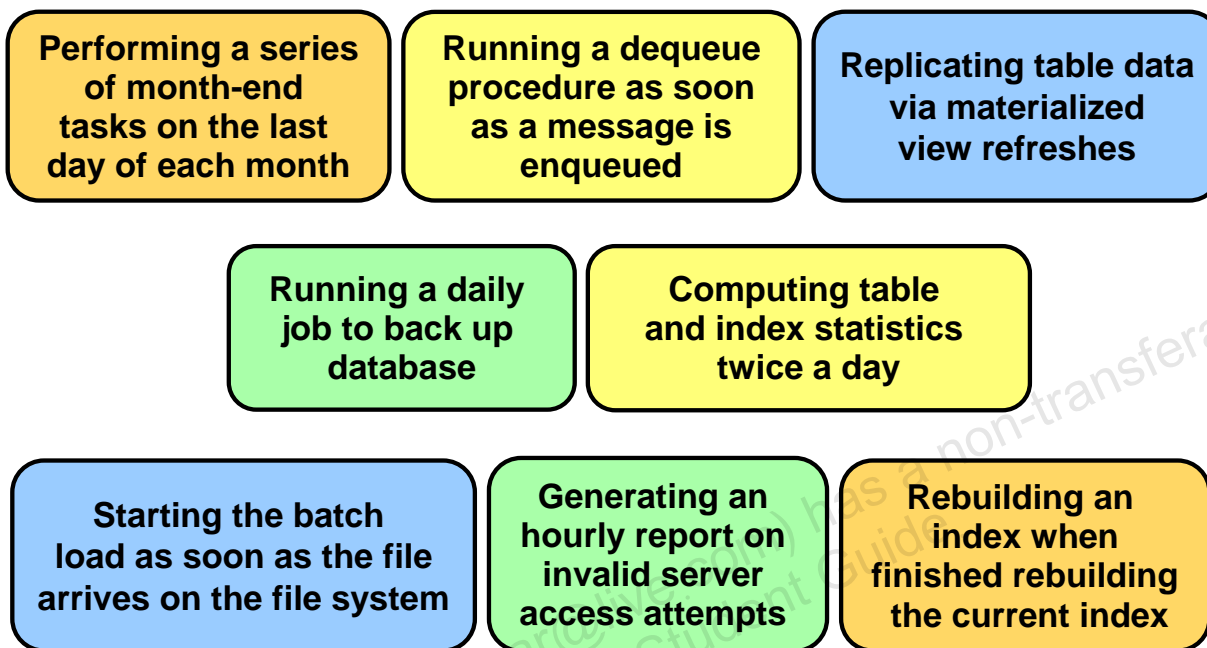
Copyright © 2009, Oracle. All rights reserved.

Objectives

For information about the various Scheduler components and their interaction, see the *Oracle Database Administrator's Guide*.

For detailed information about the DBMS_SCHEDULER package, see the *Oracle Database PL/SQL Packages and Types Reference*.

Simplifying Management Tasks



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Simplifying Management Tasks

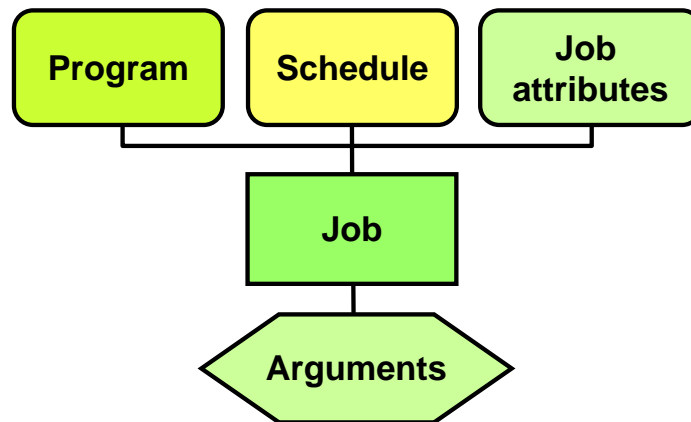
Many tasks in the Oracle environment need job-scheduling capabilities. Routine database maintenance and application logic require jobs to be scheduled and run periodically. Business-to-business (B2B) applications require scheduling for their business events. DBAs need to schedule regular maintenance jobs in specified time windows.

The Oracle database provides advanced scheduling capabilities through the database Scheduler, which is a collection of functions and procedures in the DBMS_SCHEDULER package. The Scheduler can be invoked in any SQL environment, or through Enterprise Manager (EM).

The Scheduler enables database administrators and application developers to control when and where various tasks take place in the database environment. These tasks can be time consuming and complicated; using the Scheduler, you can manage and plan these tasks.

Scheduler jobs can be started based on time or when a specified event occurs, and the Scheduler can raise events when a job's state changes (for example, from RUNNING to COMPLETE). You can also use a named series of programs that are linked together for a combined objective.

Core Components



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Core Components and Key Steps

A job has two mandatory components: “what” action needs to be done and the time or schedule “when” the action occurs. The “what” is expressed in the command region and the job attributes: the `job_type` and the `job_action` parameters. The “when” is expressed in a schedule, which can be based on time or events, or be dependent on the outcome of other jobs.

The Scheduler uses the following basic components:

- A **job** specifies what needs to be executed. It could be as an example a PL/SQL procedure, a native binary executable, a Java application, or a shell script. You can specify the program (what) and schedule (when) as part of the job definition, or you can use an existing program or schedule instead. You can use arguments for a job to customize its run-time behavior.
- A **schedule** specifies when and how many times a job is executed. A schedule can be based on time or an event. You can define a schedule for a job by using a series of dates, an event, or a combination of the two, along with additional specifications to denote repeating intervals. You can store the schedule for a job separately and then use the same schedule for multiple jobs.
- A **program** is a collection of metadata about a particular executable, script, or procedure. An automated job executes some task. Using a program enables you to modify the job task, or the “what,” without modifying the job itself. You can define arguments for a program, enabling users to modify the run-time behavior of the task.

Your Basic Work Flow

To simplify management tasks with the Scheduler:

1. Create a program (enabled or disabled)—optional
 - To reuse this action within multiple jobs
 - To change the schedule for a job without having to re-create the PL/SQL block
2. Create and use a schedule.
3. Create and submit a job.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Your Basic Work Flow

You can perform all steps either in the graphical environment of Enterprise Manager or by using the DBMS_SCHEDULER PL/SQL package via the command line.

1. Creating a Program

Use the CREATE_PROGRAM procedure to create a program. Creating a program is an optional part of using the Scheduler. You can also encode the action to be performed within an anonymous PL/SQL block in the CREATE_JOB procedure. By creating the program separately, you can define the action once, and then reuse this action within multiple jobs. This enables you to change the schedule for a job without having to re-create the PL/SQL block.

A program is created in a disabled state by default (unless the enabled parameter is set to TRUE). A disabled program cannot be executed by a job until it is enabled. You can specify that a program should be created in the enabled state by specifying a value of TRUE for enabled.

2. Creating and Using Schedules

The schedule for a job can be a predefined schedule (created with the CREATE_SCHEDULE procedure) or defined as part of the job creation.

Your Basic Work Flow (continued)

Creating and Using Schedules (continued)

The schedule specifies attributes about when the job is run, such as:

- A start time, which defines when the job is picked for execution and an end time, which specifies the time after which the job is no longer valid and is not scheduled any more
- An expression specifying a repeating interval for the job
- A complex schedule created by combining existing schedules
- A condition or change in state, called an event, that must be met before the job is started

By using a schedule (instead of specifying the execution times for a job within the job definition), you can manage the scheduled execution of multiple jobs without having to update multiple job definitions. If a schedule is modified, each job that uses that schedule automatically uses the new schedule.

3. Creating and Running a Job

A job is a combination of a schedule and a description of what to do, along with any additional arguments that are required by the job. There are many attributes that you can set for a job. Attributes control how the job executes.

Quiz

Select the statements that are true about the Scheduler:

1. Creating a program is a mandatory part of using the Scheduler.
2. When the job action is in a program (rather than directly in the job), you can change the job schedule without having to re-create the PL/SQL block.
3. Creating a job is an optional part of using the Scheduler.
4. Each job must have a schedule. It can be a predefined one or defined as part of the job creation.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Answer: 2, 4

Persistent Lightweight Jobs

Persistent lightweight jobs:

- Reduce the overhead and time required to start a job
- Have a small footprint on disk for the job metadata and for storing run-time data
- Are created from a job template (in the command line)

```
BEGIN
DBMS_SCHEDULER.CREATE_JOB (
  job_name          => 'my_lightweight_job2',
  program_name      => 'MY_PROG',
  schedule_name     => 'MY_SCHED',
  job_style         => 'LIGHTWEIGHT');
END;
/
```

Choosing the right job:

- Use regular jobs for maximum flexibility.
- Use persistent lightweight jobs when you need to create a large number of jobs in a very short time.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Lightweight Jobs

A lightweight job:

- Is for customers who need to create hundreds of jobs a second. With regular jobs, each job creates a database object describing the job, modifying several tables, and creating redo in the process. The overhead associated with this type of job need is substantial. In the Oracle Database Scheduler, there is a *persistent lightweight job*. The goal of a lightweight job is to reduce the overhead and time required to start a job. A minimal amount of metadata is created for the job. This reduces the time required and redo created when the job starts.
- Has a small footprint on disk for the job metadata and for storing run-time data. The small footprint on disk also makes load-balancing possible in RAC environments.
- Is always created from a job template. The job template must be a stored procedure or a program. The stored procedure holds all the information needed for the job, including privileges. A few job attributes can be specified: job arguments and schedule.
- Must be created in command line. The JOB_STYLE argument is not available in EM.

In the example, MY_PROG is the job template and the schedule is applied from a named schedule.

Using a Time-Based or Event-Based Schedule

ORACLE Enterprise Manager 11g
Database Control

Database Instance: orcl > Scheduler Jobs > Create Job

Logged in As SYS

Show SQL Cancel OK

General Schedule Options

Schedule Type: Standard
Standard
Use Pre-defined Schedule
Standard Using PL/SQL for repeated interval
Use Pre-defined Window
Event
Calendar

Repeat: Do Not Repeat
Do Not Repeat
By Seconds
By Minutes
By Hours
By Days
By Weeks
By Months
By Years

Start: Immediate
Later

Date: Jul 13, 2009
(example: Jul 13, 2009)

Time: 6:20:00 AM PM

General Schedule Options

Schedule

Time
-Calendaring expression
-Date-time expression

Event

Copyright © 2009, Oracle. All rights reserved.

Using a Time-Based or Event-Based Schedule

To specify a time-based schedule for a job, you can specify either a calendaring expression or a date-time expression. When using a calendaring expression, the next start time for a job is calculated using the repeat interval and the start date of the job. When using date-time expressions, the specified expression determines the next time that the job should run. If no repeat interval is specified, the job runs only once on the specified start date.

If a job uses an event-based schedule, the job runs when the event is raised. At a high level, an event can be viewed as a change in state. An event occurs when a Boolean condition changes its state from FALSE to TRUE, or TRUE to FALSE.


The Scheduler uses Oracle Streams Advanced Queuing (AQ) to raise and consume events.

Note: The Scheduler does not guarantee that a job executes on the exact time because the system may be overloaded and thus resources may be unavailable.

Creating a Time-Based Job

Example: Create a job that calls a backup script every night at 11:00, starting tonight.

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB(
    job_name=>'HR.DO_BACKUP',
    job_type => 'EXECUTABLE',
    job_action =>
      '/home/usr/dba/rman/nightly_incr.sh',
    start_date=> SYSDATE,
    repeat_interval=>'FREQ=DAILY;BYHOUR=23',
                      /* next night at 11:00 PM */
    comments => 'Nightly incremental backups');
END;
/
```



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating a Time-Based Job

Use the `CREATE_JOB` procedure of the `DBMS_SCHEDULER` package to create a job. Jobs are created disabled by default and they become active and scheduled only when they are explicitly enabled. All job names are of the form: [schema .]name.

You should use `SYSTIMESTAMP` and specify a time zone so that when the time changes because of daylight saving time, your job adjusts its execution time automatically.

By default, a job is created in the current schema. You can create a job in another schema by specifying the name of the schema, as shown in the example in the slide. The job owner is the user in whose schema the job is created, whereas the job creator is the user who created the job. Jobs are executed with the privileges of the job owner. The national language support (NLS) environment of the job when it runs is the same as that present at the time the job was created. The `job_type` parameter indicates the type of task to be performed by the job. The possible values are:

- **PLSQL_BLOCK:** An anonymous PL/SQL block
- **STORED_PROCEDURE:** A named PL/SQL, Java, or external procedure
- **EXECUTABLE:** A command that can be executed from the operating system (OS) command line

Creating a Time-Based Job (continued)

The `job_action` parameter can be the name of the procedure to run, the name of a script or operating system command, or an anonymous PL/SQL code block, depending on the value of the `job_type` parameter.

In the example in the slide, `job_type` is specified as `EXECUTABLE` and `job_action` is the full OS-dependent path of the desired external executable plus optionally any command-line arguments.

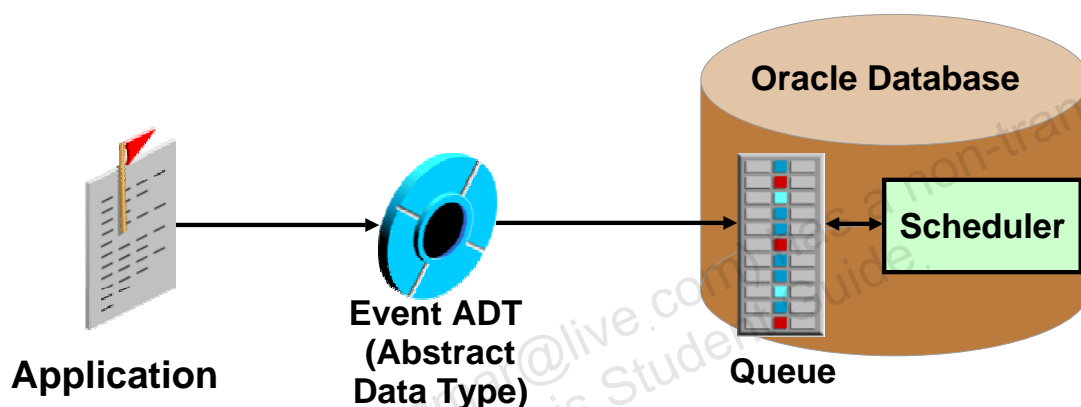
An external job is a job that runs outside the database. All external jobs run as a low-privileged guest user, as has been determined by the database administrator while configuring external job support. Because the executable is run as a low-privileged guest account, you should verify that it has access to necessary files and resources. Most, but not all, platforms support external jobs. For platforms that do not support external jobs, creating or setting the attribute of a job or a program to type `EXECUTABLE` returns an error.

Refer to your Oracle database platform-specific documentation for more information about configuring the environment to run external programs with the Scheduler.

Creating an Event-Based Schedule

To create an event-based job, you must set:

- A queue specification (where your application enqueues messages to start a job)
- An event condition (same syntax as an Oracle Streams AQ rule condition) that if TRUE starts the job



Copyright © 2009, Oracle. All rights reserved.

Creating an Event-Based Schedule


Jobs can be triggered based on events. An application can notify the Scheduler to start a job by enqueueing a message onto an Oracle Streams queue. A job started in this way is referred to as an event-based job. To create an event-based job, you must set the following two additional attributes with the `CREATE_JOB` procedure:

- **queue_spec:** A queue specification that includes the name of the queue where your application enqueues messages to raise job start events, or in the case of a secure queue, the `<queue_name>`, `<agent_name>` pair
- **event_condition:** A conditional expression based on message properties that must evaluate to TRUE for the message to start the job. You can include user data properties in the expression, provided that the message payload is a user-defined object type, and that you prefix object attributes in the expression with `tab.user_data`.

You can either specify `queue_spec` and `event_condition` as in-line job attributes, or create an event-based schedule with these attributes and then create a job that references this schedule.

Creating Event-Based Schedules with Enterprise Manager


Schedule

Time Zone 

Schedule Type

Event Parameters

★ Queue Name **SYS.ALERT_QUE**

★ Agent Name 

★ Condition

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating Event-Based Schedules with Enterprise Manager

The Create Schedule page enables you to choose between a standard, time-based schedule and an event-based schedule. If you choose an event-based schedule, then the interface changes and you can specify the queue name, agent name, and event condition, in addition to the other schedule attributes.

Note: The Scheduler runs the event-based job for each occurrence of an event that matches `event_condition`. However, events that occur while the job is already running are ignored; the event gets consumed, but does not trigger another run of the job.

References

- See the *Oracle Streams Advanced Queuing User's Guide and Reference* for information about how to create queues and enqueue messages.
- For more information about Oracle Streams AQ rules and event conditions, see the `DBMS_AQADM.ADD_SUBSCRIBER` procedure in the *Oracle Database PL/SQL Packages and Types Reference 11* manual.

Creating an Event-Based Job

Example: Create a job that runs if a batch load data file arrives on the file system before 9:00 AM.

```
BEGIN
  DBMS_SCHEDULER.CREATE_JOB(
    job_name=>'ADMIN.PERFORM_DATA_LOAD',
    job_type => 'EXECUTABLE',
    job_action => '/loaddir/start_my_load.sh',
    start_date => SYSTIMESTAMP,
    event_condition => 'tab.user_data.object_owner =
event_condition => 'tab.user_data.object_owner =
  'HR' and tab.user_data.object_name = 'DATA.TXT'
  and tab.user_data.event_type = 'FILE_ARRIVAL'
  and tab.user_data.event_timestamp < 9 ',
    queue_spec => 'HR.LOAD_JOB_EVENT_Q');

```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating an Event-Based Job

To specify event information as job attributes, you use an alternate syntax of `CREATE_JOB` that includes the `queue_spec` and `event_condition` attributes. The job can include event information in-line as job attributes or can specify event information by pointing to an event schedule. The example shown in the slide uses an in-line, event-based schedule.

The example in the slide shows a job that is started when a file arrives on the operating system, as long as the file arrives before 9:00 AM. Assume that the message payload is an object with four attributes named `object_owner`, `object_name`, `event_type`, and `event_timestamp`.

The example uses a user-defined event. Therefore, before this job can be started, when the file arrives on the file system, a program or procedure must enqueue the event object type with the proper information into the specified event queue. The `HR.LOAD_JOB_EVENT_Q` queue must be of the same type as the event object type used for notifying the Scheduler of an event occurrence. That is, the `HR.LOAD_JOB_EVENT_Q` queue must be a typed queue where the type has four attributes named `object_owner`, `object_name`, `event_type`, and `event_timestamp`.

For more information about how to create queues and enqueue messages, refer to the *Oracle Streams Advanced Queuing User's Guide and Reference* documentation.

Event-Based Scheduling

Event types:

- User- or application-generated events
- Scheduler-generated events

Events raised by Scheduler jobs:

- | | |
|-----------------|-----------------------|
| • JOB_STARTED | • JOB_SCH_LIM_REACHED |
| • JOB_SUCCEEDED | • JOB_DISABLED |
| • JOB_FAILED | • JOB_CHAIN_STALLED |
| • JOB_BROKEN | • JOB_ALL_EVENTS |
| • JOB_COMPLETED | • JOB_RUN_COMPLETED |
| • JOB_STOPPED | • JOB_OVER_MAX_DUR |

Example of raising an event:

```
DBMS_SCHEDULER.SET_ATTRIBUTE('hr.do_backup',  
    'raise_events', DBMS_SCHEDULER.JOB_FAILED);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Event-Based Scheduling

You can create a job that directly references an event as the means to start the job, instead of assigning a schedule to the job. There are two types of events:

- **User- or application-generated events:** An application can raise an event to be consumed by the Scheduler. The Scheduler reacts to the event by starting a job. An example of such events: a running job completes; a file arrives on the file system; an account within the database is locked; and the inventory reaches a low threshold.
- **Scheduler-generated events:** The Scheduler can raise an event to indicate state changes that occur within the Scheduler itself. For example, the Scheduler can raise an event when a job starts, when a job completes, when a job exceeds its allotted run time, and so on. The consumer of the event is an application that performs some action in response to the event.

You can configure a job so that the Scheduler raises an event when the job's state changes. You do this by setting the `raise_events` job attribute. By default, a job does not raise any state change events until you alter the `raise_events` attribute for a job. To alter this attribute, you must first create the job by using the `CREATE_JOB` procedure and then use the `SET_ATTRIBUTE` procedure to modify the attribute's default value. The example shows that the `hr.do_backup` job is altered, so that it raises an event if the job fails.

Event-Based Scheduling (continued)

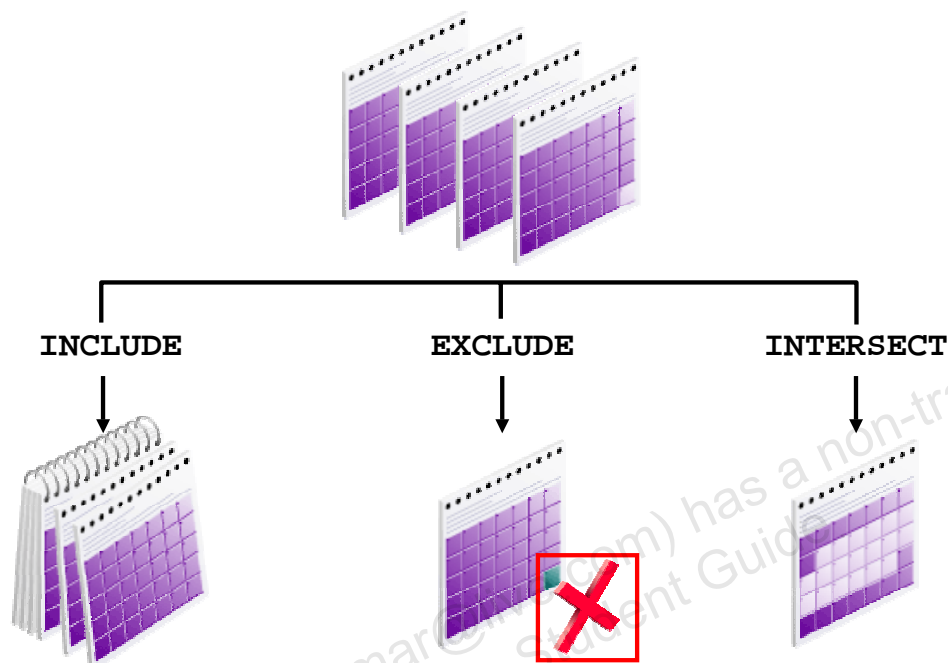
After you enable job state change events for a job, the Scheduler raises these events by enqueueing messages onto the default event queue `SYS.SCHEDULER$_EVENT_QUEUE`.

The default Scheduler event queue is a secure queue. Depending on your application, you may have to configure the queue to enable certain users to perform operations on it. See the *Oracle Streams Concepts and Administration* documentation for information about secure queues.

The default Scheduler event queue is intended primarily for Scheduler-generated events. Oracle does not recommend the use of this queue for user applications, or user-defined events.

Event Type	Description
JOB_STARTED	The job is started.
JOB_SUCCEEDED	The job is successfully completed.
JOB_FAILED	The job failed, either by raising an error or by abnormally terminating.
JOB_BROKEN	The job is disabled and changed to the BROKEN state, because it exceeded the number of failures defined by the MAX_FAILURES job attribute.
JOB_COMPLETED	The job is completed, because it reached the values set by the MAX_RUNS or END_DATE job attributes.
JOB_STOPPED	The job is stopped by a call to the STOP_JOB procedure.
JOB_SCH_LIM_REACHED	The job's schedule limit is reached. The job is not started, because the delay in starting the job exceeded the value of the SCHEDULE_LIMIT job attribute.
JOB_DISABLED	The job is disabled by the scheduler or by a call to the SET_ATTRIBUTE procedure.
JOB_CHAIN_STALLED	A job running a chain is put into the CHAIN_STALLED state. A running chain becomes stalled if there are no steps running or scheduled to run and the chain EVALUATION_INTERVAL is set to NULL. The chain waits for manual intervention.
JOB_ALL_EVENTS	JOB_ALL_EVENTS is not an event, but a constant, that provides an easy way for you to enable all events.
JOB_OVER_MAX_DUR	The job has run over the maximum time it was set to be allowed to run.
JOB_RUN_COMPLETED	A job run is completed. It either failed, succeeded, or is stopped.

Creating Complex Schedules



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating Complex Schedules

A schedule is an object in the database. When you create schedules, they are automatically saved. You can use combinations of schedules to create more complex schedules. By combining schedules, you can add specific dates to or exclude specific dates from a calendaring expression.

You can use the following options when defining the repeat interval for a schedule:

- **INCLUDE:** Adds a list of dates to the calendaring expression results
- **EXCLUDE:** Removes a list of dates from the calendaring expression results
- **INTERSECT:** Uses only the dates that are common to two or more schedules

When creating schedules to be used in combinations, you can code the list of dates by including hard-coded dates of the form [YYYY]MMDD or by including named schedules created with the `CREATE_SCHEDULE` procedure. For example, you can specify a list of dates by using the following values for the repeat interval of a schedule:

```
0115,0315,0325,0615,quarter_end_dates,1215
```

This string represents the dates January 15, March 15, March 25, June 15, December 15, and the list of dates specified by the `QUARTER_END_DATES` schedule.

If you do not specify the optional year component for hard-coded dates in your schedule, the dates are included for every year.

Quiz

Select the statements that are true about persistent lightweight jobs:

1. Persistent lightweight jobs have a small footprint on disk for the job metadata and also for storing run-time data.
2. Use persistent lightweight jobs for maximum flexibility.
3. Persistent lightweight jobs are created from a job template.
4. Persistent lightweight jobs can be created in Enterprise Manager and via command line.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Answer: 1, 3

Using Email Notification

- Email notifications for change of job state
- Triggered by job state events
- Multiple notifications, multiple recipients
- *_SCHEDULER_NOTIFICATIONS views

Using Scheduler Email Notification:

1. Specify the address of the SMTP server you will use to send email messages:

```
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE  
( 'email_server', 'host[:port]' );
```

2. Optionally, set a default sender email address:

```
DBMS_SCHEDULER.SET_SCHEDULER_ATTRIBUTE  
( 'email_sender', 'valid email address' );
```

3. Add email notifications for a specified job. *(continued)*

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Using Email Notification

The job email notification feature enables you to add email notifications to existing jobs so that events of interest that happen to the job are emailed to specified email addresses. For each job, you can add notifications for different events. You can send the email notification to more than one recipient.

To enable the email notification feature, you must:

1. Set the email_server Scheduler attribute.
2. Optionally, you can use the email_sender Scheduler attribute to specify a default sender email address for the email notifications.
3. After creating a job, execute the DBMS_SCHEDULER.ADD_JOB_EMAIL_NOTIFICATION procedure to add one or more notifications for the job.

The data dictionary supports email notifications with the *_SCHEDULER_NOTIFICATIONS views.

Adding and Removing Email Notifications

```
DBMS_SCHEDULER.ADD_JOB_EMAIL_NOTIFICATION (
  job_name          IN VARCHAR2,
  recipients         IN VARCHAR2,
  sender            IN VARCHAR2 DEFAULT NULL,
  subject           IN VARCHAR2
    DEFAULT dbms_scheduler.default_notification_subject,
  body              IN VARCHAR2
    DEFAULT dbms_scheduler.default_notification_body,
  events            IN VARCHAR2
    DEFAULT 'JOB_FAILED,JOB_BROKEN,JOB_SCH_LIM_REACHED,
            JOB_CHAIN_STALLED,JOB_OVER_MAX_DUR',
  filter_condition  IN VARCHAR2 DEFAULT NULL);
```

Comma-separated list of email addresses

Mandatory comma-separated list

```
DBMS_SCHEDULER.REMOVE_JOB_EMAIL_NOTIFICATION (
  job_name          IN VARCHAR2,
  recipients         IN VARCHAR2 DEFAULT NULL,
  events            IN VARCHAR2 DEFAULT NULL);
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Adding and Removing Email Notifications

Add one or more job email notifications with the `DBMS_SCHEDULER.ADD_JOB_EMAIL_NOTIFICATION` procedure. Email messages will be sent to the specified recipient addresses whenever any of the listed events are generated by the job. The job is automatically modified to raise these events. If a filter condition is specified, only events that match the specification in `FILTER_CONDITION` will generate an email message.

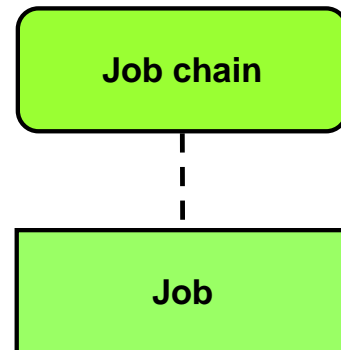
This procedure will fail if the `EMAIL_SERVER` scheduler attribute is not set or if the specified job does not exist. The user calling this procedure must be the owner of the job, have the `CREATE ANY JOB` system privilege, or have been granted the `ALTER` privilege for the job.

- The **subject** of notification emails can contain the following variables for which values will be substituted: `%job_owner%`, `%job_name%`, `%event_type%`, `%event_timestamp%`, `%log_id%`, `%error_code%`, `%error_message%`, `%run_count%`, `%failure_count%`, `%retry_count%`, `%job_subname%`, `%job_class_name%`.
- The notification email message **body** can contain any of the variables that are valid in the `SUBJECT`.
- The comma-separated list of **events** cannot be `NULL`. Refer to the list of events for the `RAISE_EVENTS` attribute of `JOBS` for valid events.
- If **filter_condition** is `NULL` (the default), all occurrences of the specified events will be emailed to all specified recipient addresses.

Remove one or more email notifications for a specified job with the `DBMS_SCHEDULER.REMOVE_JOB_EMAIL_NOTIFICATION` procedure.

Creating Job Chains

1. Create a chain object.
2. Define chain steps.
3. Define chain rules.
4. Starting the chain:
 - Enable the chain.
 - Create a job that points to the chain.



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating Job Chains

A chain is a named series of programs that are linked together for a combined objective. This is known as “dependency scheduling.” An example of a chain may be the following:

Run program A and then program B, but only run program C if programs A and B complete successfully, otherwise run program D.

Each position within a chain of interdependent programs is referred to as a step. Typically, after an initial set of chain steps has started, the execution of successive steps depends on the completion of one or more previous steps. To create and use a chain, you complete the following steps in order. All procedures mentioned are part of the DBMS_SCHEDULER package, unless noted otherwise.

1. **Create a chain** by using the CREATE_CHAIN procedure. The chain name can be optionally qualified with a schema name (for example, *myschema.myname*).
2. **Define** (one or more) **chain steps**. Defining a step gives it a name and specifies what happens during the step. Each step can point to one of the following:
 - A program
 - Another chain (a nested chain)
 - An event

You define a step that points to a program or nested chain by calling the DEFINE_CHAIN_STEP procedure.

Creating Job Chains (continued)

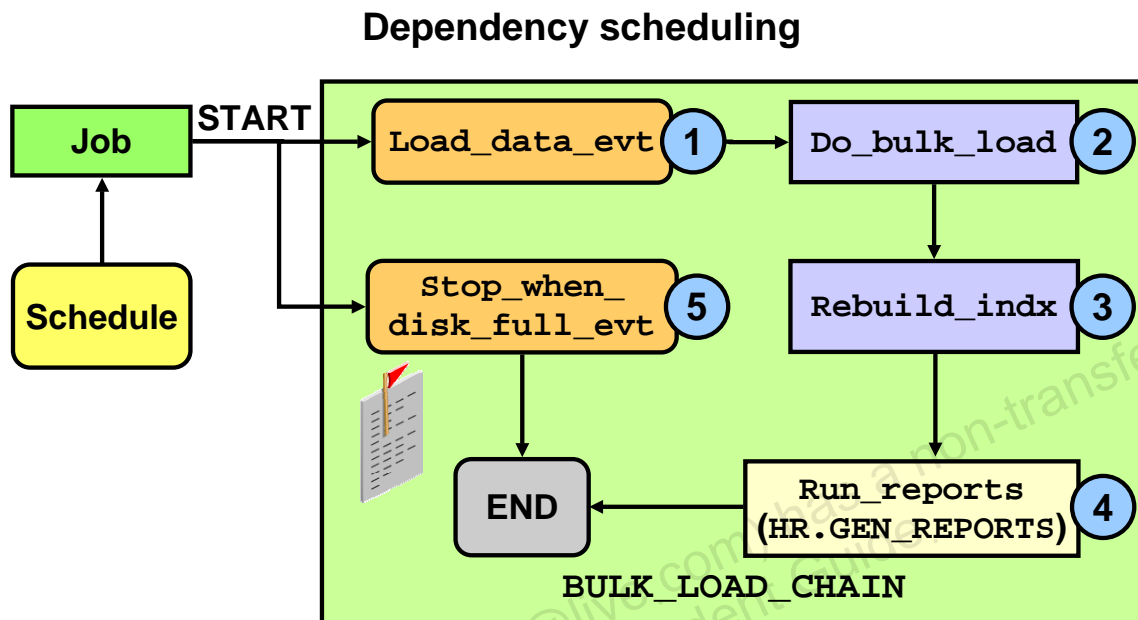
To define a step that waits for an event to occur, you use the `DEFINE_CHAIN_EVENT_STEP` procedure. Procedure arguments can point to an event schedule or can include an in-line queue specification and event condition. A step that points to an event waits until the specified event is raised. If the event occurs, the step completes successfully.

3. After creating the chain object, you **define chain rules**. Chain rules define when steps run, and define dependencies between steps. Each rule has a *condition* and an *action*:
 - If the condition evaluates to `TRUE`, the action is performed. The condition can contain any syntax that is valid in a `SQL WHERE` clause. Conditions are usually based on the outcome of one or more previous steps. For example, you may want one step to run if the two previous steps succeeded, and another to run if either of the two previous steps failed.
 - The action specifies what is to be done as a result of the rule being triggered. A typical action is to run a specified step. Possible actions include starting or stopping a step. You can also choose to end the execution of the job chain, returning either a value or a step name and error code.

All rules added to a chain work together to define the overall behavior of the chain. When the job starts and at the end of each step, all rules are evaluated to see what action or actions occur next. You add a rule to a chain with the `DEFINE_CHAIN_RULE` procedure. You call this procedure once for each rule that you want to add to the chain.

4. **Starting the chain** involves two actions:
 - Enable a chain with the `ENABLE` procedure. (A chain is always created disabled, so you can add steps and rules to the chain before it is executed by any job.) Enabling an already enabled chain does not return an error.
 - To run a chain, you must create a job of type `'CHAIN'`. The job action must refer to the chain name. You can use either event-based or time-based schedules for this job.

Example of a Chain



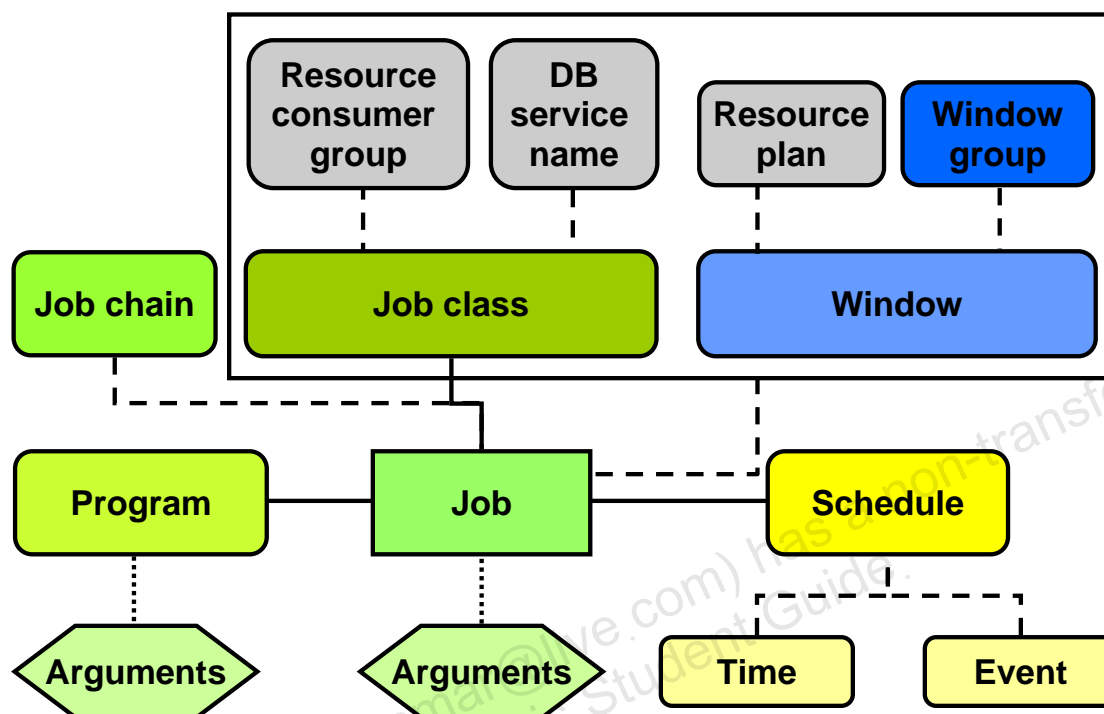
ORACLE

Copyright © 2009, Oracle. All rights reserved.

Example of a Chain

As an example of a chain, consider all the tasks and conditions that occur during a bulk data load. First, you must have data to load. Then load the data, observing the file system to make sure that you do not run out of space during the load. After the data load completes, you need to rebuild the indexes defined on the updated tables. Then you run reports against the newly loaded data. The slide shows an example of dependency scheduling.

Advanced Scheduler Concepts



Copyright © 2009, Oracle. All rights reserved.

Advanced Scheduler Concepts

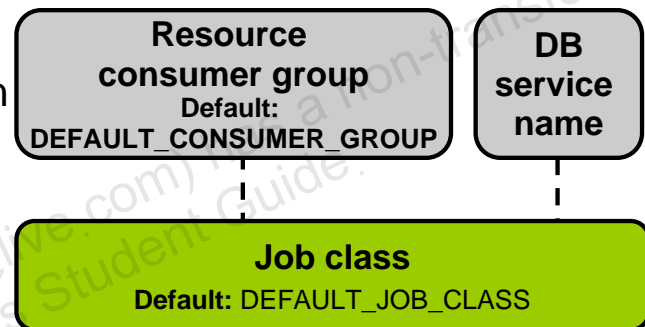
Using advanced Scheduler features, you can exercise more control over various aspects of scheduling, such as job windows and prioritizing jobs. These advanced features are summarized below, and are discussed in detail in the following slides.

- A **window** is represented by an interval of time with a well-defined beginning and end, and is used to activate different resource plans at different times. This allows you to change resource allocation during a time period such as time of day or time of the sales year.
- A **window group** represents a list of windows, and allows for easier management of windows. You can use a window or window group as the schedule for a job to ensure that the job runs only when a window and its associated resource plans are active.
- A **job class** defines a category of jobs that share common resource usage requirements and other characteristics. A job class groups jobs into larger entities.
- A **resource consumer group** associated with the job class determines the resources that are allocated to the jobs in the job class.
- A **resource plan** enables users to prioritize resources (most notably CPU) among resource consumer groups.

Note: The gray objects are not Scheduler objects.

Job Classes

- Assign the same set of attribute values to member jobs
- Are created by the `CREATE_JOB_CLASS` procedure
- Specify jobs in a job class (with the `SET_ATTRIBUTE` procedure)
- Belong to the `SYS` schema
- Set resource allocation for member jobs
- Set the service attribute to a desired database service name
- Group jobs for prioritization



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Job Classes

Job classes are policies for their participating member jobs. Each job class specifies a set of attributes, such as the logging level. When you assign a job to a job class, the job inherits those attributes. For example, you can specify the same policy for purging log entries for all payroll jobs.

- You can use the `CREATE_JOB_CLASS` procedure to create a job class. A class always belongs to the `sys` schema. To create a class, you must have the `MANAGE_SCHEDULER` privilege. There is a default job class named `DEFAULT_JOB_CLASS` that is created with the database
- After a job class has been created, you can specify jobs as members of this job class when you create the jobs, or after the jobs are created, by using the `SET_ATTRIBUTE` procedure of the `DBMS_SCHEDULER` package. If a job is not associated with a job class, the job belongs to this default job class.
- Set the service attribute of a job class to a desired database service name. This determines the instances in a Real Application Clusters environment that run the member jobs, and optionally the system resources that are assigned to the member jobs.

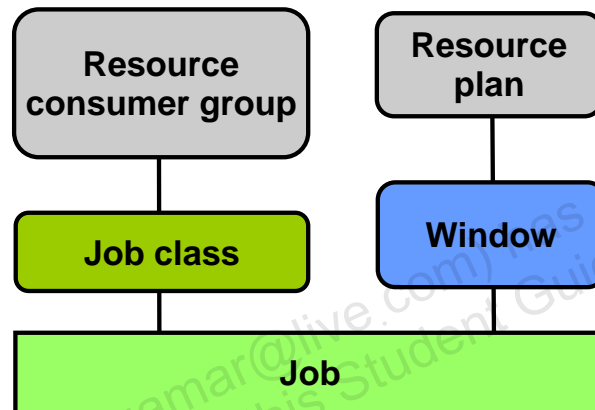
Job Classes (continued)

- Set resource allocation for member jobs. Job classes provide the link between the Database Resource Manager and the Scheduler because each job class can specify a resource consumer group as an attribute. Member jobs then belong to the specified consumer group and are assigned resources according to settings in the current resource plan. Alternatively, you can leave the `resource_consumer_group` attribute as NULL and set the service attribute of a job class to a desired database service name. That service can in turn be mapped to a resource consumer group. If both the `resource_consumer_group` and service attributes are set, and the designated service maps to a resource consumer group, the resource consumer group named in the `resource_consumer_group` attribute takes precedence. If a resource consumer group is not specified when a job class is created, the job class maps to the `DEFAULT_CONSUMER_GROUP` resource consumer group. Jobs in the default job class or in a job class associated with the default resource consumer group may not be allocated enough resources to complete their tasks when the Resource Manager is enabled.
- Group jobs for prioritization. Within the same job class, you can assign priority values of 1–5 to individual jobs so that if two jobs in the class are scheduled to start at the same time, the one with the higher priority takes precedence. This ensures that you do not have a less important job preventing the timely completion of a more important one. If two jobs have the same assigned priority value, the job with the earlier start date takes precedence. If no priority is assigned to a job, its priority defaults to 3.

Windows

Scheduler windows:

- Can start jobs or change resource allocation among jobs for various time periods
- One active at a time
- Created with the `CREATE_WINDOW` procedure



ORACLE

Copyright © 2009, Oracle. All rights reserved.

Windows

The priority of jobs can change over a period of time. For example, you might want to allocate a high percentage of the database resources to data warehouse loading jobs at night and allocate a higher percentage of the resources during the day to the application jobs. To accomplish this, you can change the database resource plan by using a Scheduler window.

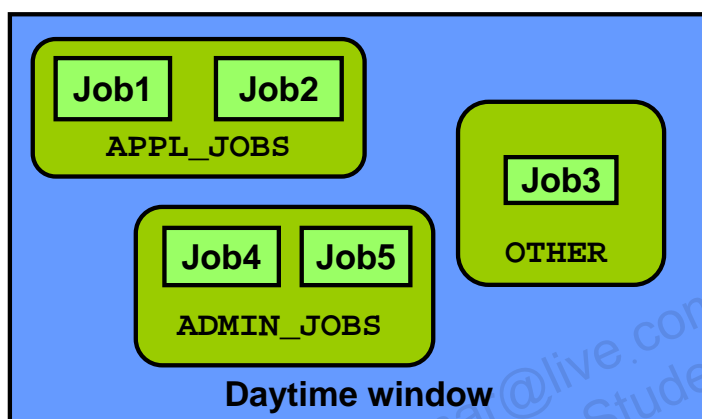
- Scheduler windows can automatically start jobs or change the resource allocation among jobs during various time periods of the day, week, and so on. A window is represented by an interval of time with a well-defined beginning and end, such as “from 12:00 AM to 6:00 AM.”
- Only one window can be in effect at any given time.
- You can create a window with the `CREATE_WINDOW` procedure.

Scheduler windows work with job classes to control resource allocation. Each window specifies the resource plan to activate when the window opens (becomes active), and each job class specifies a resource consumer group or specifies a database service, which can map to a consumer group. A job that runs within a window, therefore, has resources allocated to it according to the consumer group of its job class and the resource plan of the window (as shown in the graphic in this slide).

Prioritizing Jobs Within a Window

Prioritizing jobs:

- At the class level (via resource plans)
- At the job level (with the job priority attribute)
- Not guaranteed for jobs in different job classes



Job	Priority
Job1	1
Job2	2
Job3	3
Job4	5
Job5	2

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Prioritizing Jobs Within a Window

When creating multiple jobs in a database, you need a way to align the job processing with your business requirements and specify which jobs have the highest priority. For a particular window, you may have several classes of jobs running, each with its own priority.

There are two levels at which jobs can be prioritized: at the class level and at the job level.

- The first prioritization is at the class level, using resource plans. Prioritization among jobs of different classes is done purely on a class resource allocation basis.
- The second prioritization is within the class, with the job priority attribute of the job.

Prioritization levels are relevant only when two jobs within the same class are supposed to start at the same time. The job with the higher priority starts first.

Prioritization is not guaranteed for jobs in different job classes. For example, a high-priority job in the APPL_JOBS job class might not get started before a low-priority job in the ADMIN_JOBS job class, even if they share the same schedule. If the APPL_JOBS job class has a lower level of resource available, the high-priority job in that class has to wait for resources to become available, even if there are resources available to lower-priority jobs in a different job class.

Creating a Job Array

1. Declare variables of types `sys.job` and `sys.job_array`:

```
DECLARE
  newjob sys.job;
  newjobarr sys.job_array;
```

2. Initialize the job array:

```
BEGIN
  newjobarr := SYS.JOB_ARRAY();
```

3. Size the job array to hold the number of jobs needed:

```
newjobarr.EXTEND(100);
```

(... continued)

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating a Job Array

A more efficient way to create a set of jobs is the use of a job array. This also applies to lightweight jobs. In the example in the slide, 100 job specifications are created in a job array and submitted to the job queue in a single transaction. Notice that for a lightweight job there is a very limited amount of information needed. In the example, the `start_time` parameter defaults to `NULL`, so the job is scheduled to start immediately.

1. Declare the variable to hold a job definition and a job array variable.
2. Initialize the job array using the `SYS.JOB_ARRAY` constructor. This creates a place for one job in the array.
3. Set the size of the array to the number of expected jobs.
4. Create each job and place it in the array. In the slide example, the only difference is the name of the job. The `start_time` variable of the job is omitted and it defaults to `NULL`, indicating that the job will run immediately.
5. Use the `CREATE_JOBS` procedure to submit all the jobs in the array as one transaction.

Note: If the array is very small, the performance will not be significantly better than submitting a single job.

Creating a Job Array

4. Place jobs in the job array:

```
FOR i IN 1..100 LOOP
    newjob := SYS.JOB(job_name => 'LWTJK' || to_char(i),
                     job_style => 'LIGHTWEIGHT',
                     job_template => 'MY_PROG',
                     enabled => TRUE );
    newjobarr(i) := newjob;
END LOOP;
```

5. Submit the job array as one transaction:

```
DBMS_SCHEDULER.CREATE_JOBS(newjobarr,
                           'TRANSACTIONAL');
```

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating a Job Array (continued)

The full code of this example is:

```
DECLARE
    newjob sys.job;
    newjobarr sys.job_array;
BEGIN
    -- Create an array of JOB object types
    newjobarr := sys.job_array();
    -- Allocate sufficient space in the array
    newjobarr.extend(100);
    -- Add definitions for jobs
    FOR i IN 1..100 LOOP
        -- Create a JOB object type
        newjob := sys.job(job_name => 'LWTJK' || to_char(i),
                         job_style => 'LIGHTWEIGHT',
                         job_template => 'PROG_1',
                         enabled => TRUE );

        -- Add job to the array
        newjobarr(i) := newjob;
    END LOOP;
    -- Call CREATE_JOBS to create jobs in one transaction
    DBMS_SCHEDULER.CREATE_JOBS(newjobarr, 'TRANSACTIONAL');
END;
```

/

Quiz

Select the statements that are true about the advanced Scheduler concepts and functionality:

1. Lightweight jobs can be created with a job array.
2. Prioritizing jobs at the class level (via resource plans) and at the job level (with the job priority attribute) are mutually exclusive.
3. Scheduler windows work with job classes to control resource allocation.
4. Job chains are used to implement “dependency scheduling.”

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Answer: 1, 3, 4

Creating a File Watcher and an Event-Based Job

Perform the following tasks:

1. Create a Scheduler credential object and grant `EXECUTE`.
2. Create a file watcher and grant `EXECUTE`.
3. Create a Scheduler program object with a metadata argument that references the event message.
4. Create an event-based job that references the file watcher. (Optionally, enable the job to run for each instance of the file arrival event.)
5. Enable the file watcher, the program, and the job.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating a File Watcher and an Event-Based Job

Perform the following tasks to create a file watcher and create the event-based job that starts when the designated file arrives:

1. Create a Scheduler credential object (a credential) with which to authenticate with the host operating system for access to the file and grant `EXECUTE` on the credential to the schema that owns the event-based job that the file watcher will start.
2. Create a file watcher and grant `EXECUTE` on the file watcher to any schema that owns an event-based job that references the file watcher.
3. Create a Scheduler program object with a metadata argument that references the event message.
 - Define the metadata argument using the `EVENT_MESSAGE` attribute.
 - Create the stored procedure with an argument of the `SYS.SCHEDULER_FILEWATCHER_RESULT` type that the program invokes. The stored procedure must have an argument of the `SYS.SCHEDULER_FILEWATCHER_RESULT` type, which is the data type of the event message. The position of that argument must match the position of the defined metadata argument. The procedure can access attributes of this abstract data type to learn about the arrived file.

Creating a File Watcher and an Event-Based Job (continued)

4. Create an event-based job that references the file watcher. You can use the `DBMS_SCHEDULER.SET_ATTRIBUTE` procedure to enable the job to run for each instance of the file arrival event, even if the job is already processing a previous event. Set the `PARALLEL_INSTANCES` attribute to `TRUE`.

```
BEGIN
  DBMS_SCHEDULER.SET_ATTRIBUTE( ' ', 'PARALLEL_INSTANCES', TRUE );
END;
```

This enables the job to run as a lightweight job so that multiple instances of the job can be started quickly. If `PARALLEL_INSTANCES` is set to the default value of `FALSE`, file watcher events that occur while the event-based job is already processing another will be discarded.

5. Enable the file watcher, the program, and the job.

Enabling File Arrival Events from Remote Systems

Perform the following tasks to enable the raising of file arrival events at remote systems:

1. Set up the database to run remote external jobs.
2. Install, configure, register, and start the Scheduler agent on the first remote system.
3. Repeat step 2 for each additional remote system.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Enabling File Arrival Events from Remote Systems

To receive file arrival events from a remote system, you must install the Scheduler agent on that system, and you must register the agent with the database. The remote system does not need an Oracle Database instance to generate file arrival events.

Refer to the *Oracle Database Administrator's Guide 11g Release 2* for detailed information.

Scheduling Remote Database Jobs

- Create a job that runs stored procedures and anonymous PL/SQL blocks on another database instance on the same host or a remote host.
- The target database can be any release of Oracle Database.
- `DBMS_SCHEDULER.CREATE_DATABASE_DESTINATION` and `DBMS_SCHEDULER.CREATE_CREDENTIAL` can be used for remote database jobs.
- Jobs with job types of `PLSQL_BLOCK` and `STORED_PROCEDURE` can be the subject of `SET_ATTRIBUTE` calls for the `DESTINATION` and `CREDENTIAL` attributes.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Scheduling Remote Database Jobs

You can now create a job that runs stored procedures and anonymous PL/SQL blocks on another database instance on the same host or a remote host. The target database can be any release of Oracle Database.

There are no new procedures to support remote database jobs, rather there have been changes to existing `DBMS_SCHEDULER` procedures to support this functionality. Additional information is provided over the next few pages.

Creating Remote Database Jobs

Perform the following tasks to create a remote job:

1. Set up the originating database for remote jobs.
2. Create the job by using `DBMS_SCHEDULER.CREATE_JOB`.
3. Create a credential by using `DBMS_SCHEDULER.CREATE_CREDENTIAL`.
4. Set the job `CREDENTIAL_NAME` attribute by using `DBMS_SCHEDULER.SET_ATTRIBUTE`.
5. Set the job `DESTINATION` attribute by using `DBMS_SCHEDULER.SET_ATTRIBUTE`.
6. Enable the job by using `DBMS_SCHEDULER.ENABLE`.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Creating Remote Database Jobs

You can perform the tasks listed in the slide to create a remote database job.

To set up the originating database for remote jobs, perform the following steps:

1. Verify that XML DB is installed.
2. Enable HTTP connections to the database.
3. Execute the `prvtrsch.plb` script.
4. Set a registration password for the Scheduler agents.

```
BEGIN
  DBMS_XDB.SETHTTPPORT(port);
END;
```

```
BEGIN
  DBMS_SCHEDULER.SET_AGENT_REGISTRATION_PASS('password');
END;
```

Refer to the *Oracle Database Administrator's Guide 11g Release 2* for a detailed example.

Scheduling Multiple Destination Jobs

- This enables you to specify several targets on which your jobs should execute.
- It provides the ability to monitor and control the jobs from the database on which they were created.
- While running, a multiple-destination job is viewed as a collection of jobs, which are near-identical copies of each other.
- All jobs will execute based on the time zone that is specified in the start date of the job or will use the time zone of the source database.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Scheduling Multiple Destination Jobs

The multiple destination jobs feature enables you to specify multiple targets on which the jobs should execute. You have the ability to monitor and control the jobs from the database at which you created them. The following capabilities are included:

- Specifying several databases or machines on which a job must execute
- Modifying a job scheduled on multiple targets as a single entity
- Stopping or dropping jobs running on one or more remote targets
- Finding out the status of job instances on all of a job's targets

Note that in the initial release of this feature, all destinations will run based on the time zone that is specified in the start date of the job or will default to the time zone of the source database.

Viewing Scheduler Meta Data

Major Scheduler management views, displaying:

- *_SCHEDULER_JOBS: All jobs, enabled and disabled
- *_SCHEDULER_SCHEDULES: All schedules
- *_SCHEDULER_PROGRAMS: All programs
- *_SCHEDULER_RUNNING_JOBS: Active job states
- *_SCHEDULER_JOB_LOG: All job state changes
- *_SCHEDULER_JOB_RUN_DETAILS: All completed job runs

```
SELECT job_name, status, error#, run_duration
FROM USER_SCHEDULER_JOB_RUN_DETAILS;
```

JOB_NAME	STATUS	ERROR#	RUN_DURATION
-----	-----	-----	-----
GATHER_STATS_JOB	SUCCESS	0	+000 00:08:20
PART_EXCHANGE_JOB	FAILURE	6576	+000 00:00:00

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Viewing Scheduler Meta Data

The job table is a container for all the jobs, with one table per database. The job table stores information for all jobs such as the owner name or the level of logging. You can find this information in the *_SCHEDULER_JOBS views.

Jobs are database objects, and can, therefore, accumulate and take up too much space. To avoid this, job objects are automatically dropped by default after completion. This behavior is controlled by the auto_drop job attribute.

There are many views available to the DBA and privileged users that provide essential operating information regarding the scheduler, jobs, schedules, windows, and so on. These views include:

- *_SCHEDULER_PROGRAM_ARGS: Shows all arguments defined for all programs as well as the default values if they exist
- *_SCHEDULER_JOBS: Shows all jobs, enabled as well as disabled
- *_SCHEDULER_JOB_RUN_DETAILS show all completed (failed or successful) job runs. It has a row for each job instance. Each row contains information about the job execution for that instance. The ERROR# is the number of the first error encountered
- *_SCHEDULER_GLOBAL_ATTRIBUTE: Shows the current values of Scheduler attributes
- *_SCHEDULER_JOB_ARGS: Shows all set argument values for all jobs
- *_SCHEDULER_JOB_CLASSES: Shows all job classes

Scheduler Data Dictionary Views (continued)

For job chains:

- *_SCHEDULER_RUNNING_CHAINS: Shows all active chains
- *_SCHEDULER_CHAIN_STEPS: Shows all steps for all chains
- *_SCHEDULER_CHAINS: Shows all chains
- *_SCHEDULER_CHAIN_RULES: Shows all rules for all chains

For windows and other advanced objects:

- *_SCHEDULER_WINDOWS show all windows.
- *_SCHEDULER_WINDOW_GROUPS show all window groups.
- *_SCHEDULER_WINDOWGROUP_MEMBERS show the members of all window groups, one row for each group member.
- *_SCHEDULER_JOB_LOG show all state changes made to jobs.
- *_SCHEDULER_CREDENTIALS displays a list of credentials in the database with obfuscated passwords.
- *_SCHEDULER_JOB_ROLES show all jobs by database role.

Lightweight jobs are visible through the same views as regular jobs are:

- *_SCHEDULER_JOBS: Shows all jobs, including the JOB_STYLE= 'LIGHTWEIGHT'
- *_SCHEDULER_JOB_ARGS: Shows all set argument values also for lightweight jobs
- Because lightweight jobs are not database objects, they are not visible through the *_OBJECTS views.

Starting with the Oracle Database 11gR2:

- *_SCHEDULER_NOTIFICATIONS shows which email notifications have been set.
- *_SCHEDULER_FILE_WATCHERS shows file watcher configuration information.

The following views display information about multiple destination jobs:

- *_SCHEDULER_DESTS: Shows all the destinations on which remote jobs can be scheduled. The views contain both the external destinations (for remote external jobs) as well as the database destinations for remote database jobs.
- *_SCHEDULER_EXTERNAL_DESTS: Shows all the agents that have registered with the database and can be used as a destination for remote external jobs.
- *_SCHEDULER_DB_DESTS: Shows all the databases on which you can schedule remote database jobs.
- *_SCHEDULER_GROUPS: Shows the groups in your schema or all groups in the database.
- *_SCHEDULER_GROUP_MEMBERS: Shows the group members in your schema or all group members in the database.
- *_SCHEDULER_JOB_DESTS: Shows the state of a job at a remote database.

Note: In the views listed above, the asterisk at the beginning of a view name can be replaced with DBA, ALL, or USER.

Quiz

Select the statements that are true about the Oracle Scheduler:

1. Creating remote database jobs is a manual task, requiring the use of OS-specific commands.
2. A Scheduler credential is an object with which to authenticate with the host operating system for file access.
3. You can specify several targets on which your jobs should execute and monitor them from the database on which they were created.

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Answer: 2, 3

Summary

In this lesson, you should have learned how to:

- Simplify management tasks by using the Scheduler
- Create a job, program, and schedule
- Monitor job execution
- Use a time-based or event-based schedule for executing Scheduler jobs
- Describe the use of windows, window groups, job classes, and consumer groups
- Use email notification
- Use job chains to perform a series of related tasks
- Describe Scheduler jobs on remote systems
- Use advanced Scheduler concepts to prioritize jobs

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Practice 17 Overview: Automating Tasks with the Scheduler

This practice covers the following topics:

- Creating a job that runs a program outside the database
- Creating a program and a schedule
- Creating a job that uses a program and a schedule
- Create a lightweight job
- Monitoring job runs

ORACLE

Copyright © 2009, Oracle. All rights reserved.

Note

This practice uses both Enterprise Manager Database Control and SQL*Plus.