Enterprise Scheduling With Oracle Database 11*g*

An Oracle White Paper
July 2007

ORACLE

Enterprise Scheduling With Oracle Database 11*g*

INTRODUCTION

Automating the business process is a key factor in reducing IT operating expenses. The need for an effective scheduling solution becomes even more apparent as organizations look to grid computing as a means of increasing the return from their existing resources. Grid computing has the potential to drastically change the economics of computing. By virtualizing compute and data resources and dynamically provisioning resources as required, enterprises can realize tremendous improvements in operational efficiency and resource utilization of their IT infrastructure. Efficient usage of resources in a grid environment requires a powerful and flexible scheduling solution.

An effective scheduling solution must allow organizations to automate the business processes so that they can occur without manual intervention. It must enable customers to manage their jobs on an exception only basis. Users must be able to easily define, schedule, manage and monitor their jobs with minimum development effort. It must provide the ability to manage jobs similarly on all platforms. Finally an effective scheduler should automatically optimize resource utilization across the IT environment and effectively align resource allocation with the business requirements.

Oracle Database 10*g* introduced a new feature called Oracle Scheduler. The Scheduler provides complex enterprise scheduling functionality that enables an organization to easily and effectively manage database maintenance and other routine tasks. The Scheduler enables limited computing resources to be allocated appropriately among competing jobs, thus aligning job processing with your business needs. It leverages the reliability and scalability of the Oracle database to provide a robust environment for running jobs.

BENEFITS

The Oracle Scheduler provides a number of benefits:

- **Easy to use**
  - Minimum development time is required since jobs can be easily defined and scheduled using simple mouse operations.
  - Scheduler objects are modular and can be shared with other users, thus reducing the development time for new jobs.

- A graphical interface makes it easy for users to manipulate existing scheduler objects. Object properties can be modified to create new objects.
- The same operation can be performed on multiple jobs. For example multiple jobs can be stopped in one call.

- **Easy to manage**
  - Jobs can be easily moved from one system to another, for example from a development environment to production, using the EXPORT/IMPORT utility in the database.
  - Exception based management enables administrators to quickly focus on the jobs with errors without having to wade through all the jobs.
  - Jobs can be managed as a group.
  - All scheduler activities can be logged, providing an audit trail of all scheduler activities.
  - Time zone support makes it easy to manage jobs in any time zone.
- **Web based graphical interface** (Enterprise Manager)
  - The Scheduler can be accessed and controlled from anywhere, providing the utmost flexibility.
  - All Scheduler activity can be carried out from the same graphical interface.
  - Jobs can be filtered and sorted for easy viewing.
- **Feature of the database**
  - Existing database knowledge can be leveraged, therefore eliminating the need to learn a new system and syntax.
  - Since the Scheduler is part of the database it is platform independent, therefore jobs can be managed similarly on all platforms.
  - The Scheduler can immediately exploit new database features.
  - The Scheduler inherits all the database features: high security, high availability, and high scalability

ARCHITECTURE

Figure 1, shows the Scheduler architecture.

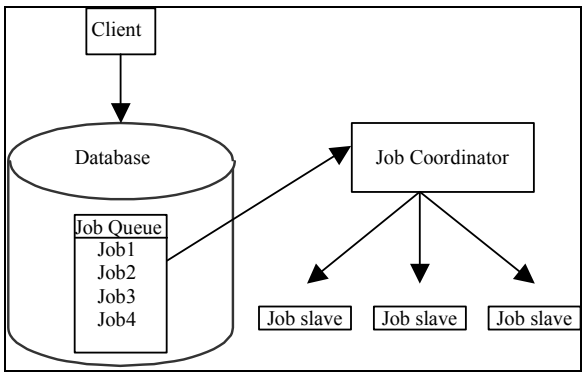


Figure 1: Scheduler Architecture

- Users create and submit jobs to the job queue using either a graphical user interface (GUI) or the DBMS_SCHEDULER package. The database stores the Scheduler object information such as object definition, state change and historical information.
- The job coordinator picks up jobs from the queue and passes them to the job slave.
- The job slave executes the job and updates the job information in the queue. The job coordinator spawns the job slaves. It dynamically controls the slave pool, increasing and reducing its size depending on the number of jobs that need to be executed.
- In a RAC configuration, there is only one queue but each instance of the database will have a job coordinator. The coordinators communicate with each other to exchange information to ensure that the job system remains in sync.

FUNCTIONALITY

The Scheduler can execute OS jobs, such as shell scripts and executables, PL/SQL blocks, and PL/SQL or Java stored procedures. OS jobs can run as any user on or across various platforms, such as Unix, Windows, z/OS and OS/400. The Scheduler has a graphical user interface (GUI) as well as an API. Either interface can be used to manipulate the Scheduler objects. The API is a PL/SQL package in the database and it can be used for scheduling within an application.

Job Definition

To automate tasks users must first define the jobs. Jobs can be defined using the GUI or the DBMS_SCHEDULER package. When using the web based GUI, as shown in Figure 2, users can easily create and edit jobs with simple mouse operations.

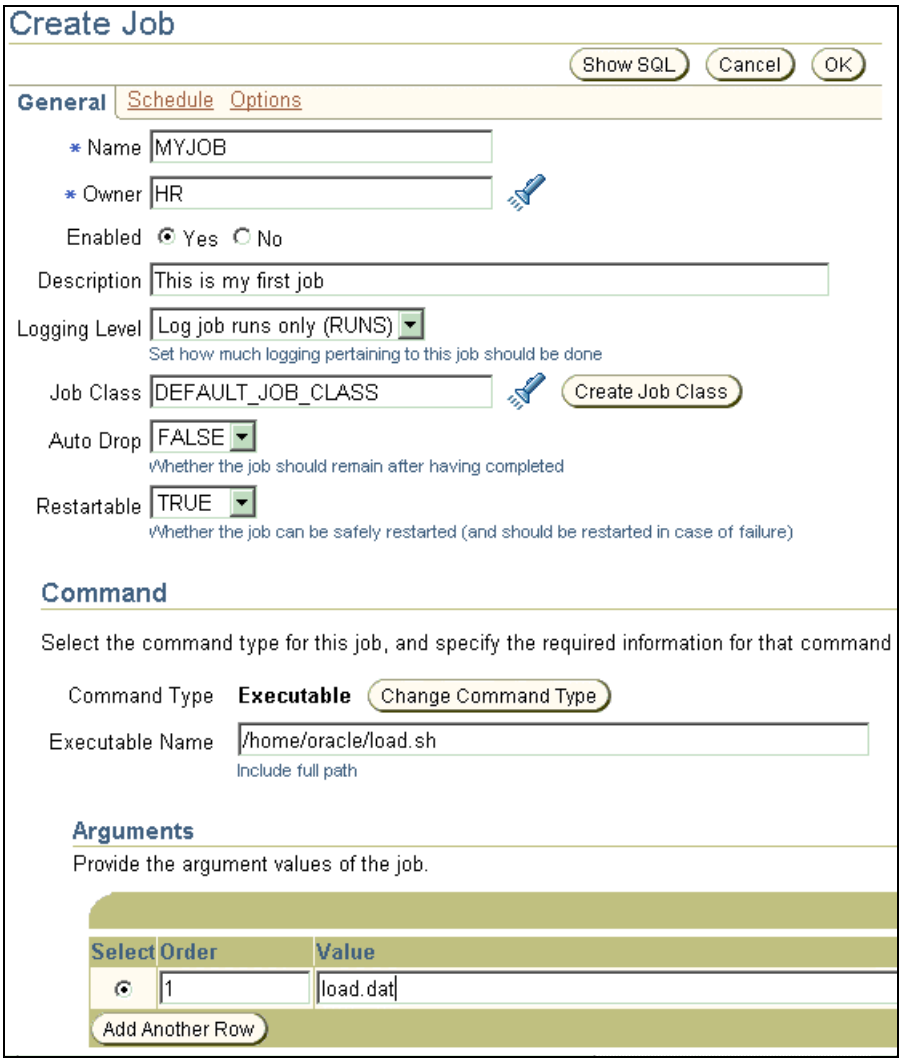


Figure 2: Job Definition Screen

There are two key components that define a job:

- Job command - what the job should do
- Job schedule - when the job should execute

OS jobs have 2 additional attributes:

- Job destination - where the job should run
- Job credential - which OS user the job should run as

The job command can be a single step such as a shell script or PL/SQL block/stored procedure or it can be multi-step, such as a set of inter-dependent steps.

Command arguments are supported in the job definition, which enables users to write generic commands that can be customized by the value of the argument.

These generic commands can be saved as database objects called progams and shared with other users using standard database object privileges. To customize an existing program, users can specify values for the program arguments during the job creation, which will override the default values, specified when the program was created.

Job execution can be based on an event such as the arrival of a file, or using a combination of date and time. The schedule of the job can be explicitly specified in the job. Alternatively a reference to an existing schedule or time window is also supported in the job definition.

OS jobs can run on any node in the enterprise, across various platforms. The destination attribute defines the host and port on which the job should run.

OS jobs run as any OS user on the platform. Credentials specify who the job should run as on the host. They are a combination of username, password and domain (on Windows).

Besides the job command and schedule there are several other attributes that can be specified in the job definition. The job attributes define how the job should be handled during its execution. The key supported attributes are:

- Job_class: This attribute specifies the resources that will be allocated to this job.
- Auto_drop: When set, the job definition will be deleted after the job is executed.
- Restartable: In the case of a failure, either an application error or a database/system crash, the job will automatically be restarted.
- Scheduler_limit: If the delay in starting the job is larger than the interval specified, then the job will not be started. For example, if a job was supposed to start at noon and the schedule limit is set to 60 minutes, the job will not be run if it has not started to run by 1PM.
- Max_runs: Specifies the maximum number of times this job can run. The job will be disabled once max_runs is reached.
- Max_failures: Indicates the maximum number of times this job can fail before it is disabled.
- Max_run_duration: Setting this attribute enables a user to take some action if the job runs longer than the interval specified.

Schedules

Schedules define when a job should execute. Similar to programs, schedules can also be saved as a database object. As an alternative to specifying the start-conditions within a job, a job definition can also point to a saved schedule. This enables users to create a library of commonly used job schedules and share them with other users, thus further reducing development time.

When the start condition of a job is based on a date and time combination it is often referred to as time-based scheduling. Time-based scheduling however, is not ideal for situations where jobs need to be triggered based on real-time events such as the arrival of a file. In such cases, dynamic job scheduling is required and the term event-based scheduling is used to describe it.

Time-Based Scheduling

Figure 3 shows the web based GUI that can be used to create time-based schedules. Users can specify a fixed date and time, a repeating schedule, or a defined rule, for example the last Sunday of every other month.

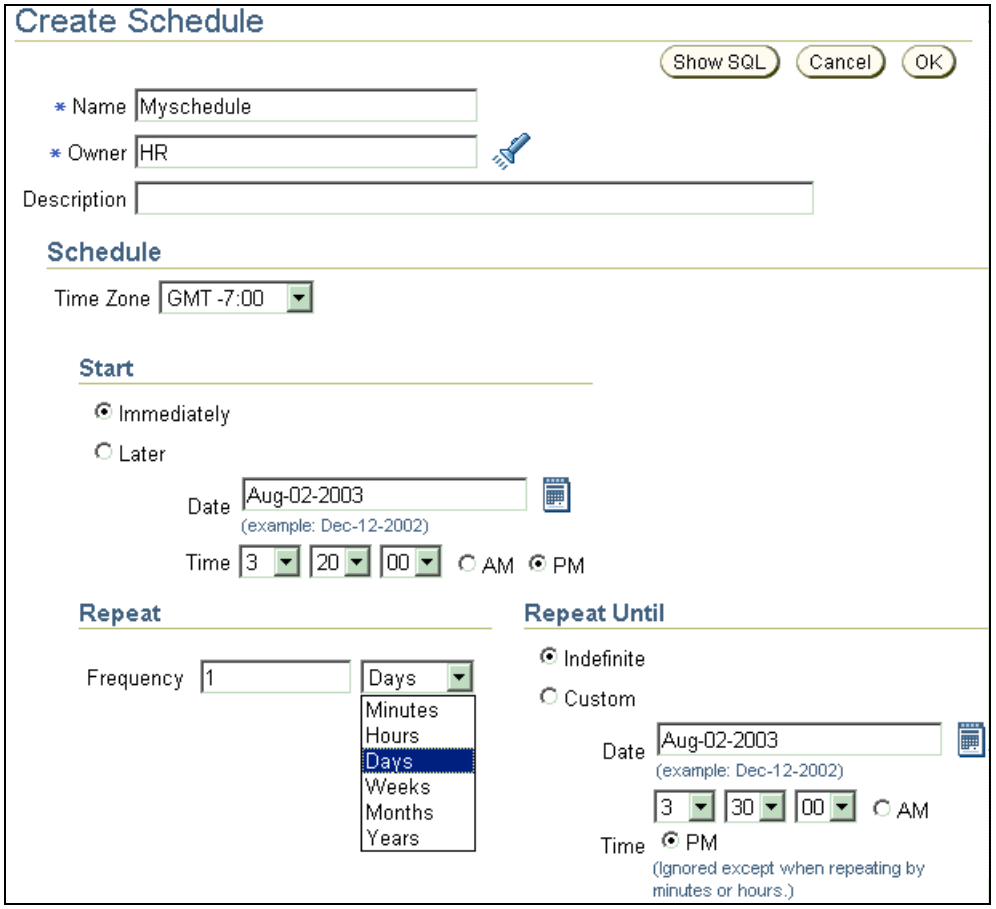


Figure 3: Create Schedule Screen

The DBMS_SCHEDULER API supports two syntaxes for defining the repeat interval: Calendaring expression and PL/SQL. The Calendaring expression is an easy to use, intuitive syntax that provides predictive scheduling using keywords. A PL/SQL expression that returns a date can also be used to specify the repeat interval. Because PL/SQL is a programming language, it makes it possible to satisfy the most demanding schedule requirements, removing any limitations that can be present with the first syntax.

Aside from a normal calendar, businesses generally run on a variety of other calendars, for example, a holiday calendar, fiscal calendar or payroll calendar. Special days, such as company holidays, can be easily defined using rules. For example, although Thanksgiving does not occur on the same date every year, it is always on the fourth Thursday of November.

Great scheduling flexibility can be achieved by combining existing schedules. The Scheduler can perform set operations such as INTERSECT on existing schedules, enabling users to create new composite schedules with minimum effort. For example, if weekdays and holidays are defined, a workday schedule can be easily created by excluding holidays from weekdays. Similarly, it is possible to create a schedule that defines the last workday of every month.

Although common frequencies such as every month or every year might be sufficient for some applications, it can also be very limiting for other applications, for example, financial applications that are based on a fiscal year rather than a calendar year. For such situations user-defined frequencies are supported so custom frequencies such as a fiscal quarter or fiscal year can be defined. This enables users to define not only the last workday of every month but also the last workday of every fiscal quarter. The Scheduler's rich scheduling algorithm easily overcomes the challenge of defining complex and sophisticated schedules such as the Saturday closest to January 31st.

Event-Based Scheduling

Event-based scheduling as the name implies triggers jobs based on real-time events. Events are defined as any state changes or occurrences in the system. Since events can occur at any time and with no associated rhythmic pattern, event-based scheduling eliminates the need to predict and plan for when the event will occur. The just-in-time scheduling optimizes the entire processing of jobs.

Event-based schedules can be defined using the web based GUI shown in Figure 4 or the DBMS_SCHEDULER API. Jobs can be dependent upon one or multiple events. Once all the event conditions are met the job will be executed. After an event has been consumed by the Scheduler it will be cleared and therefore, no longer active.

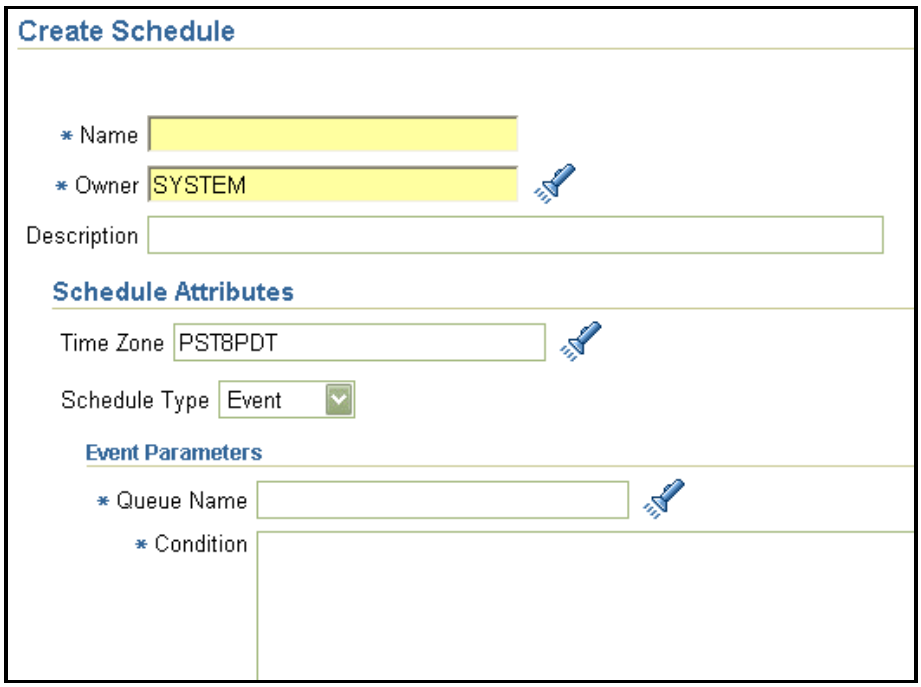


Figure 4: Create Event Schedule Screen

Dependency Scheduling

The ability to execute a task based on an event or on predecessor tasks is one of the most fundamental features of a scheduling system. In the Scheduler, a chain is used to depict a group of inter-dependent tasks that collectively accomplish a business objective. Chains automate the execution of tasks using conditional or branching logic. A simple example of a chain would be, execute job C one hour after the successful completion of job A and job B.

A chain consists of multiple steps combined using dependency rules. A step in the chain can be a program, another chain or an event such as a user intervention. The dependency rules define the conditions that can be used to start or stop a task or the chain itself. Conditions can include the success, failure, completion or exit-code of previous tasks. Logical expressions, such as AND/OR, can be used in the conditions to make it more comprehensive. Figure 5 shows a sample chain, which includes another chain as one of the steps.

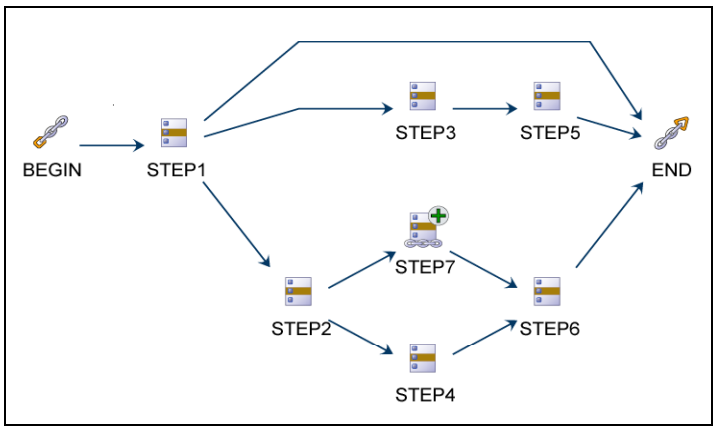


Figure 5: Sample Chain

Job Prioritization

The Scheduler ensures that limited computing resources are allocated appropriately among competing jobs, thus aligning job processing with your business needs. Not all jobs have the same priority and the priority might change over a period of time. The Scheduler regulates the number of jobs that can run at a given time and optimizes system resources to effectively ensure that high-priority jobs finish before low-priority jobs. Without this feature the system could choke if the job volume is high. All the jobs would get started but nothing would complete since not enough resources would be allocated to each job, resulting in thrashing of the system. The Scheduler will attempt to maximize the throughput of jobs.

Job class is a way to manage jobs when resources are at capacity and jobs are waiting to run. Jobs that share common characteristics and behavior can be grouped into larger entities called job classes. You can prioritize among job classes by controlling the resources allocated to each class. Jobs can also be prioritized within a job class. Any number of job classes can be created but a job can only belong in one job class. You may define that only jobs from a specific department or of certain priority can belong to a job class. For example an administrator can create a job class and define it to only accept data warehousing jobs.

The Scheduler exploits the Resource Manager feature of the database to process jobs more efficiently. Using a Resource plan shown in Figure 6, an administrator can control the resource allocation among the job classes by creating policies for managing the consumption of system resources. Because the Scheduler maximizes throughput, it will not start any more jobs within a class when the CPU load of the job class exceeds the threshold defined in the Resource plan. This ensures that the critical jobs have priority and have enough resources to complete. Based on the policy defined in Figure 6, the jobs that belong to the warehouse_group will get 80% of the resources. If the jobs in the warehouse_group do not use all the allocated CPU then the unused CPU will be allocated to other groups according to the allocations specified in level 2.

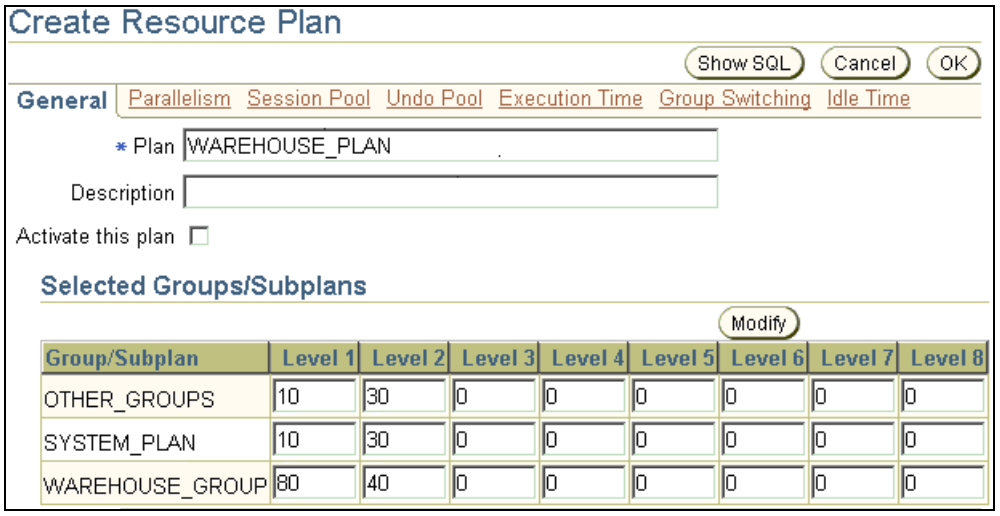


Figure 6: Create Resource Plan Screen

Prioritization of jobs can change over a period of time. For example it might be critical that all the data warehouse loading jobs complete before the morning when reports will be generated against the data warehouse. This means that the data warehousing jobs should have priority in the night but they are not very important during the day. Because the definition of critical jobs can change across time, the Scheduler enables an administrator to also change the priority of jobs over that time frame. The Scheduler uses a window, shown in Figure 7, to change the job prioritization based on a schedule. A window is a time frame with which a Resource plan can be associated. When the window becomes active, the Resource plan associated with it will come into effect and resources will be allocated as specified in the resource plan. Figure 7 shows a window that becomes active daily at 6:00 PM and is active for 12 hours. During this time the WAREHOUSE_PLAN is the active Resource plan and resources will be allocated as specified in the plan.

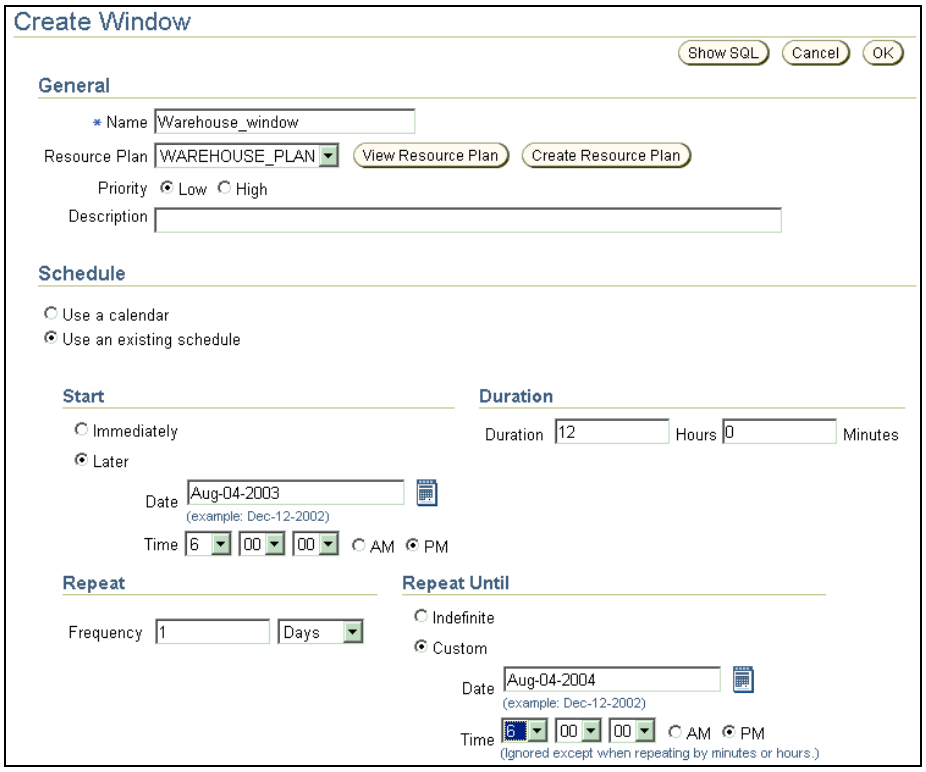
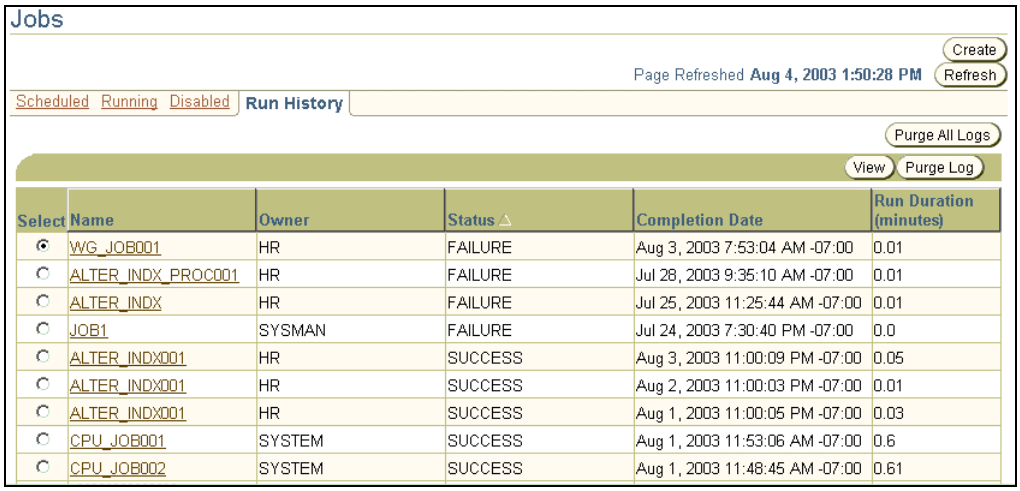


Figure 7: Window Definition Screen

Monitoring and Managing

Monitoring and managing is a key activity in a job system. Jobs can be managed at a group level making it easy to manage large number of jobs. The GUI provides a central overview of all scheduler objects, enabling administrators to easily monitor the progress of jobs. It enables them to quickly identify and rectify the malfunctions in Scheduler activities. As shown in Figure 8, jobs can be filtered and sorted by any attribute of the job, making it easy to identify jobs that are in an error state. Jobs can be viewed, altered, stopped or killed without having to go to another system or screen, by simply clicking on the job name, making it easy to resolve problems.

Special circumstances, for example a change in the status of a job that may arise during the execution of a job can be designated as event triggers. In these cases the Scheduler will raise an event and can take a variety of actions. For example if a job failed, an administrator can be notified. The scheduler supports event triggers for various stages of the job execution including job failure or completion, or if a job ran longer than expected.



Select	Name	Owner	Status	Completion Date	Run Duration (minutes)
◉	WG_JOB001	HR	FAILURE	Aug 3, 2003 7:53:04 AM -07:00	0.01
○	ALTER_INDX_PROC001	HR	FAILURE	Jul 28, 2003 9:35:10 AM -07:00	0.01
○	ALTER_INDX	HR	FAILURE	Jul 25, 2003 11:25:44 AM -07:00	0.01
○	JOB1	SYSMAN	FAILURE	Jul 24, 2003 7:30:40 PM -07:00	0.0
○	ALTER_INDX001	HR	SUCCESS	Aug 3, 2003 11:00:09 PM -07:00	0.05
○	ALTER_INDX001	HR	SUCCESS	Aug 2, 2003 11:00:03 PM -07:00	0.01
○	ALTER_INDX001	HR	SUCCESS	Aug 1, 2003 11:00:05 PM -07:00	0.03
○	CPU_JOB001	SYSTEM	SUCCESS	Aug 1, 2003 11:53:06 AM -07:00	0.6
○	CPU_JOB002	SYSTEM	SUCCESS	Aug 1, 2003 11:48:45 AM -07:00	0.61

Figure 8: Job Monitoring Screen

Job statistics, such as the CPU used by the job, are available and can be used by administrators for planning future capacity requirements.

The Scheduler has a flexible logging mechanism that is designed to be as granular and flexible as possible. The logging level that is set determines what information is logged. Logging levels can be specified at the job class level. It can also be set at the job level if additional logging is required. Since the desired logging level may be dictated by the application or system security policy, the Scheduler supports three levels: full, off and job run. For certain jobs it might be desirable to collect not only the execution information but also every operation performed on the job. In this case full logging can be specified. For other jobs it might only be important to know whether the job ran successfully or not, in which case the logging level would be set to runs only.

Setting full logging on will log every state change that a job undergoes. This creates an audit trail of all Scheduler related activities, which provides a means to understand precisely what changes were made, who made them and when. For normal situations logging only the information related to the execution of the job may be sufficient. Detailed information about the job execution makes it easy to determine what ran and when. All errors generated during execution are logged, therefore problems can be fixed quickly. It is also possible to set logging completely off, in which case nothing is logged.

Any amount of logging can become unmanageable over time. The Scheduler automatically cleans the logs based on the log history attribute. The log history attribute indicates how long to keep the history of jobs. It can be set as a global parameter at the Scheduler level as well as the job class. This helps ensure that the size of the log remains manageable and the important records easily accessible.

Security and User Access

Beginning in November 2001, Oracle started the Unbreakable campaign. It is Oracle's commitment to our customer base to build, deliver and support the most secure, mission-critical software in the business.

To protect the enterprise from unauthorized access, the Scheduler offers a flexible security model that can be customized according to your security needs. There are two levels of security: access to the database and user access to the scheduler objects. Database user authentication controls the users who have access to the database, thus limiting the number of users who can access the Scheduler objects. It supports encryption to ensure that only authorized users can access the information in the database.

Access to the job system is further restricted based on user access. User access control deals with the concept of who has access to what information and what type of operations can be performed. It provides protection against users taking unauthorized actions while logged into the database. What tasks a user can do once logged on depends on the privileges the user has been granted. The Scheduler leverages the highly granular discretionary access control security mechanism built into the database to enforce fine-grained security on the Scheduler objects. It provides the ability to grant only those privileges to a user that will allow him to perform his job functions, but no more. Access to the Scheduler objects can be restricted at the individual object level and object privileges need to be explicitly granted for access.

An administrator can define database users and their privileges using the Enterprise Manager GUI shown in Figure 9, or alternatively, using commands from SQL*PLUS. Privileges can be quickly and easily changed in response to changing business needs.

As explained in the previous section, by setting the full logging attribute, an audit trail enables accountability of actions taken by users of the database.

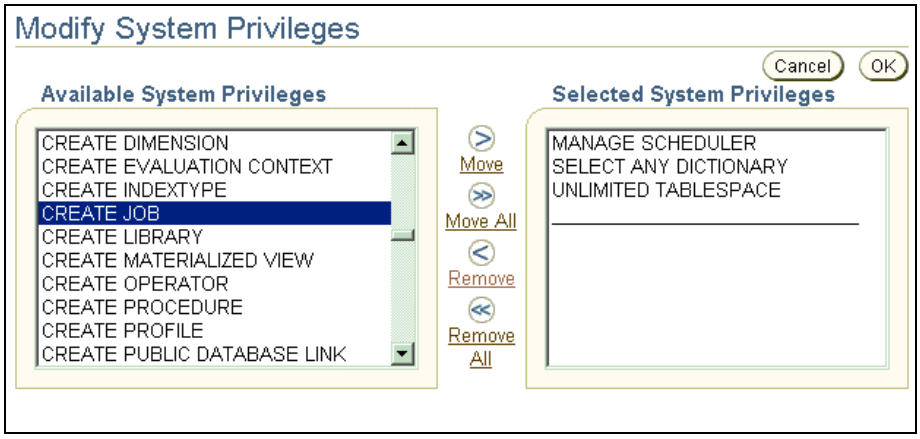


Figure 9: User Access Screen

High Availability and Recovery

It is imperative that the job system can continue operating and be recoverable in the event of a system or component failure. Backup and recovery of all Scheduler activities are handled by the Oracle database. If the database goes down while a job is running, following normal database rules, the transaction will be rolled back. When the database comes back up, all scheduler activities will be resumed and jobs that have the restartable attribute set will be restarted.

The job environment can also be easily replicated, which provides a reliable backup in the event of component failure. Any solution that is used for handling the database failover will also handle the Scheduler, and no extra steps are necessary for ensuring fault tolerance of the Scheduler. In the event that the primary database fails, the backup database will automatically take over.

Load Balancing and Scalability

The scalability of jobs leverages on the scalability features of the Oracle database. Oracle Database 10*g* is uniquely scalable with technologies such as Real Application Clusters. Oracle Real Application Clusters (RAC) provides scalability and reliability without any change to your applications. It runs a single database on a group of servers clustered together that cooperates to perform the same task.

The Scheduler fully supports execution of jobs in such a clustered environment. As the load on the system increases, additional servers can be added to maintain the performance levels. This enables customers to run large numbers of jobs without degradation in performance. Each RAC instance has a job coordinator running; therefore, as instances are added to the cluster, the database can handle increasing numbers of jobs. Load is balanced across the RAC instances automatically. Alternatively, in the job definition, users can specify the service where a job should run. Specifying a service will increase the performance of the job if the data required by the job is cached on the instances that the service handles.

CONCLUSION

Oracle Database 11*g* provides a feature rich Scheduler that enables companies to easily and effectively manage their database maintenance and other routine jobs. It has features to ensure that critical jobs have priority and have enough resources to complete, thus aligning job scheduling with the company's business requirements. A web based GUI provides an easy to use interface with the utmost flexibility in accessing and managing the job system from anywhere. Finally, because the Scheduler is a database feature, it leverages the high availability and scalability features of the database to provide a secure and robust environment for executing jobs.

White Paper Title: Enterprise Scheduling With Oracle Database 11*g*
July 2007
Author: Vira Goorah

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com