26

Managing Resources with Oracle Database Resource Manager

Oracle Database Resource Manager (Resource Manager) enables you to manage resource allocation for a database.

> **Note:**
>
> A multitenant container database is the only supported architecture in Oracle Database 21c and later releases. While the documentation is being revised, legacy terminology may persist. In most cases, "database" and "non-CDB" refer to a CDB or PDB, depending on context. In some contexts, such as upgrades, "non-CDB" refers to a non-CDB from a previous release.

> **Note:**
>
> This chapter discusses using PL/SQL package procedures to administer the Resource Manager. An easier way to administer the Resource Manager is with the graphical user interface of Oracle Enterprise Manager Cloud Control (Cloud Control). For instructions about administering Resource Manager with Cloud Control, see the Cloud Control online help.
>
> To use Resource Manager with Cloud Control:
>
> 1. Access the Database Home Page.
> 2. From the Administration menu, select **Resource Manager**.

- About Oracle Database Resource Manager
  Oracle Database Resource Manager (the Resource Manager) enables you to manage multiple workloads within a database that are contending for system and database resources.
- Enabling Oracle Database Resource Manager and Switching Plans
  You enable Oracle Database Resource Manager (the Resource Manager) by setting the `RESOURCE_MANAGER_PLAN` initialization parameter. This parameter specifies the top plan, identifying the plan to be used for the current instance. If no plan is specified with this parameter, the Resource Manager is not enabled.
- Assigning Sessions to Resource Consumer Groups
  There are automatic and manual methods that database administrators, users, and applications can use to assign sessions to resource consumer groups. When a session is assigned to a resource consumer group, Oracle Database Resource Manager (the Resource Manager) can manage resource allocation for it.

- Managing Resource Plans
  Resource Manager allocates resources to pluggable databases (PDBs) in a multitenant container database (CDB).
- Putting It All Together: Oracle Database Resource Manager Examples
  Examples illustrate how to allocate resources with Resource Manager.
- Managing Multiple Database Instances on a Single Server
  Oracle Database provides a method for managing CPU allocations on a multi-CPU server running multiple database instances. This method is called instance caging. Instance caging and Oracle Database Resource Manager (the Resource Manager) work together to support desired levels of service across multiple instances.
- Maintaining Consumer Groups, Plans, and Directives
  You can maintain consumer groups, resource plans, and resource plan directives for Oracle Database Resource Manager (the Resource Manager). You perform maintenance tasks using the `DBMS_RESOURCE_MANAGER` PL/SQL package.
- Viewing Database Resource Manager Configuration and Status
  You can use several static data dictionary views and dynamic performance views to view the current configuration and status of Oracle Database Resource Manager (the Resource Manager).
- Interacting with Operating-System Resource Control
  Many operating systems provide tools for resource management. These tools often contain "workload manager" or "resource manager" in their names, and are intended to allow multiple applications to share the resources of a single server, using an administrator-defined policy. Examples are Hewlett Packard's Process Resource Manager or Solaris Containers, Zones, and Resource Pools.
- Oracle Database Resource Manager Reference
  Resource Manager includes predefined resource plans, consumer groups, and consumer groups mapping rules. You can query data dictionary views for information about your Resource Manager configuration.
- Operating System CPU Resource Management
  Starting with Oracle Database 23ai, Oracle offers an alternative method for CPU resource management, which operates across all database instances on the server.

26.1 About Oracle Database Resource Manager

Oracle Database Resource Manager (the Resource Manager) enables you to manage multiple workloads within a database that are contending for system and database resources.

> **Note:**
>
> The Resource Manager manages activity in the CDB root automatically.

- CDB and PDB Resource Management
  Using Oracle Resource Manager (Resource Manager), you can create CDB resource plans and set initialization parameters to allocate resources to PDBs.
- Purpose of Resource Management
  When database resource allocation decisions are left to the operating system, workload management can be problematic. The Resource Manager helps solve these problems.

- Consumer Groups, Plans, and Plan Directives
  Resource Manager includes several elements that you can manage.
- User Interface for PDB Resource Management
  You can manage PDB resources using `DBMS_RESOURCE_MANAGER` and initialization parameters.

26.1.1 CDB and PDB Resource Management

Using Oracle Resource Manager (Resource Manager), you can create CDB resource plans and set initialization parameters to allocate resources to PDBs.

In a CDB, multiple workloads within multiple PDBs can complete for system and CDB resources. Resource Manager can manage resources on two levels: CDB and PDB.

CDB Resource Plans

A CDB resource plan allocates resources to its PDBs according to its set of resource plan directives (directives). A parent-child relationship exists between a CDB resource plan and its directives. Each resource plan directive references either a set of PDBs or an individual PDB.

A performance profile specifies shares of system resources for a set of PDBs. PDB performance profiles enable you to manage resources for large numbers of PDBs by specifying Resource Manager directives for profiles instead of individual PDBs.

The directives control allocation of CPU and parallel execution servers. A directive can control the allocation of resources to PDBs based on the share value that you specify for each PDB or PDB performance profile. A higher share value results in more guaranteed resources. For PDBs and PDB performance profiles, you can also set utilization limits for CPU and parallel servers.

You can create a CDB resource plan by using the `CREATE_CDB_PLAN` procedure in the `DBMS_RESOURCE_MANAGER` PL/SQL package, and set a CDB resource plan using the `RESOURCE_MANAGER_PLAN` parameter. You create directives for a CDB resource plan by using the `CREATE_CDB_PLAN_DIRECTIVE` procedure.

PDB Resource Plans

A CDB resource plan allocates a portion of the system resources to a PDB. A PDB resource plan determines how this portion is allocated within the PDB.

You can create a PDB resource plan by using the `CREATE_PLAN` procedure in the `DBMS_RESOURCE_MANAGER` PL/SQL package, and set a PDB resource plan using the `RESOURCE_MANAGER_PLAN` parameter. You create directives for a PDB resource plan by using the `CREATE_PLAN_DIRECTIVE` procedure.

PDB-Level Memory Controls

In a CDB, PDBs may contend for SGA or PGA memory. When you set the following initialization parameters with the PDB as the current container, the parameters limit the memory usage of the current PDB.

Examples of important parameters include:

- `SGA_TARGET` specifies the maximum SGA that the PDB can use at any time.
- `PGA_AGGREGATE_LIMIT` sets the maximum PGA that the PDB can use at any time.

The only shared memory sizing parameter that should be set for a PDB is `SGA_TARGET`, which specifies the maximum SGA that the PDB can use at any time.

PDB-Level I/O Controls

Intensive disk I/O can cause poor performance. Several factors can result in excess disk I/O, such as poorly designed SQL or index and table scans in high-volume transactions. If one PDB generates excessive disk I/O, then it can degrade the performance of other PDBs in the same CDB.

On non-Engineered Systems, use one or both of the following initialization parameters to limit the I/O generated by a particular PDB:

- `MAX_IOPS` limits the number of I/O operations for each second.
- `MAX_MBPS` limits the MB/s for I/O operations.

For Engineered Systems, manage PDB I/Os with I/O Resource Management.

See Also:

- Managing Resource Plans
- *Oracle Database Reference* to learn more about `DB_CACHE_SIZE` and other initialization parameters
- *Oracle Database PL/SQL Packages and Types Reference* to learn more about the `DBMS_RESOURCE_MANAGER` package
- *Oracle Exadata Storage Server Software User's Guide* to learn more about I/O Resource Management

26.1.2 Purpose of Resource Management

When database resource allocation decisions are left to the operating system, workload management can be problematic. The Resource Manager helps solve these problems.

- Purpose of Resource Management for a CDB
  Resource Manager allows a CDB to have more control over how hardware resources are allocated.
- Purpose of Resource Management for PDBs
  In a CDB, workloads within multiple PDBs can compete for system and CDB resources. Resource plans solve this problem.

26.1.2.1 Purpose of Resource Management for a CDB

Resource Manager allows a CDB to have more control over how hardware resources are allocated.

Resource Management Problems for a CDB

When database resource allocation decisions are left to the operating system, you may encounter the following problems with workload management:

- Excessive overhead

  Excessive overhead results from operating system context switching between Oracle Database server processes when the number of server processes is high.

- Inefficient scheduling

  The operating system deschedules database servers while they hold latches, which is inefficient.

- Inappropriate allocation of resources

  The operating system distributes resources equally among all active processes and cannot prioritize one task over another.

- Inability to manage database-specific resources, such as parallel execution servers and active sessions

The Resource Manager Solution

The Resource Manager helps to overcome these problems by allowing a CDB more control over how hardware resources are allocated. In an environment with multiple concurrent user sessions that run jobs with differing priorities, all sessions should not be treated equally. The Resource Manager enables you to classify sessions into groups based on session attributes, and to then allocate resources to those groups in a way that optimizes hardware utilization for your application environment.

With the Resource Manager, you can:

- Guarantee certain sessions a minimum amount of CPU regardless of the load on the system and the number of users.
- Distribute available CPU by allocating percentages of CPU time to different users and applications. In a data warehouse, a higher percentage can be given to ROLAP (relational online analytical processing) applications than to batch jobs.
- Limit the degree of parallelism of any operation performed by members of a group of users.
- Manage the order of parallel statements in the parallel statement queue. Parallel statements from a critical application can be enqueued ahead of parallel statements from a low priority group of users.
- Limit the number of parallel execution servers that a group of users can use. This ensures that all the available parallel execution servers are not allocated to only one group of users.
- Create an active session pool. An *active session pool* consists of a specified maximum number of user sessions allowed to be concurrently active within a group of users. Additional sessions beyond the maximum are queued for execution, but you can specify a timeout period, after which queued jobs will terminate. The active session pool limits the total number of sessions actively competing for resources, thereby enabling active sessions to make faster progress.
- Monitor resources

  Automatically record statistics about resource usage. You can examine these statistics using real-time SQL monitoring and the Resource Manager dynamic performance views (the `V$RSRC_*` views). See "Monitoring Oracle Database Resource Manager" for information about using real-time SQL monitoring and the Resource Manager dynamic performance views.

- Limit the amount of PGA memory used by each session that belongs to a group of users.
- Manage runaway sessions or calls in the following ways:
  - By detecting when a session or call consumes more than a specified amount of CPU, physical I/O, logical I/O, or elapsed time, and then automatically either terminating the session or call, or switching to a consumer group with a lower resource allocation or a limit on the percentage of CPU that the group can use. The SQL statements that are

terminated due to their excessive consumption of system resources are *quarantined*, that is, they are not allowed to run again by generating compilation errors during their subsequent runs.

A logical I/O, also known as a buffer I/O, refers to reads and writes of buffers in the buffer cache. When a requested buffer is not found in memory, the database performs a physical I/O to copy the buffer from either disk or the flash cache into memory, and then a logical I/O to read the cached buffer.

  - By recording detailed information about SQL statements that consume more than a specified amount of CPU, physical I/O, logical I/O, or elapsed time with real-time SQL monitoring.
  - By using the Automatic Workload Repository (AWR) to analyze a persistent record of SQL statements that consume more than a specified amount of CPU, physical I/O, logical I/O, or elapsed time.
  - By logging information about a runaway session without taking any other action related to the session.
- Prevent the execution of operations that the optimizer estimates will run for a longer time than a specified limit.
- Limit the amount of time that a session can be idle. This can be further defined to mean only sessions that are blocking other sessions.
- Allow a database to use different resource plans, based on changing workload requirements. You can dynamically change the resource plan, for example, from a daytime resource plan to a nighttime resource plan, without having to shut down and restart the instance. You can also schedule a resource plan change with Oracle Scheduler. See Oracle Scheduler Concepts for more information.

26.1.2.2 Purpose of Resource Management for PDBs

In a CDB, workloads within multiple PDBs can compete for system and CDB resources. Resource plans solve this problem.

Resource Management Problems for PDBs

When multiple PDBs in a CDB are contending for resources, you may encounter the following problems with workload management:

- Inappropriate allocation of resources among PDBs

  The operating system distributes resources equally among all active processes and cannot prioritize one task over another. Therefore, one or more PDBs might use an inordinate amount of the system resources, leaving the other PDBs starved for resources.
- Inappropriate allocation of resources within a single PDB

  One or more sessions connected to a single PDB might use an inordinate amount of the system resources, leaving other sessions connected to the same PDB starved for resources.
- Inconsistent performance of PDBs

  A single PDB might perform inconsistently when other PDBs are competing for more system resources or less system resources at various times.
- Lack of resource usage data for PDBs

  Resource usage data is critical for monitoring and tuning PDBs. Operating system monitoring tools are typically not useful because there are multiple PDBs running on the system.

The Resource Management Solution in the Multitenant Environment

Resource Manager helps to overcome these problems by enabling you to prioritize and limit the resource usage of specific PDBs. With the Resource Manager, you can:

- Specify that different PDBs should receive different shares of the system resources so that more resources are allocated to the more important PDBs
- Limit the CPU usage of a particular PDB
- Limit the number of parallel execution servers that a particular PDB can use
- Limit the memory usage of a particular PDB
- Specify the amount of memory guaranteed for a particular PDB
- Specify the maximum amount of memory a particular PDB can use
- Use PDB performance profiles for different sets of PDB

  A performance profile for a set of PDBs can specify shares of system resources, CPU usage, and number of parallel execution servers. PDB performance profiles enable you to manage resources for large numbers of PDBs by specifying Resource Manager directives for profiles instead of individual PDBs.
- Limit the resource usage of different sessions connected to a single PDB
- Limit the I/O generated by specific PDBs
- Monitor the resource usage of PDBs

26.1.3 Consumer Groups, Plans, and Plan Directives

Resource Manager includes several elements that you can manage.

- About the Elements of Resource Manager
  The elements of the Resource Manager include resource consumer groups, resource plans, and resource plan directives.
- About Resource Consumer Groups
  A resource consumer group (consumer group) is a collection of user sessions that are grouped together based on their processing needs.
- About Resource Plan Directives
  The Resource Manager allocates resources to consumer groups according to the set of resource plan directives (directives) that belong to the currently active resource plan.
- About Resource Plans
  A resource plan is a container for directives that specify how resources are allocated to resource consumer groups.
- About Subplans
  Instead of referencing a consumer group, a resource plan directive (directive) can reference another resource plan. In this case, the plan is referred to as a subplan.

26.1.3.1 About the Elements of Resource Manager

The elements of the Resource Manager include resource consumer groups, resource plans, and resource plan directives.

Element	Description
Resource consumer group	A group of sessions that are grouped together based on resource requirements. The Resource Manager allocates resources to resource consumer groups, not to individual sessions.
Resource plan	A container for directives that specify how resources are allocated to resource consumer groups. You specify how the database allocates resources by activating a specific resource plan.
Resource plan directive	Associates a resource consumer group with a particular plan and specifies how resources are to be allocated to that resource consumer group.

You use the `DBMS_RESOURCE_MANAGER` PL/SQL package to create and maintain these elements. The elements are stored in tables in the data dictionary. You can view information about them with data dictionary views.

> **See Also:**
>
> "Resource Manager Data Dictionary Views"

26.1.3.2 About Resource Consumer Groups

A resource consumer group (consumer group) is a collection of user sessions that are grouped together based on their processing needs.

When a session is created, it is automatically mapped to a consumer group based on mapping rules that you set up. As a database administrator (DBA), you can manually switch a session to a different consumer group. Similarly, an application can run a PL/SQL package procedure that switches its session to a particular consumer group.

Because the Resource Manager allocates resources (such as CPU) only to consumer groups, when a session becomes a member of a consumer group, its resource allocation is determined by the allocation for the consumer group.

There are special consumer groups that are always present in the data dictionary. They cannot be modified or deleted. They are:

- `SYS_GROUP`

  This is the initial consumer group for all sessions created by user accounts `SYS` or `SYSTEM`. This initial consumer group can be overridden by session-to-consumer group mapping rules.

- `OTHER_GROUPS`

  This consumer group contains all sessions that have not been assigned to a consumer group. Every resource plan must contain a directive to `OTHER_GROUPS`.

There can be no more than 28 resource consumer groups in any active plan.

- Consumer Groups for PDBs
  In a CDB, background and administrative tasks map to the Resource Manager consumer groups that run them optimally.

> **See Also:**
>
> - Table 26-18
> - "Specifying Session-to-Consumer Group Mapping Rules"

26.1.3.2.1 Consumer Groups for PDBs

In a CDB, background and administrative tasks map to the Resource Manager consumer groups that run them optimally.

Resource Manager uses the following rules to map a task to a consumer group:

- A task is mapped to a consumer group in the container that starts the task.

  If a task starts in the CDB root, then the task maps to a consumer group in the CDB root. If the task starts in a PDB, then the task maps to a consumer group in the PDB.
- Many maintenance and administrative tasks automatically map to a consumer group.

  For example, automated maintenance tasks map to `ORA$AUTOTASK`. In certain cases, the tasks map to a consumer group, but the mapping is modifiable. Such tasks include RMAN backup, RMAN image copy, Oracle Data Pump, and In-Memory population.

> **Note:**
>
> *Oracle Database Administrator's Guide* to learn more about the mapping rules for predefined consumer groups

26.1.3.3 About Resource Plan Directives

The Resource Manager allocates resources to consumer groups according to the set of resource plan directives (directives) that belong to the currently active resource plan.

There is a parent-child relationship between a resource plan and its resource plan directives. Each directive references one consumer group, and no two directives for the currently active plan can reference the same consumer group.

A directive has several ways in which it can limit resource allocation for a consumer group. For example, it can control how much CPU the consumer group gets as a percentage of total CPU, and it can limit the total number of sessions that can be active in the consumer group.

CDB resource plan allocates resources to its PDBs according to its set of resource plan directives (directives). A parent-child relationship exists between a CDB resource plan and its resource plan directives. Each directive references either a set of PDBs in a performance profile, or a single PDB. You can specify directives for both individual PDBs and for PDB performance profiles in the same CDB. No two directives for the currently active plan can reference the same PDB or the same PDB performance profile.

- Resources Managed by the Resource Manager
  Resource plan directives specify how resources are allocated to resource consumer groups or subplans. Each directive can specify several different methods for allocating resources to its consumer group or subplan.

- Resource Plan Directives for PDBs
  Directives control allocation of CPU and parallel execution servers.
- Performance Profiles for PDBs
  A **PDB performance profile** configures resource plan directives for a set of PDBs that have the same priorities and resource controls.

26.1.3.3.1 Resources Managed by the Resource Manager

Resource plan directives specify how resources are allocated to resource consumer groups or subplans. Each directive can specify several different methods for allocating resources to its consumer group or subplan.

- CPU
  To manage CPU resources, Resource Manager allocates resources among consumer groups and redistributes CPU resources that were allocated but were not used. You can also set a limit on the amount of CPU resources that can be allocated to a particular consumer group.
- Exadata I/O
  Management attributes enable you to specify CPU resource allocation for Exadata I/O.
- Parallel Execution Servers
  Resource Manager can manage usage of the available parallel execution servers for a database.
- Program Global Area (PGA)
  To manage PGA resources, Resource Manager can limit the amount of PGA memory that can be allocated to each session in a particular consumer group.
- Runaway Queries
  Runaway sessions and calls can adversely impact overall performance if they are not managed properly. Resource Manager can take action when a session or call consumes more than a specified amount of CPU, physical I/O, logical I/O, or elapsed time. Resource Manager can either switch the session or call to a consumer group that is allocated a small amount of CPU or terminate the session or call.
- Active Session Pool with Queuing
  You can control the maximum number of concurrently active sessions allowed within a consumer group. This maximum defines the **active session pool**.
- Undo Pool
  You can specify an undo pool for each consumer group. An undo pool controls the total amount of undo for uncommitted transactions that can be generated by a consumer group.
- Idle Time Limit
  You can specify an amount of time that a session can be idle, after which it is terminated.

26.1.3.3.1.1 CPU

To manage CPU resources, Resource Manager allocates resources among consumer groups and redistributes CPU resources that were allocated but were not used. You can also set a limit on the amount of CPU resources that can be allocated to a particular consumer group.

- Management Attributes
  Management attributes enable you to specify how CPU resources are to be allocated among consumer groups and subplans.

- Utilization Limit
  Use the `UTILIZATION_LIMIT` attribute to impose an absolute upper limit on CPU utilization for a resource consumer group. This absolute limit overrides any redistribution of CPU within a plan.

26.1.3.3.1.1.1 Management Attributes

Management attributes enable you to specify how CPU resources are to be allocated among consumer groups and subplans.

Multiple levels of CPU resource allocation (up to eight levels) provide a means of prioritizing CPU usage within a plan. Consumer groups and subplans at level 2 get resources that were not allocated at level 1 or that were allocated at level 1 but were not completely consumed by a consumer group or subplan at level 1. Similarly, resource consumers at level 3 are allocated resources only when some allocation remains from levels 1 and 2. The same rules apply to levels 4 through 8. Multiple levels not only provide a way of prioritizing, but they provide a way of explicitly specifying how all primary and leftover resources are to be used.

Use the management attributes `MGMT_P`*n*, where *n* is an integer between 1 and 8, to specify multiple levels of CPU resource allocation. For example, use the `MGMT_P1` directive attribute to specify CPU resource allocation at level 1 and `MGMT_P2` directive attribute to specify resource allocation at level 2.

Use management attributes with parallel statement directive attributes, such as Degree of Parallelism Limit and Parallel Server Limit, to control parallel statement queuing. When parallel statement queuing is used, management attributes are used to determine which consumer group is allowed to issue the next parallel statement. For example, if you set the `MGMT_P1` directive attribute for a consumer group to 80, that group has an 80% chance of issuing the next parallel statement.

> **See Also:**
>
> *Oracle Database VLDB and Partitioning Guide* for information about parallel statement queuing

Table 26-1 illustrates a simple resource plan with three levels.

Table 26-1 A Simple Three-Level Resource Plan

Consumer Group	Level 1 CPU Allocation	Level 2 CPU Allocation	Level 3 CPU Allocation
HIGH_GROUP	80%		
LOW_GROUP		50%	
MAINT_SUBPLAN		50%	
OTHER_GROUPS			100%

High priority applications run within `HIGH_GROUP`, which is allocated 80% of CPU. Because `HIGH_GROUP` is at level one, it gets priority for CPU utilization, but only up to 80% of CPU. This leaves a remaining 20% of CPU to be shared 50-50 by `LOW_GROUP` and the `MAINT_SUPLAN` at level 2. Any unused allocation from levels 1 and 2 are then available to `OTHER_GROUPS` at level 3. Because `OTHER_GROUPS` has no sibling consumer groups or subplans at its level, 100% is specified.

Within a particular level, CPU allocations are not fixed. If there is not sufficient load in a particular consumer group or subplan, residual CPU can be allocated to remaining consumer groups or subplans. Thus, when there is only one level, unused allocation by any consumer group or subplan can be redistributed to other "sibling" consumer groups or subplans. If there are multiple levels, then the unused allocation is distributed to the consumer groups or subplans at the next level. If the last level has unused allocations, these allocations can be redistributed to all other levels in proportion to their designated allocations.

As an example of redistribution of unused allocations from one level to another, if during a particular period, HIGH_GROUP consumes only 25% of CPU, then 75% is available to be shared by LOW_GROUP and MAINT_SUBPLAN. Any unused portion of the 75% at level 2 is then made available to OTHER_GROUPS at level 3. However, if OTHER_GROUPS has no session activity at level 3, then the 75% at level 2 can be redistributed to all other consumer groups and subplans in the plan proportionally.

26.1.3.3.1.1.2 Utilization Limit

Use the UTILIZATION_LIMIT attribute to impose an absolute upper limit on CPU utilization for a resource consumer group. This absolute limit overrides any redistribution of CPU within a plan.

In the previous scenario, suppose that due to inactivity elsewhere, LOW_GROUP acquires 90% of CPU. Suppose that you do not want to allow LOW_GROUP to use 90% of the server because you do not want non-critical sessions to inundate the CPUs. The UTILIZATION_LIMIT attribute of resource plan directives can prevent this situation.

Setting the UTILIZATION_LIMIT attribute is optional. If you omit this attribute for a consumer group, there is no limit on the amount of CPU that the consumer group can use. Therefore, if all the other applications are idle, a consumer group that does not have UTILIZATION_LIMIT set can be allocated 100% of the CPU resources.

You can also use the UTILIZATION_LIMIT attribute as the sole means of limiting CPU utilization for consumer groups, without specifying level limits.

Table 26-2 shows a variation of the previous plan. In this plan, using UTILIZATION_LIMIT, CPU utilization is capped at 75% for LOW_GROUP, 50% for MAINT_SUBPLAN, and 75% for OTHER_GROUPS. (Note that the sum of all utilization limits can exceed 100%. Each limit is applied independently.)

Table 26-2 A Three-Level Resource Plan with Utilization Limits

Consumer Group	Level 1 CPU Allocation	Level 2 CPU Allocation	Level 3 CPU Allocation	Utilization Limit
HIGH_GROUP	80%			
LOW_GROUP		50%		75%
MAINT_SUBPLAN		50%		50%
OTHER_GROUPS			100%	75%

In the example described in Table 26-2, if HIGH_GROUP is using only 10% of the CPU at a given time, then the remaining 90% is available to LOW_GROUP and the consumer groups in MAINT_SUBPLAN at level 2. If LOW_GROUP uses only 20% of the CPU, then 70% can be allocated to MAINT_SUBPLAN. However, MAINT_SUBPLAN has a UTILIZATION_LIMIT of 50%. Therefore, even though more CPU resources are available, the server cannot allocate more than 50% of the CPU to the consumer groups that belong to the subplan MAINT_SUBPLAN.

You can set UTILIZATION_LIMIT for both a subplan and the consumer groups that the subplan contains. In such cases, the limit for a consumer group is computed using the limits specified

for the subplan and that consumer group. For example, the `MAINT_SUBPLAN` contains the consumer groups `MAINT_GROUP1` and `MAINT_GROUP2`. `MAINT_GROUP1` has `UTILIZATION_LIMIT` set to 40%. However, the limit for `MAINT_SUBPLAN` is set to 50%. Therefore, the limit for consumer group `MAINT_GROUP1` is computed as 40% of 50%, or 20%. For an example of how to compute `UTILIZATION_LIMIT` for a consumer group when limits are specified for both the consumer group and the subplan to which the group belongs, see "Example 4 - Specifying a Utilization Limit for Consumer Groups and Subplans".

See Also:

- "Creating Resource Plan Directives "
- "Putting It All Together: Oracle Database Resource Manager Examples"

26.1.3.3.1.2 Exadata I/O

Management attributes enable you to specify CPU resource allocation for Exadata I/O.

See Also:

The Exadata documentation for information about using management attributes for Exadata I/O

26.1.3.3.1.3 Parallel Execution Servers

Resource Manager can manage usage of the available parallel execution servers for a database.

- Degree of Parallelism Limit
  You can limit the maximum degree of parallelism for any operation within a consumer group. Use the `PARALLEL_DEGREE_LIMIT_P1` directive attribute to specify the degree of parallelism for a consumer group.
- Parallel Server Limit
  Use the `PARALLEL_SERVER_LIMIT` directive attribute to specify the maximum percentage of the parallel execution server pool that a particular consumer group can use. The number of parallel execution servers used by a particular consumer group is counted as the sum of the parallel execution servers used by all sessions in that consumer group.
- Parallel Queue Timeout
  The `PARALLEL_QUEUE_TIMEOUT` directive attribute enables you to specify the maximum time, in seconds, that a parallel statement can wait in the parallel statement queue before it is timed out.

26.1.3.3.1.3.1 Degree of Parallelism Limit

You can limit the maximum degree of parallelism for any operation within a consumer group. Use the `PARALLEL_DEGREE_LIMIT_P1` directive attribute to specify the degree of parallelism for a consumer group.

The degree of parallelism limit applies to one operation within a consumer group; it does not limit the total degree of parallelism across all operations within the consumer group. However,

you can combine both the PARALLEL_DEGREE_LIMIT_P1 and the PARALLEL_SERVER_LIMIT directive attributes to achieve the desired control. For more information about the PARALLEL_SERVER_LIMIT attribute, see "Parallel Server Limit".

> **See Also:**
>
> *Oracle Database VLDB and Partitioning Guide* for more information about degree of parallelism in producer/consumer operations

26.1.3.3.1.3.2 Parallel Server Limit

Use the PARALLEL_SERVER_LIMIT directive attribute to specify the maximum percentage of the parallel execution server pool that a particular consumer group can use. The number of parallel execution servers used by a particular consumer group is counted as the sum of the parallel execution servers used by all sessions in that consumer group.

It is possible for a single consumer group to launch enough parallel statements to use all of the available parallel execution servers. If this happens when a high-priority parallel statement from a different consumer group is run, then no parallel execution servers are available to allocate to this group. You can avoid such a scenario by limiting the number of parallel execution servers that can be used by a particular consumer group. You can also set the directive PARALLEL_STMT_CRITICAL to BYPASS_QUEUE for the high-priority consumer group so that parallel statements from the consumer group bypass the parallel statement queue.

For example, assume that the total number of parallel execution servers is 32, as set by the PARALLEL_SERVERS_TARGET initialization parameter, and the PARALLEL_SERVER_LIMIT directive attribute for the consumer group MY_GROUP is set to 50%. This consumer group can use a maximum of 50% of 32, or 16 parallel execution servers.

If your resource plan has management attributes (MGMT_P1, MGMT_P2, and so on), then a separate parallel statement queue is managed as a First In First Out (FIFO) queue for each management attribute.

If your resource plan does not have any management attributes, then a single parallel statement queue is managed as a FIFO queue.

In the case of an Oracle Real Application Clusters (Oracle RAC) environment, the target number of parallel execution servers is the sum of (PARALLEL_SERVER_LIMIT * PARALLEL_SERVERS_TARGET / 100) across all Oracle RAC instances. If a consumer group is using the number of parallel execution servers computed above or more, then it has exceeded its limit, and its parallel statements will be queued.

If a consumer group does not have any parallel statements running within an Oracle RAC database, then the first parallel statement is allowed to exceed the limit specified by PARALLEL_SERVER_LIMIT.

> **Note:**
>
> In an Oracle Real Application Clusters (Oracle RAC) environment, the PARALLEL_SERVER_LIMIT attribute applies to the entire cluster and not to a single instance.

- Managing Parallel Statement Queuing Using Parallel Server Limit
  The PARALLEL_SERVER_LIMIT attribute enables you to specify when parallel statements from a consumer group can be queued. Oracle Database maintains a separate parallel statement queue for each consumer group.

> **See Also:**
>
> - "Creating Resource Plan Directives "
> - "Managing Parallel Statement Queuing Using Parallel Server Limit"
> - *Oracle Database VLDB and Partitioning Guide* for information about parallel statement queuing

26.1.3.3.1.3.2.1 Managing Parallel Statement Queuing Using Parallel Server Limit

The PARALLEL_SERVER_LIMIT attribute enables you to specify when parallel statements from a consumer group can be queued. Oracle Database maintains a separate parallel statement queue for each consumer group.

A parallel statement from a consumer group is not run and instead is added to the parallel statement queue of that consumer group if the following conditions are met:

- PARALLEL_DEGREE_POLICY is set to AUTO.

  Setting this initialization parameter to AUTO enables automatic degree of parallelism (Auto DOP), parallel statement queuing, and in-memory parallel execution.

  Note that parallel statements which have PARALLEL_DEGREE_POLICY set to MANUAL or LIMITED are executed immediately and are not added to the parallel statement queue.
- The number of active parallel execution servers across all consumer groups exceeds the PARALLEL_SERVERS_TARGET initialization parameter setting. This condition applies regardless of whether you specify PARALLEL_SERVER_LIMIT. If PARALLEL_SERVER_LIMIT is not specified, then it defaults to 100%.
- The sum of the number of active parallel execution servers for the consumer group and the degree of parallelism of the parallel statement exceeds the target number of active parallel execution servers.

  The target number of active parallel execution servers is computed as follows:

  PARALLEL_SERVER_LIMIT/100 * PARALLEL_SERVERS_TARGET

> **Note:**
>
> Although parallel execution server usage is monitored for all sessions, the parallel execution server directive attributes you set affect only sessions for which parallel statement queuing is enabled (PARALLEL_DEGREE_POLICY is set to AUTO). If a session has the PARALLEL_DEGREE_POLICY set to MANUAL, parallel statements from this session are not queued. However, any parallel execution servers used by such sessions are included in the count that is used to determine the limit for PARALLEL_SERVER_LIMIT. Even if this limit is exceeded, parallel statements from this session are not queued.

> **See Also:**
>
> "Parallel Server Limit"

26.1.3.3.1.3.3 Parallel Queue Timeout

The `PARALLEL_QUEUE_TIMEOUT` directive attribute enables you to specify the maximum time, in seconds, that a parallel statement can wait in the parallel statement queue before it is timed out.

When you use parallel statement queuing, if the database does not have sufficient resources to execute a parallel statement, the statement is queued until the required resources become available. However, there is a chance that a parallel statement may be waiting in the parallel statement queue for longer than is desired. You can prevent such scenarios by specifying the maximum time a parallel statement can wait in the parallel statement queue.

The `PARALLEL_QUEUE_TIMEOUT` attribute can be set for each consumer group. This attribute is applicable even if you do not specify other management attributes (`MGMT_P1`, `MGMT_P2`, and so on) in your resource plan.

> **See Also:**
>
> *Oracle Database VLDB and Partitioning Guide* for more information about parallel statement queuing

> **Note:**
>
> Because the parallel statement queue is clusterwide, all directives related to the parallel statement queue are also clusterwide.

You can control how a timed out parallel statement is handled by setting the `PQ_TIMEOUT_ACTION` attribute for each consumer group. You can set this attribute to the following values:

- `CANCEL` - The statement execution ends with the error `ORA-07454`. This is the default action for a timed out parallel statement.
- `RUN` - The statement runs immediately. If there are not sufficient parallel servers to run the statement immediately, then the statement is downgraded to run at a lower degree of parallelism.

> **See Also:**
>
> "Example of Managing Parallel Statements Using Directive Attributes" for more information about the combined use of all the parallel execution server directive attributes

26.1.3.3.1.4 Program Global Area (PGA)

To manage PGA resources, Resource Manager can limit the amount of PGA memory that can be allocated to each session in a particular consumer group.

To limit the PGA resources for each session in a consumer group, set the `session_pga_limit` parameter in the package procedure `DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE`. The value of this parameter is the maximum amount of PGA memory, in megabytes, allowed for each session in the consumer group. If a session exceeds the limit set for its consumer group, then error ORA-10260 is raised. This limit includes parallel query child processes and job queue processes.

For example, poorly written PL/SQL code can consume an unbounded amount of PGA. You can use the `session_pga_limit` parameter to limit sessions that run PL/SQL code to ensure that those sessions do not use an inordinate amount of PGA resource.

The following table illustrates a simple resource plan with PGA limits.

Table 26-3 A Simple Resource Plan with PGA Limits

Consumer Group	session_pga_limit Value
HIGH_GROUP	20
LOW_GROUP	10
MAINT_SUBPLAN	Null (unlimited)
OTHER_GROUPS	Null (unlimited)

In this resource plan, high priority applications run within `HIGH_GROUP`, and each session in that group is limited to 20 MB of PGA resource. The sessions used by the lower priority applications within `LOW_GROUP` are limited to 10 MB of PGA resource. The sessions used for maintenance jobs within `MAINT_SUPLAN` and other sessions within `OTHER_GROUPS` can use unlimited PGA resource.

> **Note:**
>
> You can limit the PGA usage of a whole instance with the `PGA_AGGREGATE_LIMIT` initialization parameter.

26.1.3.3.1.5 Runaway Queries

Runaway sessions and calls can adversely impact overall performance if they are not managed properly. Resource Manager can take action when a session or call consumes more than a specified amount of CPU, physical I/O, logical I/O, or elapsed time. Resource Manager can either switch the session or call to a consumer group that is allocated a small amount of CPU or terminate the session or call.

> **Note:**
>
> Starting with Oracle Database 12*c* Release 2 (12.2), Resource Manager can also limit the amount of PGA memory that can be allocated to each session in a particular consumer group.

- Automatic Consumer Group Switching
  You can control resource allocation by specifying criteria that, if met, causes the automatic switching of a session to a specified consumer group.
- Canceling SQL and Terminating Sessions
  You can use the Resource Manager to cancel long-running SQL queries or to terminate long-running sessions based on their amount of consumption of system resources, such as CPU and I/O.
- Execution Time Limit
  You can specify a maximum execution time allowed for an operation.

> **See Also:**
>
> "Program Global Area (PGA)"

26.1.3.3.1.5.1 Automatic Consumer Group Switching

You can control resource allocation by specifying criteria that, if met, causes the automatic switching of a session to a specified consumer group.

Typically, this method is used to switch a session from a high-priority consumer group—one that receives a high proportion of system resources—to a lower priority consumer group because that session exceeded the expected resource consumption for a typical session in the group.

See "Specifying Automatic Switching by Setting Resource Limits" for more information.

26.1.3.3.1.5.2 Canceling SQL and Terminating Sessions

You can use the Resource Manager to cancel long-running SQL queries or to terminate long-running sessions based on their amount of consumption of system resources, such as CPU and I/O.

The SQL queries canceled by the Resource Manager can be configured for quarantine using the`DBMS_SQLQ` package subprograms, so that those queries are not allowed to run again.

> **See Also:**
>
> - "Specifying Automatic Switching by Setting Resource Limits" for more information about how to configure the Resource Manager to cancel SQL queries or to terminate sessions based on their consumption of system resources
> - "Quarantine for Execution Plans for SQL Statements Consuming Excessive System Resources" for more information about how to configure quarantine settings for SQL queries using the `DBMS_SQLQ` package subprograms

26.1.3.3.1.5.3 Execution Time Limit

You can specify a maximum execution time allowed for an operation.

If the database estimates that an operation will run longer than the specified maximum execution time, then the operation is terminated with an error. This error can be trapped and the operation rescheduled.

26.1.3.3.1.6 Active Session Pool with Queuing

You can control the maximum number of concurrently active sessions allowed within a consumer group. This maximum defines the **active session pool**.

An **active session** is a session that is actively processing a transaction or SQL statement. Specifically, an active session is either in a transaction, holding a user enqueue, or has an open cursor and has not been idle for over 5 seconds. An active session is considered active even if it is blocked, for example waiting for an I/O request to complete. When the active session pool is full, a session that is trying to process a call is placed into a queue. When an active session completes, the first session in the queue can then be removed from the queue and scheduled for execution. You can also specify a period after which a session in the execution queue times out, causing the call to terminate with an error.

Active session limits should not be used for OLTP workloads. In addition, active session limits should not be used to implement connection pooling or parallel statement queuing.

To manage parallel statements, you must use parallel statement queuing with the `PARALLEL_SERVER_LIMIT` attribute and management attributes (`MGMT_P1`, `MGMT_P2`, and so on).

26.1.3.3.1.7 Undo Pool

You can specify an undo pool for each consumer group. An undo pool controls the total amount of undo for uncommitted transactions that can be generated by a consumer group.

When the total undo generated by a consumer group exceeds its undo limit, the current DML statement generating the undo is terminated. No other members of the consumer group can perform further data manipulation until undo space is freed from the pool.

26.1.3.3.1.8 Idle Time Limit

You can specify an amount of time that a session can be idle, after which it is terminated.

You can also specify a more stringent idle time limit that applies to sessions that are idle and blocking other sessions.

26.1.3.3.2 Resource Plan Directives for PDBs

Directives control allocation of CPU and parallel execution servers.

A directive can control the allocation of resources to PDBs based on the share value that you specify for each PDB or PDB performance profile. A higher share value results in more resources. The settings apply to the set of PDBs that use each profile.

For example, you can specify that `salespdb` is allocated double the resources allocated to `hrpdb` by setting the share value for `salespdb` twice as high as the share value for `hrpdb`. Similarly, you can specify that the `salespdb` performance profile is allocated double the resources allocated to the `hrpdb` performance profile by setting the share value for the `salespdb` performance profile twice as high as the share value for the `hrpdb` performance profile.

You can also specify utilization limits for PDBs and PDB performance profiles. The limit controls allocation to the PDB or performance profile. For example, the limit can control how much CPU `salespdb` gets as a percentage of the total CPU available to the CDB.

You can use both shares and utilization limits together for precise control over the resources allocated to each PDB and PDB performance profile in a CDB.

> **See Also:**
>
> *Oracle Database SQL Language Reference* for more information about PDB lockdown profiles

26.1.3.3.3 Performance Profiles for PDBs

A **PDB performance profile** configures resource plan directives for a set of PDBs that have the same priorities and resource controls.

For example, you might create a performance profiles called Gold, Silver, and Bronze. Each profile specifies a different set of directives depending on the importance of the type of PDB. Gold PDBs are more mission critical than Silver PDBs, which are more mission critical than Bronze PDBs. A PDB specifies its performance profile with the `DB_PERFORMANCE_PROFILE` initialization parameter.

You can use PDB lockdown profiles to specify PDB initialization parameters that control resources, such as `SGA_TARGET` and `PGA_AGGREGATE_LIMIT`. A lockdown profile prevents the PDB administrator from modifying the settings.

Oracle recommends using matching names for performance profiles and lockdown profiles. To prevent PDB owners from switching profiles, Oracle recommends putting the PDB performance profile in the PDB lockdown profile.

26.1.3.4 About Resource Plans

A resource plan is a container for directives that specify how resources are allocated to resource consumer groups.

In addition to the resource plans that are predefined for each Oracle database, you can create any number of resource plans. However, only one resource plan is active at a time. When a resource plan is active, each of its child resource plan directives controls resource allocation for a different consumer group. Each plan must include a directive that allocates resources to the consumer group named `OTHER_GROUPS`. `OTHER_GROUPS` applies to all sessions that belong to a consumer group that is not part of the currently active plan.

> **Note:**
>
> Although the term "resource plan" (or just "plan") denotes one element of the Resource Manager, in this chapter it is also used to refer to a complete *resource plan schema*, which includes the resource plan element itself, its resource plan directives, and the consumer groups that the directives reference. For example, when this chapter refers to the `DAYTIME` resource plan, it could mean either the resource plan element named `DAYTIME`, or the particular resource allocation schema that the `DAYTIME` resource plan and its directives define. Thus, for brevity, it is acceptable to say, "the `DAYTIME` plan favors interactive applications over batch applications."

- About CDB Resource Plans
  Create CDB resource plans that allocate shares and resource limits for PDBs.
- About PDB Resource Plans
  A PDB resource plan determines how the resources for a specific PDB are allocated to consumer groups within this PDB.
- Example: A Simple Resource Plan
  An example illustrates a simple resource plan.

26.1.3.4.1 About CDB Resource Plans

Create CDB resource plans that allocate shares and resource limits for PDBs.

- Shares for Allocating Resources to PDBs
  To allocate resources among PDBs, assign a share value to each PDB or performance profile. A higher share value results in more guaranteed resources for a PDB or the PDBs that use the performance profile.
- Utilization Limits for PDBs
  A utilization limit restrains the system resource usage of a specific PDB or a specific PDB performance profile.
- The Default Directive for PDBs
  When you do not explicitly define directives for a PDB, the PDB uses the default directive for PDBs.

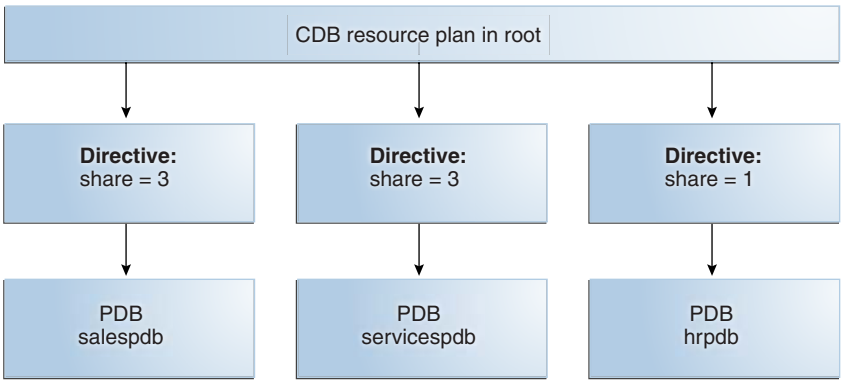
26.1.3.4.1.1 Shares for Allocating Resources to PDBs

To allocate resources among PDBs, assign a share value to each PDB or performance profile. A higher share value results in more guaranteed resources for a PDB or the PDBs that use the performance profile.

Specify a share value for a PDB using the `DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE` procedure and for a PDB performance profile using the `DBMS_RESOURCE_MANAGER.CREATE_CDB_PROFILE_DIRECTIVE` procedure. In both cases, the `shares` parameter specifies the share value for the PDB. Multiple PDBs can use the same PDB performance profile.

The following figure shows an example of three PDBs with share values specified for them in a CDB resource plan.

Figure 26-1 Shares in a CDB Resource Plan



The preceding figure shows that the total number of shares is seven (3 plus 3 plus 1). The `salespdb` and the `servicespdb` PDB are each guaranteed 3/7 of the resources, while the `hrpdb` PDB is guaranteed 1/7 of the resources. However, any PDB can use more than the guaranteed amount of a resource when no resource contention exists.

The following table shows the resource allocation to the PDBs in the preceding figure based on the share values. The table assumes that loads of the PDBs consume all system resources allocated.

Table 26-4 Resource Allocation for Sample PDBs

Resource	Resource Allocation	See Also
CPU	The `salespdb` and `servicespdb` PDBs can consume the same amount of CPU resources. The `salespdb` and `servicespdb` PDBs are each guaranteed three times more CPU resource than the `hrpdb` PDB.	CPU for more information about this resource
Parallel execution servers	Queued parallel queries from the `salespdb` and `servicespdb` PDBs are selected equally. Queued parallel queries from the `salespdb` and `servicespdb` PDBs are selected three times as often as queued parallel queries from the `hrpdb` PDB.	Degree of Parallelism Limit for more information about this resource

26.1.3.4.1.2 Utilization Limits for PDBs

A utilization limit restrains the system resource usage of a specific PDB or a specific PDB performance profile.

You can specify utilization limits for CPU and parallel execution servers. Utilization limits for a PDB are set by the CDB resource plan.

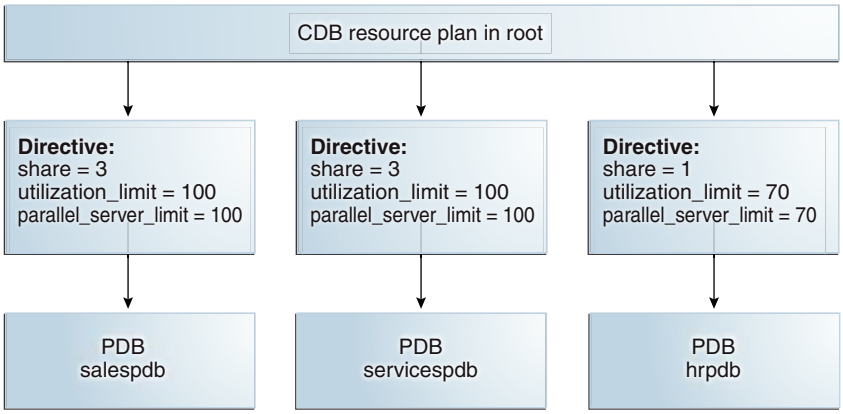
The following table describes utilization limits for PDBs and the Resource Manager action taken when a PDB reaches a utilization limit. For limits specified with a PDB performance profile, the limit applies to every PDB that uses the PDB performance profile. For example, if `pdb1` and `pdb20` have a performance profile `BRONZE`, and if `BRONZE` has a limit set to 10%, then `pdb1` has a 10% limit and `pdb20` has a 10% limit.

Table 26-5 Utilization Limits for PDBs

Resource	Resource Utilization Limit	Resource Manager Action When Limit Is Reached
CPU	The CPU utilization limit for sessions connected to a PDB is set by the `utilization_limit` parameter in subprograms of the `DBMS_RESOURCE_MANAGER` package. The `utilization_limit` parameter specifies the percentage of the system resources that a PDB can use. The value ranges from 0 to 100. You can also limit CPU for a PDB by setting the initialization parameters `CPU_COUNT` (upper limit) and `CPU_MIN_COUNT` (lower limit). For example, if you set `CPU_COUNT` to `8` and `CPU_MIN_COUNT` to `.1` at the PDB level, then the PDB cannot use more than 8 CPU threads at any time and must have at least 1 CPU thread 10% of the time. If both `utilization_limit` and `CPU_COUNT` are specified, then the more restrictive (lower) value is enforced.	Resource Manager throttles the PDB sessions so that the CPU utilization for the PDB does not exceed the utilization limit.
Parallel execution servers	You can limit the number of parallel execution servers in a PDB by means of parallel statement queuing. The limit is a “queuing point” because the database queues parallel queries when the limit is reached. You can set the limit (queuing point) in either of the following ways: • The value of the `PARALLEL_SERVERS_TARGET` initialization parameter setting in the PDB • The value of the `PARALLEL_SERVERS_TARGET` initialization parameter setting in the CDB root multiplied by the value of the `parallel_server_limit` directive set for the PDB in the CDB resource manager plan For example, if the `PARALLEL_SERVERS_TARGET` initialization parameter is set to 200 in the CDB root, and if the `parallel_server_limit` directive for a PDB is set to 10%, then utilization limit for the PDB is 20 parallel execution servers (200 * .10). If the limit is set in both preceding ways, then the lower limit of the two is used. See *Oracle Database Reference* for the default value for `PARALLEL_SERVERS_TARGET`. **Note:** Oracle recommends using the `PARALLEL_SERVERS_TARGET` initialization parameter instead of the `parallel_server_limit` directive in a CDB plan.	Resource Manager queues parallel queries when the number of parallel execution servers used by the PDB would exceed the limit. **Note:** In a CDB, parallel statements are queued based on the `PARALLEL_SERVERS_TARGET` settings at both the PDB and CDB level. A statement is queued when the number of parallel servers used by the PDB exceeds the target for the PDB or when the number of parallel servers used by all PDBs exceeds the target for the CDB.

The following figure shows an example of three PDBs with shares and utilization limits specified for them in a CDB resource plan.

Figure 26-2 Shares and Utilization Limits in a CDB Resource Plan



The preceding figure shows that there are no utilization limits on the `salespdb` and `servicespdb` PDBs because `utilization_limit` and `parallel_server_limit` are both set to 100% for them. However, the `hrpdb` PDB is limited to 70% of the applicable system resources because `utilization_limit` and `parallel_server_limit` are both set to 70%.

> **Note:**
>
> This scenario assumes that the `PARALLEL_SERVERS_TARGET` initialization parameter does not specify a lower limit in a PDB. When the `PARALLEL_SERVERS_TARGET` initialization parameter specifies a lower limit for parallel execution servers in a PDB, the lower limit is used.

> **See Also:**
>
> - Parallel Execution Servers
> - *Oracle Database Reference* to learn about `CPU_COUNT`

26.1.3.4.1.3 The Default Directive for PDBs

When you do not explicitly define directives for a PDB, the PDB uses the default directive for PDBs.

The following table shows the attributes of the initial default directive for PDBs.

Table 26-6 Initial Default Directive Attributes for PDBs

Directive Attribute	Value
`shares`	1
`utilization_limit`	100

Table 26-6 (Cont.) Initial Default Directive Attributes for PDBs

Directive Attribute	Value
parallel_server_limit	100

When a PDB is plugged into a CDB and no directive is defined for it, the PDB uses the default directive for PDBs.

You can create new directives for the new PDB. You can also change the default directive attribute values for PDBs by using the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure in the `DBMS_RESOURCE_MANAGER` package.

When a PDB is unplugged from a CDB, the directive for the PDB is retained. If the same PDB is plugged back into the CDB, then it uses the directive defined for it if the directive was not deleted manually.

Figure 26-3 shows an example of the default directive in a CDB resource plan.

Figure 26-3 Default Directive in a CDB Resource Plan

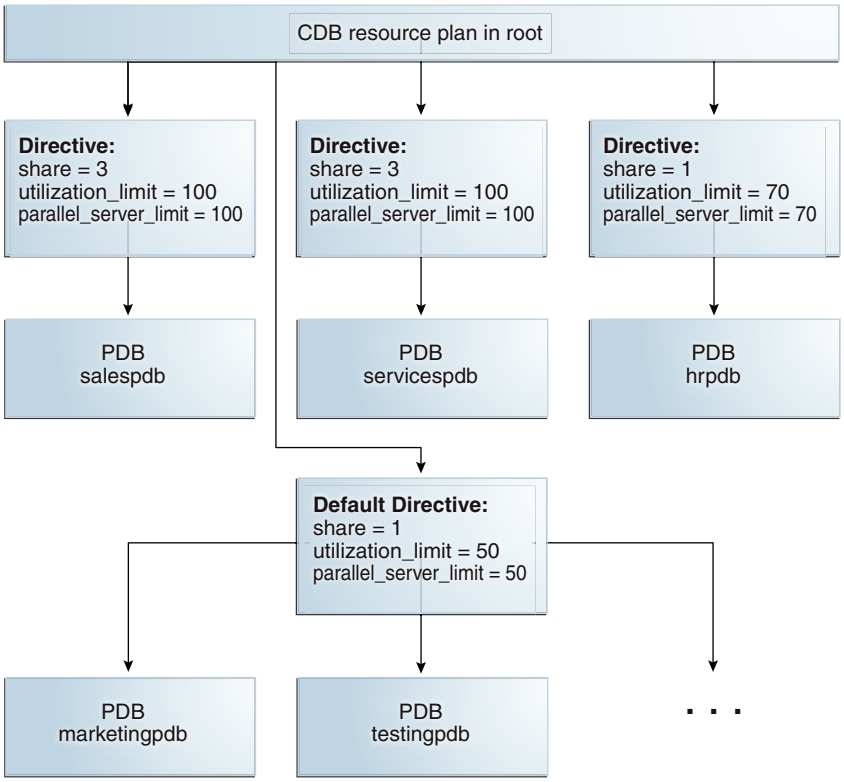


Figure 26-3 shows that the default PDB directive specifies that the `share` is 1, the `utilization_limit` is 50%, and the `parallel_server_limit` is 50%. Any PDB that is part of the CDB and does not have directives defined for it uses the default PDB directive. Figure 26-3 shows the PDBs `marketingpdb` and `testingpdb` using the default PDB directive. Therefore, `marketingpdb` and `testingpdb` each get 1 share and utilization limits of 50.

> **See Also:**
>
> - "Creating New CDB Resource Plan Directives for a PDB"
> - "Updating the Default Directive for PDBs in a CDB Resource Plan"
> - "Parallel Server Limit"
> - *Oracle Multitenant Administrator's Guide* for information about creating and removing PDBs and application containers
> - *Oracle Multitenant Administrator's Guide* for information about unplugging a PDB from a CDB

26.1.3.4.2 About PDB Resource Plans

A PDB resource plan determines how the resources for a specific PDB are allocated to consumer groups within this PDB.

A PDB resource plan differs from a CDB resource plan, which determines the amount of resources allocated to each PDB. The following restrictions apply to PDB resource plans:

- A PDB resource plan cannot have subplans.
- A PDB resource plan cannot have a multiple-level scheduling policy.

If you create a PDB by upgrading a non-CDB from a previous release, and if the non-CDB contains resource plans, then these resource plans might not conform to the preceding restrictions. In this case, Oracle Database automatically transforms these resource plans into equivalent PDB resource plans that meet these requirements. The original resource plans and directives are recorded in the `DBA_RSRC_PLANS` and `DBA_RSRC_PLAN_DIRECTIVES` views with the `LEGACY` status.

- CDB Resource Plan Requirements When Creating PDB Resource Plans
  When you create PDB resource plans, the CDB resource plan must meet certain requirements.
- PDB Resource Plan: Example
  A one-to-many relationship exists between CDB resource plans and PDB resource plans.

> **See Also:**
>
> - "About CDB Resource Plans"
> - Resources Managed by the Resource Manager

26.1.3.4.2.1 CDB Resource Plan Requirements When Creating PDB Resource Plans

When you create PDB resource plans, the CDB resource plan must meet certain requirements.

Create directives for a CDB resource plan by using the `DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE` procedure. Create directives for a PDB resource plan using the `CREATE_PLAN_DIRECTIVE` procedure in the same package. When you create one or more PDB resource plans and there is no CDB resource plan, the CDB uses the `DEFAULT_CDB_PLAN` that is supplied with Oracle Database.

When the CDB resource plan is set to DEFAULT_CDB_PLAN or DEFAULT_MAINTENANCE_PLAN, the share value and utilization limit for each PDB is determined as follows:

share = CPU_MIN_COUNT of the PDB

utilization_limit = CPU_COUNT of the PDB/CPU_COUNT of the CDB

The following table describes the requirements for the CDB resource plan and the results when the requirements are not met. The parameter values described in the "CDB Resource Plan Requirements" column are for the CREATE_CDB_PLAN_DIRECTIVE procedure. The parameter values described in the "Results When Requirements Are Not Met" column are for the CREATE_PLAN_DIRECTIVE procedure.

Table 26-7 CDB Resource Plan Requirements for PDB Resource Plans

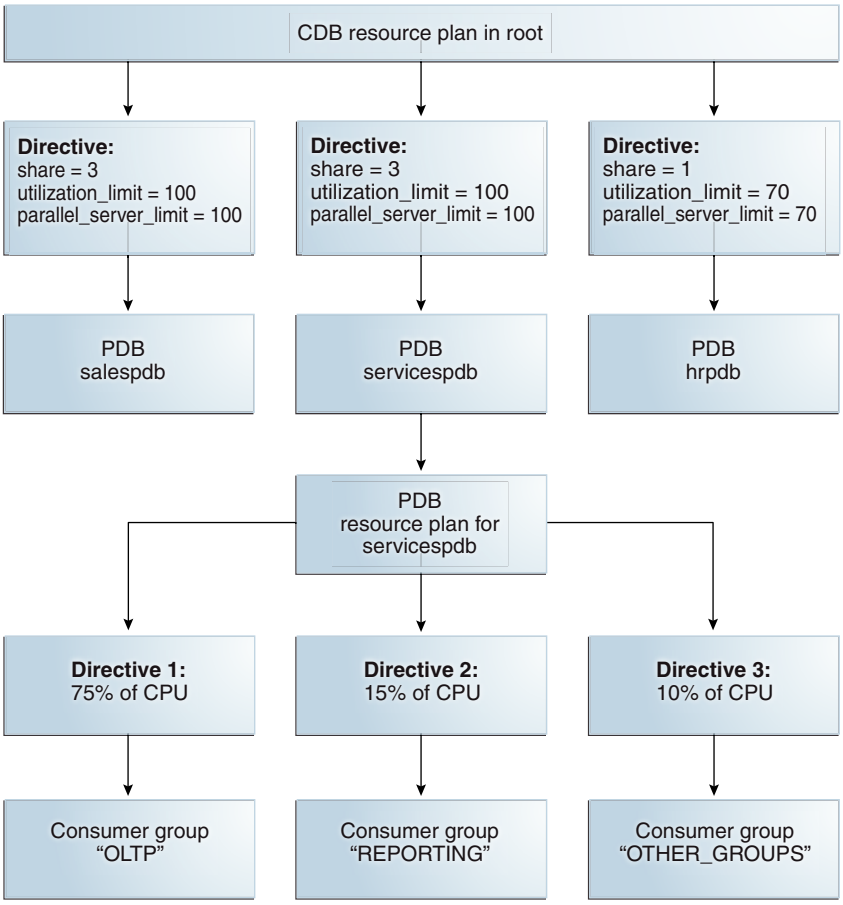
Resource	CDB Resource Plan Requirements	Results When Requirements Are Not Met
CPU	One of the following requirements must be met: • A share value must be specified for the PDB using the shares parameter. • A utilization limit for CPU below 100 must be specified for the PDB using the utilization_limit parameter. These values can be set in a directive for the specific PDB or in a default directive.	The CPU allocation policy of the PDB resource plan is not enforced. The CPU limit specified by the utilization_limit parameter in the PDB resource plan is not enforced.
Parallel execution servers	One of the following requirements must be met: • A share value must be specified for the PDB using the shares parameter. • A parallel server limit below 100 must be specified for the PDB using the parallel_server_limit parameter. These values can be set in a directive for the specific PDB or in a default directive.	The parallel execution server allocation policy of the PDB resource plan is not enforced. The parallel server limit specified by parallel_server_limit in the PDB resource plan is not enforced. However, you can set the PARALLEL_SERVERS_TARGET initialization parameter in a PDB to enforce the parallel limit.

26.1.3.4.2.2 PDB Resource Plan: Example

A one-to-many relationship exists between CDB resource plans and PDB resource plans.

The following figure shows an example of a CDB resource plan and a PDB resource plan.

Figure 26-4 A CDB Resource Plan and a PDB Resource Plan



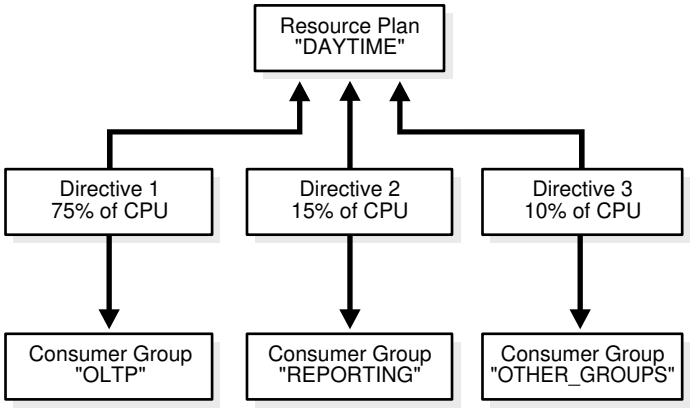
The preceding figure shows some of the directives in a PDB resource plan for the `servicespdb` PDB. Other PDBs in the CDB can also have PDB resource plans.

26.1.3.4.3 Example: A Simple Resource Plan

An example illustrates a simple resource plan.

Figure 26-5 shows a simple resource plan for an organization that runs online transaction processing (OLTP) applications and reporting applications simultaneously during the daytime. The currently active plan, `DAYTIME`, allocates CPU resources among three resource consumer groups. Specifically, `OLTP` is allotted 75% of the CPU time, `REPORTS` is allotted 15%, and `OTHER_GROUPS` receives the remaining 10%. Any group can use more resources than it is guaranteed if there is no resource contention. For example, `OLTP` is guaranteed 75% of the CPU, but if there is no resource contention, it can use up to 100% of the CPU.

Figure 26-5 A Simple Resource Plan



Oracle Database provides a procedure (`CREATE_SIMPLE_PLAN`) that enables you to quickly create a simple resource plan. This procedure is discussed in "Creating a Simple Resource Plan ".

> **Note:**
>
> The currently active resource plan does not enforce allocations until CPU usage is at 100%. If the CPU usage is below 100%, the database is not CPU-bound and hence there is no need to enforce allocations to ensure that all sessions get their designated resource allocation.
>
> In addition, when allocations are enforced, unused allocation by any consumer group can be used by other consumer groups. In the previous example, if the `OLTP` group does not use all of its allocation, the Resource Manager permits the `REPORTS` group or `OTHER_GROUPS` group to use the unused allocation.

26.1.3.5 About Subplans

Instead of referencing a consumer group, a resource plan directive (directive) can reference another resource plan. In this case, the plan is referred to as a subplan.

The subplan itself has directives that allocate resources to consumer groups and other subplans. The resource allocation scheme then works like this: The *top* resource plan (the currently active plan) divides resources among consumer groups and subplans. Each subplan allocates its portion of the total resource allocation among its consumer groups and subplans. You can create hierarchical plans with any number of subplans.

You create a resource subplan in the same way that you create a resource plan. To create a plan that is to be used only as a subplan, you use the `SUB_PLAN` argument in the package procedure `DBMS_RESOURCE_MANAGER.CREATE_PLAN`.

In any top level plan, you can reference a subplan only once. A subplan is not required to have a directive to `OTHER_GROUPS` and cannot be set as a resource plan.

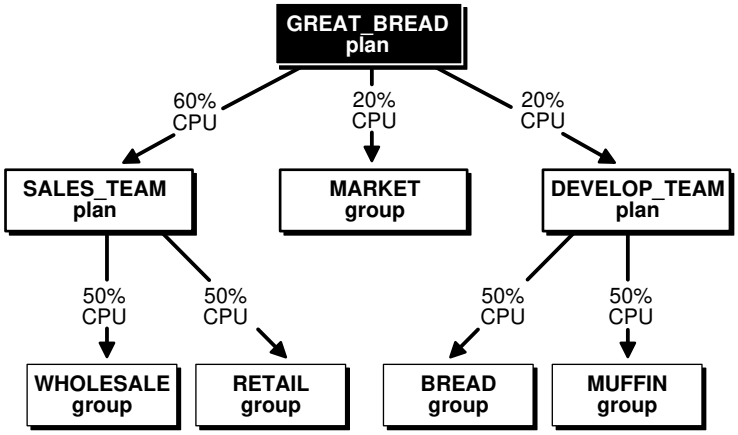
- Example: A Resource Plan with Subplans
  An example illustrates a resource plan with subplans.

26.1.3.5.1 Example: A Resource Plan with Subplans

An example illustrates a resource plan with subplans.

In this example, the Great Bread Company allocates the CPU resource as shown in Figure 26-6. The figure illustrates a top plan (`GREAT_BREAD`) and all of its descendents. For simplicity, the requirement to include the `OTHER_GROUPS` consumer group is ignored, and resource plan directives are not shown, even though they are part of the plan. Rather, the CPU percentages that the directives allocate are shown along the connecting lines between plans, subplans, and consumer groups.

Figure 26-6 A Resource Plan With Subplans



The `GREAT_BREAD` plan allocates resources as follows:

- 20% of CPU resources to the consumer group `MARKET`
- 60% of CPU resources to subplan `SALES_TEAM`, which in turn divides its share equally between the `WHOLESALE` and `RETAIL` consumer groups
- 20% of CPU resources to subplan `DEVELOP_TEAM`, which in turn divides its resources equally between the `BREAD` and `MUFFIN` consumer groups

It is possible for a subplan or consumer group to have multiple parents. An example would be if the `MARKET` group were included in the `SALES_TEAM` subplan. However, a plan cannot contain any loops. For example, the `SALES_TEAM` subplan cannot have a directive that references the `GREAT_BREAD` plan.

> **See Also:**
>
> "Putting It All Together: Oracle Database Resource Manager Examples" for an example of a more complex resource plan.

26.1.4 User Interface for PDB Resource Management

You can manage PDB resources using `DBMS_RESOURCE_MANAGER` and initialization parameters.

- About Resource Manager Administration Privileges
  You must have the required privileges to administer the Resource Manager.
- DBMS_RESOURCE_MANAGER for CDBs and PDBs
  The DBMS_RESOURCE_MANAGER package maintains plans, consumer groups, and plan directives for CDBs and PDBs.
- Initialization Parameters for PDB-Level Resources
  Use initialization parameters to control CPU, memory, sessions, and I/O in a PDB.

26.1.4.1 About Resource Manager Administration Privileges

You must have the required privileges to administer the Resource Manager.

You must have the system privilege ADMINISTER_RESOURCE_MANAGER to administer the Resource Manager. This privilege (with the ADMIN option) is granted to database administrators through the DBA role.

Being an administrator for the Resource Manager enables you to execute all of the procedures in the DBMS_RESOURCE_MANAGER PL/SQL package.

You may, as an administrator with the ADMIN option, choose to grant the administrative privilege to other users or roles. To do so, use the DBMS_RESOURCE_MANAGER_PRIVS PL/SQL package. The relevant package procedures are listed in the following table.

Procedure	Description
GRANT_SYSTEM_PRIVILEGE	Grants the ADMINISTER_RESOURCE_MANAGER system privilege to a user or role.
REVOKE_SYSTEM_PRIVILEGE	Revokes the ADMINISTER_RESOURCE_MANAGER system privilege from a user or role.

The following PL/SQL block grants the administrative privilege to user HR, but does not grant HR the ADMIN option. Therefore, HR can execute all of the procedures in the DBMS_RESOURCE_MANAGER package, but HR cannot use the GRANT_SYSTEM_PRIVILEGE procedure to grant the administrative privilege to others.

```
BEGIN
  DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SYSTEM_PRIVILEGE(
   GRANTEE_NAME   => 'HR',
   PRIVILEGE_NAME => 'ADMINISTER_RESOURCE_MANAGER',
   ADMIN_OPTION   => FALSE);
END;
/
```

You can revoke this privilege using the REVOKE_SYSTEM_PRVILEGE procedure.

> **Note:**
>
> The ADMINISTER_RESOURCE_MANAGER system privilege can only be granted or revoked using the DBMS_RESOURCE_MANAGER_PRIVS package. It cannot be granted or revoked through the SQL GRANT or REVOKE statements.

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference.* for information about the `DBMS_RESOURCE_MANAGER` package
> - *Oracle Database PL/SQL Packages and Types Reference.* for information about the `DBMS_RESOURCE_MANAGER_PRIVS` package
> - *Oracle Database Security Guide* for information about the `ADMIN` option

26.1.4.2 DBMS_RESOURCE_MANAGER for CDBs and PDBs

The `DBMS_RESOURCE_MANAGER` package maintains plans, consumer groups, and plan directives for CDBs and PDBs.

The following table describes the program units related to managing resources in PDBs.

Table 26-8 DBMS_RESOURCE_MANAGER Program Units

PL/SQL Program Unit	Description
`CREATE_CDB_PLAN_DIRECTIVE`	This procedure creates the plan directives of the CDB resource plan. Plan directives specify the resource allocation policy for PDBs.
`CREATE_CDB_PROFILE_DIRECTIVE`	This procedure creates the performance profile directives of the CDB resource plan. The directives specify the resource allocation policy for PDBs that use the performance profile.
`CREATE_CDB_PLAN`	This procedure creates entries that define CDB resource plans.
`UPDATE_CDB_DEFAULT_DIRECTIVE`	This procedure updates the plan directives of the CDB resource plan.
`UPDATE_CDB_PLAN`	This procedure updates the CDB resource plan.

> **Note:**
>
> *Oracle Database PL/SQL Packages and Types Reference* to learn more about `DBMS_RESOURCE_MANAGER`

26.1.4.3 Initialization Parameters for PDB-Level Resources

Use initialization parameters to control CPU, memory, sessions, and I/O in a PDB.

- CPU-Related Initialization Parameters for PDBs
  The `CPU_COUNT` initialization parameter specifies the number of CPU threads available to a PDB.
- Memory-Related Initialization Parameters for PDBs
  Several initialization parameters control the SGA and PGA usage of a PDB.
- Session-Related Initialization Parameters for PDBs
  Several initialization parameters control how sessions consume resources in a PDB.

- I/O-Related Initialization Parameters for PDBs
  The MAX_IOPS and MAX_MBPS initialization parameters limit the disk I/O generated by a PDB.

26.1.4.3.1 CPU-Related Initialization Parameters for PDBs

The CPU_COUNT initialization parameter specifies the number of CPU threads available to a PDB.

The term **CPU thread** refers to a thread of execution on a CPU core. For example, a server might have 4 CPU sockets. Each CPU in a socket might have 2 cores, making a total of 8 cores. If each CPU core were multithreaded, with 2 threads of execution in each core, then the server would have a total of 16 CPU threads.

If CPU Resource Management is enabled for the CDB, then the PDB-level CPU_MIN_COUNT and CPU_COUNT initialization parameters manage CPU resources for the PDB. CPU Resource Management is enabled, for example, when the DEFAULT_CDB_PLAN is set at the CDB level. The CPU Resource Manager *cages* (restricts) the CPU for the PDB according to the lesser of CPU_COUNT and the PDB-level utilization_limit directive, if it exists. If the CDB resource plan has no shares or utilization limits set in non-default directives, then the CPU Resource Manager uses the PDB-level CPU_MIN_COUNT to set the PDB shares in its CDB resource plan.

> **Note:**
>
> CPU_COUNT and CPU_MIN_COUNT do not specify where the threads must be obtained, that is, on which specific CPU or core.

Table 26-9 Initialization Parameters That Control CPU Usage in PDBs

Initialization Parameter	Description	Default Value at PDB Level
CPU_COUNT	Specifies the maximum number of CPU threads that the PDB can use at one time. If set to a nonzero value, then Oracle Database uses this count rather than the actual number of CPUs, thus disabling dynamic CPU reconfiguration. CPU_COUNT works the same way as the utilization_limit directive in the CDB plan. However, the CPU_COUNT limit is expressed in terms of number of CPU threads rather than utilization percentage. If both the utilization_limit and CPU_COUNT are specified, then the lower limit is enforced. **Note:** When the PDB is plugged into a new container, the CPU_COUNT setting remains with the plugged-in PDB.	CPU_COUNT of the CDB

Table 26-9 (Cont.) Initialization Parameters That Control CPU Usage in PDBs

Initialization Parameter	Description	Default Value at PDB Level
CPU_MIN_COUNT	Specifies the minimum number of CPU threads for the PDB. Valid values are 0.1 to the CPU_COUNT setting. The value must be a multiple of 0.05. When less than 1, the value specifies the minimum percentage of time that the PDB requires a thread. For example, a setting of 0.1 means that over a span of 10 seconds, the PDB requires a CPU thread at least 1 second. The CDB is oversubscribed when the sum of the PDB-level CPU_MIN_COUNT settings across all PDBs that are open on a database instance exceeds the CDB-level CPU_MIN_COUNT setting. For example, if a single-instance CDB containing 100 PDBs has a CPU_MIN_COUNT of 8, and if each PDB has a CPU_MIN_COUNT setting of .1, then the CDB is oversubscribed. The minimum CPU is only guaranteed when the CDB is not oversubscribed.	CPU_COUNT

26.1.4.3.2 Memory-Related Initialization Parameters for PDBs

Several initialization parameters control the SGA and PGA usage of a PDB.

When the PDB is the current container, the initialization parameters in the following table control the memory usage of the current PDB. When one or more of these parameters is set for a PDB, ensure that the CDB and the other PDBs have sufficient memory for their operations.

The initialization parameters in the following table control the memory usage of PDBs only if the following conditions are met:

- The NONCDB_COMPATIBLE initialization parameter is set to FALSE in the CDB root.
- The MEMORY_TARGET initialization parameter is not set or is set to 0 (zero) in the CDB root.

Table 26-10 Initialization Parameters That Control the Memory Usage of PDBs

Initialization Parameter	Description (When Set at PDB Level)	Default at PDB Level
DB_CACHE_SIZE	Sets the minimum guaranteed buffer cache size for the PDB. This parameter is optional at the PDB level. If the SGA_TARGET initialization parameter is not set, and if the DB_CACHE_SIZE initialization parameter is set at the CDB level, then the following requirements must be met: • The value of DB_CACHE_SIZE set in a PDB must be less than or equal to 50% of the DB_CACHE_SIZE value at the CDB level. • The sum of the DB_CACHE_SIZE values across all the PDBs in the CDB must be less than or equal to 50% of the DB_CACHE_SIZE value at the CDB level. If SGA_TARGET is set at the CDB level, then the following requirements must be met to avoid an error: • The values of DB_CACHE_SIZE plus SHARED_POOL_SIZE in a PDB must be less than or equal to 50% of the PDB's SGA_TARGET value. • The values of DB_CACHE_SIZE plus SHARED_POOL_SIZE in a PDB must be less than or equal to 50% of the SGA_TARGET value at the CDB level. • The sum of DB_CACHE_SIZE plus SHARED_POOL_SIZE across all the PDBs in a CDB must be less than or equal to 50% of the SGA_TARGET value at the CDB level.	None

Table 26-10 (Cont.) Initialization Parameters That Control the Memory Usage of PDBs

Initialization Parameter	Description (When Set at PDB Level)	Default at PDB Level
SHARED_POOL_SIZE	Sets the minimum guaranteed shared pool size for the PDB. If the SGA_TARGET initialization parameter is not set, but the SHARED_POOL_SIZE initialization parameter is set at the CDB level, then the following requirements must be met: • The value of SHARED_POOL_SIZE set in a PDB must be less than or equal to 50% of the SHARED_POOL_SIZE value at the CDB level. • The sum of the SHARED_POOL_SIZE values across all the PDBs in the CDB must be less than or equal to 50% of the SHARED_POOL_SIZE value at the CDB level. When the SGA_TARGET initialization parameter is set to a nonzero value at the CDB level, the following requirements must be met to avoid an error: • The values of DB_CACHE_SIZE plus SHARED_POOL_SIZE in a PDB must be less than or equal to 50% of the PDB's SGA_TARGET value. • The values of DB_CACHE_SIZE plus SHARED_POOL_SIZE in a PDB must be less than or equal to 50% of the SGA_TARGET value at the CDB level. • The sum of DB_CACHE_SIZE plus SHARED_POOL_SIZE across all the PDBs in a CDB must be less than or equal to 50% of the SGA_TARGET value at the CDB level.	None When this parameter is not set at the PDB level, the PDB has no limit for the amount of shared pool it can use, other than the CDB's shared pool size.
SGA_MIN_SIZE	Sets the minimum SGA size for the PDB. The setting must meet the following requirements: • SGA_TARGET initialization parameter must *not* be set or must be set to 0 (zero) in the CDB root. Otherwise, setting SGA_MIN_SIZE in a PDB has no effect. • It must be less than or equal to 50% of the setting for the SGA_TARGET in the CDB root. • It must be less than or equal to 50% of the setting for the SGA_TARGET in the PDB. • The sum of the SGA_MIN_SIZE settings for all PDBs must be less than or equal to 50% of the setting for the SGA_TARGET in the CDB root. Note that setting SGA_MIN_SIZE at the CDB level has no effect.	0
SGA_TARGET	Sets the maximum SGA size for the PDB. The PDB enforces the PDB-level SGA_TARGET setting only if the SGA_TARGET initialization parameter is set to a nonzero value in the CDB root. The SGA_TARGET setting in the PDB must be less than or equal to the SGA_TARGET setting in the CDB root.	SGA_TARGET at the CDB level multiplied by ratio of PDB-level CPU_COUNT / CDB-level CPU_COUNT

Table 26-10 (Cont.) Initialization Parameters That Control the Memory Usage of PDBs

Initialization Parameter	Description (When Set at PDB Level)	Default at PDB Level
PGA_AGGREGATE_LIMIT	Sets the maximum PGA size for the PDB. If you set PGA_AGGREGATE_LIMIT manually, then the value must meet the following requirements: • It must be less than or equal to the setting for the PGA_AGGREGATE_LIMIT in the CDB root. • It must be greater than or equal to two times the setting for the PGA_AGGREGATE_TARGET in the PDB.	PGA_AGGREGATE_LIMIT at the CDB level multiplied by ratio of PDB-level CPU_COUNT / CDB-level CPU_COUNT
PGA_AGGREGATE_TARGET	Sets the target aggregate PGA size for the PDB. If you set PGA_AGGREGATE_TARGET manually, then the value must meet the following requirements: • It must be less than or equal to the PGA_AGGREGATE_TARGET value set at the CDB level. • It must be less than or equal to 50% of the PGA_AGGREGATE_LIMIT initialization parameter value set at the CDB level. • It must be less than or equal to 50% of the PGA_AGGREGATE_LIMIT value set in the PDB.	PGA_AGGREGATE_TARGET at the CDB level multiplied by ratio of PDB-level CPU_COUNT / CDB-level CPU_COUNT

Example 26-1 Setting the Maximum Aggregate PGA Memory Available for a PDB

With the PDB as the current container, run the following SQL statement to set the PGA_AGGREGATE_LIMIT initialization parameter both in memory and in the SPFILE to 90 MB:

```
ALTER SYSTEM SET PGA_AGGREGATE_LIMIT = 90M SCOPE = BOTH;
```

Example 26-2 Setting the Minimum SGA Size for a PDB

With the PDB as the current container, run the following SQL statement to set the SGA_MIN_SIZE initialization parameter both in memory and in the SPFILE to 500 MB:

```
ALTER SYSTEM SET SGA_MIN_SIZE = 500M SCOPE = BOTH;
```

26.1.4.3.3 Session-Related Initialization Parameters for PDBs

Several initialization parameters control how sessions consume resources in a PDB.

Table 26-11 Initialization Parameters That Control the Session Usage of PDBs

Initialization Parameter	Description (When Set at PDB Level)	Default at PDB Level
SESSIONS	Sets the maximum of number of sessions that a PDB can use. If the PDB tries to use more sessions than configured by its SESSIONS parameter, then an ORA-00018 error message is generated. For PDBs, the SESSIONS parameter does not count recursive sessions and therefore does not require the 10% adjustment. The SESSIONS parameter for a PDB can only be modified by the PDB. It cannot be set higher than the SESSIONS value set at the CDB level.	SESSIONS at the CDB level multiplied by ratio of PDB-level CPU_COUNT / CDB-level CPU_COUNT
MAX_IDLE_TIME	Specifies the maximum number of minutes that a session can be idle. After the maximum is reached, Oracle Database automatically terminates the session.	0 (not set)
MAX_IDLE_BLOCKER_TIME	Sets the number of minutes that a session can be idle before it is a candidate for termination. With this parameter, an idle session is terminated if it is blocking another session. Oracle Database considers a session blocked in any of the following situations: • The session is holding a lock needed by another session. • The session is a parallel operation and its consumer group, PDB, or CDB has either reached its maximum parallel server limit or has queued parallel operations. • The PDB or database instance for the session is about to hit its sessions or processes limit. Unlike MAX_IDLE_TIME, MAX_IDLE_BLOCKER_TIME terminates resources only when they are needed.	0 (not set)

26.1.4.3.4 I/O-Related Initialization Parameters for PDBs

The MAX_IOPS and MAX_MBPS initialization parameters limit the disk I/O generated by a PDB.

A large amount of disk I/O can cause poor performance. Several factors can result in excess disk I/O, such as poorly designed SQL or index and table scans in high-volume transactions. If one PDB is generating heavy disk I/O, then it can degrade the performance of other PDBs.

Use one or both of the following initialization parameters to limit the I/O generated by a specific PDB:

- The MAX_IOPS initialization parameter limits the number of I/O operations for each second.
- The MAX_IOPS initialization parameter limits the megabytes for I/O operations for each second.

If you set both preceding initialization parameters for a single PDB, then Oracle Database enforces both limits. The MAX_IOPS and MAX_IOPS limits are *not* enforced for Oracle Exadata, which uses I/O Resource Management (IORM) to manage I/Os between PDBs.

If these initialization parameters are set with the CDB root as the current container, then the values become the default values for all containers in the CDB. If they are set with an application root as the current container, then the values become the default values for all application PDBs in the application container. When they are set with a PDB or application

PDB as the current container, then the settings take precedence over the default settings in the CDB root or the application root.

The default for both initialization parameters is 0 (zero). If these initialization parameters are set to 0 (zero) in a PDB, and the CDB root is set to 0, then there is no I/O limit for the PDB. If these initialization parameters are set to 0 (zero) in an application PDB, and its application root is set to 0, then there is no I/O limit for the application PDB.

Critical I/O operations, such as ones for the control file and password file, are exempted from the limit and continue to run even if the limit is reached. However, all I/O operations, including critical I/O operations, are counted when the number of I/O operations and the megabytes for I/O operations are calculated.

You can use the `DBA_HIST_RSRC_PDB_METRIC` view to calculate a reasonable I/O limit for a PDB. Consider the values in the following columns when calculating a limit: `IOPS`, `IOMBPS`, `IOPS_THROTTLE_EXEMPT`, and `IOMBPS_THROTTLE_EXEMPT`. The `rsmgr:io rate limit` wait event indicates that a limit was reached.

Example 26-3 Limiting the I/O Generated by a PDB

With the PDB as the current container, run the following SQL statement to set the `MAX_IOPS` initialization parameter both in memory and in the SPFILE to a limit of 1,000 I/O operations for each second:

```
ALTER SYSTEM SET MAX_IOPS = 1000 SCOPE = BOTH;
```

Example 26-4 Limiting the Megabytes of I/O Generated by a PDB

With the PDB as the current container, run the following SQL statement to set the `MAX_MBPS` initialization parameter both in memory and in the SPFILE to a limit of 200 MB of I/O for each second:

```
ALTER SYSTEM SET MAX_MBPS = 200 SCOPE = BOTH;
```

See Also:

- *Oracle Multitenant Administrator's Guide* for information about modifying a PDB at the system level
- *Oracle Database Reference* for more information about the `MAX_IOPS` initialization parameter
- *Oracle Database Reference* for more information about the `MAX_MBPS` initialization parameter

26.2 Enabling Oracle Database Resource Manager and Switching Plans

You enable Oracle Database Resource Manager (the Resource Manager) by setting the `RESOURCE_MANAGER_PLAN` initialization parameter. This parameter specifies the top plan,

identifying the plan to be used for the current instance. If no plan is specified with this parameter, the Resource Manager is not enabled.

By default the Resource Manager is not enabled, except in the following situations:

- During preconfigured maintenance windows, described later in this section.
- When Oracle Database In-Memory is enabled by setting the `INMEMORY_SIZE` initialization parameter to a value greater than 0.

The following statement in a text initialization parameter file activates the Resource Manager upon database startup and sets the top plan as `mydb_plan`.

```
RESOURCE_MANAGER_PLAN = mydb_plan
```

You can also activate or deactivate the Resource Manager, or change the current top plan, using the `DBMS_RESOURCE_MANAGER.SWITCH_PLAN` package procedure or the `ALTER SYSTEM` statement.

The following SQL statement sets the top plan to `mydb_plan`, and activates the Resource Manager if it is not already active:

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = 'mydb_plan';
```

An error message is returned if the specified plan does not exist in the data dictionary.

Automatic Enabling of the Resource Manager by Oracle Scheduler Windows

The Resource Manager automatically activates if an Oracle Scheduler window that specifies a resource plan opens. When the Scheduler window closes, the resource plan associated with the window is disabled, and the resource plan that was running before the Scheduler window opened is reenabled. (If no resource plan was enabled before the window opened, then the Resource Manager is disabled.) In an Oracle Real Application Clusters environment, a Scheduler window applies to all instances, so the window's resource plan is enabled on every instance.

Note that by default a set of automated maintenance tasks run during **maintenance windows**, which are predefined Scheduler windows that are members of the `MAINTENANCE_WINDOW_GROUP` window group and which specify the `DEFAULT_MAINTENANCE_PLAN` resource plan. Thus, the Resource Manager activates by default during maintenance windows. You can modify these maintenance windows to use a different resource plan, if desired.

> **Note:**
>
> If you change the plan associated with maintenance windows, then ensure that you include the subplan `ORA$AUTOTASK` in the new plan.

> **See Also:**
>
> - "Windows"
> - Managing Automated Database Maintenance Tasks

Disabling Plan Switches by Oracle Scheduler Windows

In some cases, the automatic change of Resource Manager plans at Scheduler window boundaries may be undesirable. For example, if you have an important task to finish, and if you set the Resource Manager plan to give your task priority, then you expect that the plan will remain the same until you change it. However, because a Scheduler window could activate after you have set your plan, the Resource Manager plan might change while your task is running.

To prevent this situation, you can set the `RESOURCE_MANAGER_PLAN` initialization parameter to the name of the plan that you want for the system and prepend "`FORCE:`" to the name, as shown in the following SQL statement:

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = 'FORCE:mydb_plan';
```

Using the prefix `FORCE:` indicates that the current resource plan can be changed only when the database administrator changes the value of the `RESOURCE_MANAGER_PLAN` initialization parameter. This restriction can be lifted by rerunning the command without preceding the plan name with "`FORCE:`".

The `DBMS_RESOURCE_MANAGER.SWITCH_PLAN` package procedure has a similar capability.

> **See Also:**
>
> *Oracle Database PL/SQL Packages and Types Reference* for more information on `DBMS_RESOURCE_MANAGER.SWITCH_PLAN`.

Disabling the Resource Manager

To disable the Resource Manager, complete the following steps:

1. Issue the following SQL statement:

   ```
   ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = '';
   ```

2. Disassociate the Resource Manager from all Oracle Scheduler windows.

   To do so, for any Scheduler window that references a resource plan in its `resource_plan` attribute, use the `DBMS_SCHEDULER.SET_ATTRIBUTE` procedure to set `resource_plan` to the empty string ('). Qualify the window name with the `SYS` schema name if you are not logged in as user `SYS`. You can view Scheduler windows with the `DBA_SCHEDULER_WINDOWS` data dictionary view. See "Altering Windows" and *Oracle Database PL/SQL Packages and Types Reference* for more information.

   > **Note:**
   >
   > By default, all maintenance windows reference the `DEFAULT_MAINTENANCE_PLAN` resource plan. To completely disable the Resource Manager, you must alter all maintenance windows to remove this plan. However, use caution, because resource consumption by automated maintenance tasks will no longer be regulated, which may adversely affect the performance of your other sessions. See Managing Automated Database Maintenance Tasks for more information on maintenance windows.

26.3 Assigning Sessions to Resource Consumer Groups

There are automatic and manual methods that database administrators, users, and applications can use to assign sessions to resource consumer groups. When a session is assigned to a resource consumer group, Oracle Database Resource Manager (the Resource Manager) can manage resource allocation for it.

> **Note:**
>
> Sessions that are not assigned to a consumer group are placed in the consumer group `OTHER_GROUPS`.

- Overview of Assigning Sessions to Resource Consumer Groups
  Before you enable the Resource Manager, you must specify how user sessions are assigned to resource consumer groups.
- Assigning an Initial Resource Consumer Group
  The initial consumer group of a session is determined by the mapping rules that you configure.
- Specifying Session-to-Consumer Group Mapping Rules
  You can create and prioritize session-to-consumer group mapping rules.
- Switching Resource Consumer Groups
  You can switch the resource consumer group of a session.
- Specifying Automatic Consumer Group Switching
  You can configure the Resource Manager to automatically switch a session to another consumer group when a certain condition is met.
- Granting and Revoking the Switch Privilege
  A user or application must have the switch privilege to switch a session to a specified resource consumer group.

26.3.1 Overview of Assigning Sessions to Resource Consumer Groups

Before you enable the Resource Manager, you must specify how user sessions are assigned to resource consumer groups.

You do this by creating *mapping rules* that enable the Resource Manager to automatically assign each session to a consumer group upon session startup, based upon session attributes. After a session is assigned to its initial consumer group and is running, you can call a procedure to manually switch the session to a different consumer group. You would typically do this if the session is using excessive resources and must be moved to a consumer group that is more limited in its resource allocation. You can also grant the *switch privilege* to users and to applications so that they can switch their sessions from one consumer group to another.

The database can also automatically switch a session from one consumer group to another (typically lower priority) consumer group when there are changes in session attributes or when a session exceeds designated resource consumption limits.

26.3.2 Assigning an Initial Resource Consumer Group

The initial consumer group of a session is determined by the mapping rules that you configure.

For information on how to configure mapping rules, see "Specifying Session-to-Consumer Group Mapping Rules".

26.3.3 Specifying Session-to-Consumer Group Mapping Rules

You can create and prioritize session-to-consumer group mapping rules.

- About Session-to-Consumer Group Mapping Rules
  You can specify the initial consumer group for a session and dynamically switch the session to a different consumer group if the session attributes change.
- Creating Consumer Group Mapping Rules
  You use the `SET_CONSUMER_GROUP_MAPPING` procedure to map a session attribute/value pair to a consumer group.
- Modifying and Deleting Consumer Group Mapping Rules
  To modify a consumer group mapping rule, run the `SET_CONSUMER_GROUP_MAPPING` procedure against the desired attribute/value pair, specifying a new consumer group.
- Creating Mapping Rule Priorities
  To resolve conflicting mapping rules, you can establish a priority ordering of the session attributes from most important to least important.

26.3.3.1 About Session-to-Consumer Group Mapping Rules

You can specify the initial consumer group for a session and dynamically switch the session to a different consumer group if the session attributes change.

By creating session-to-consumer group mapping rules, you can:

- Specify the initial consumer group for a session based on session attributes.
- Enable the Resource Manager to dynamically switch a running session to another consumer group based on changing session attributes.

The mapping rules are based on session attributes such as the user name, the service that the session used to connect to the database, or the name of the client program.

To resolve conflicts among mapping rules, the Resource Manager orders the rules by priority. For example, suppose user `SCOTT` connects to the database with the `SALES` service. If one mapping rule states that user `SCOTT` starts in the `MED_PRIORITY` consumer group, and another states that sessions that connect with the `SALES` service start in the `HIGH_PRIORITY` consumer group, mapping rule priorities resolve this conflict.

There are two types of session attributes upon which mapping rules are based: login attributes and run-time attributes. The login attributes are meaningful only at session login time, when the Resource Manager determines the initial consumer group of the session. Run-time attributes apply any time during and after session login. You can reassign a logged in session to another consumer group by changing any of its run-time attributes.

You use the `SET_CONSUMER_GROUP_MAPPING` and `SET_CONSUMER_GROUP_MAPPING_PRI` procedures to configure the automatic assignment of sessions to consumer groups. You must use a pending area for these procedures. (You must create the pending area, run the procedures,

optionally validate the pending area, and then submit the pending area. For examples of using the pending area, see "Creating a Complex Resource Plan".)

A session is automatically switched to a consumer group through mapping rules at distinct points in time:

- When the session first logs in, the mapping rules are evaluated to determine the initial group of the session.
- If a session attribute is dynamically changed to a new value (which is only possible for run-time attributes), then the mapping rules are reevaluated, and the session might be switched to another consumer group.

Predefined Consumer Group Mapping Rules

Each Oracle database comes with a set of predefined consumer group mapping rules:

- As described in "About Resource Consumer Groups", all sessions created by user accounts `SYS` or `SYSTEM` are initially mapped to the `SYS_GROUP` consumer group.
- Sessions performing a data load with Data Pump or performing backup or copy operations with RMAN are automatically mapped to the predefined consumer groups designated in Table 26-19.

You can use the `DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING` procedure to modify or delete any of these predefined mapping rules.

> **See Also:**
>
> - "Assigning an Initial Resource Consumer Group"
> - "Specifying Automatic Switching with Mapping Rules"

26.3.3.2 Creating Consumer Group Mapping Rules

You use the `SET_CONSUMER_GROUP_MAPPING` procedure to map a session attribute/value pair to a consumer group.

The parameters for this procedure are the following:

Parameter	Description
`ATTRIBUTE`	The session attribute type, specified as a package constant
`VALUE`	The value of the attribute
`CONSUMER_GROUP`	The consumer group to map to for this attribute/value pair

`ATTRIBUTE` can be one of the following:

Attribute	Type	Description
`ORACLE_USER`	Login	The Oracle Database user name
`SERVICE_NAME`	Login	The database service name used by the client to establish a connection

Attribute	Type	Description
CLIENT_OS_USER	Login	The operating system user name of the client that is logging in
CLIENT_PROGRAM	Login	The name of the client program used to log in to the server
CLIENT_MACHINE	Login	The name of the computer from which the client is making the connection
CLIENT_ID	Login	The client identifier for the session The client identifier session attribute is set by the DBMS_SESSION.SET_IDENTIFIER procedure.
MODULE_NAME	Run-time	The module name in the currently running application as set by the DBMS_APPLICATION_INFO.SET_MODULE procedure or the equivalent OCI attribute setting
MODULE_NAME_ACTION	Run-time	A combination of the current module and the action being performed as set by either of the following procedures or their equivalent OCI attribute setting: • DBMS_APPLICATION_INFO.SET_MODULE • DBMS_APPLICATION_INFO.SET_ACTION The attribute is specified as the module name followed by a period (.), followed by the action name (*module_name.action_name*).
SERVICE_MODULE	Run-time	A combination of service and module names in this form: *service_name.module_name*
SERVICE_MODULE_ACTION	Run-time	A combination of service name, module name, and action name, in this form: *service_name.module_name.action_name*
ORACLE_FUNCTION	Run-time	An RMAN or Data Pump operation. Valid values are DATALOAD, BACKUP, and COPY. There are predefined mappings for each of these values. If your session is performing any of these functions, it is automatically mapped to a predefined consumer group. See Table 26-19 for details.

For example, the following PL/SQL block causes user SCOTT to map to the DEV_GROUP consumer group every time that they log in:

```
BEGIN
  DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING
     (DBMS_RESOURCE_MANAGER.ORACLE_USER, 'SCOTT', 'DEV_GROUP');
END;
/
```

Again, you must create a pending area before running the `SET_CONSUMER_GROUP_MAPPING` procedure.

You can use wildcards for the value of most attributes in the `value` parameter in the `SET_CONSUMER_GROUP_MAPPING` procedure. To specify values with wildcards, use the same semantics as the SQL `LIKE` operator. Specifically, wildcards use the following semantics:

- `%` for a multicharacter wildcard
- _ for a single character wildcard
- \ to escape the wildcards

Wildcards can only be used if the attribute is one of the following:

- `CLIENT_OS_USER`
- `CLIENT_PROGRAM`
- `CLIENT_MACHINE`
- `MODULE_NAME`
- `MODULE_NAME_ACTION`
- `SERVICE_MODULE`
- `SERVICE_MODULE_ACTION`

26.3.3.3 Modifying and Deleting Consumer Group Mapping Rules

To modify a consumer group mapping rule, run the `SET_CONSUMER_GROUP_MAPPING` procedure against the desired attribute/value pair, specifying a new consumer group.

To delete a rule, run the `SET_CONSUMER_GROUP_MAPPING` procedure against the desired attribute/value pair and specify a `NULL` consumer group.

26.3.3.4 Creating Mapping Rule Priorities

To resolve conflicting mapping rules, you can establish a priority ordering of the session attributes from most important to least important.

You use the `SET_CONSUMER_GROUP_MAPPING_PRI` procedure to set the priority of each attribute to a unique integer from 1 (most important) to 12 (least important). The following example illustrates this setting of priorities:

```
BEGIN
  DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING_PRI(
    EXPLICIT => 1,
    SERVICE_MODULE_ACTION => 2,
    SERVICE_MODULE => 3,
    MODULE_NAME_ACTION => 4,
    MODULE_NAME => 5,
    SERVICE_NAME => 6,
    ORACLE_USER => 7,
    CLIENT_PROGRAM => 8,
    CLIENT_OS_USER => 9,
```

```
    CLIENT_MACHINE => 10,
    CLIENT_ID => 11);
END;
/
```

In this example, the priority of the database user name is set to 7 (less important), while the priority of the module name is set to 5 (more important).

> **Note:**
>
> `SET_CONSUMER_GROUP_MAPPING_PRI` requires that you include the pseudo-attribute `EXPLICIT` as an argument. It must be set to 1. It indicates that explicit consumer group switches have the highest priority. You explicitly switch consumer groups with these package procedures, which are described in detail in *Oracle Database PL/SQL Packages and Types Reference*:
>
> - `DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP`
> - `DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS`
> - `DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_USER`

To illustrate how mapping rule priorities work, continuing with the previous example, assume that in addition to the mapping of user `SCOTT` to the `DEV_GROUP` consumer group, there is also a module name mapping rule as follows:

```
BEGIN
  DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING
     (DBMS_RESOURCE_MANAGER.MODULE_NAME, 'EOD_REPORTS', 'LOW_PRIORITY');
END;
/
```

Now if the application in user `SCOTT`'s session sets its module name to `EOD_REPORTS`, the session is reassigned to the `LOW_PRIORITY` consumer group, because module name mapping has a higher priority than database user mapping.

You can query the view `DBA_RSRC_MAPPING_PRIORITY` to see the current priority ordering of session attributes.

> **See Also:**
>
> - *Oracle Database PL/SQL Packages and Types Reference* for information about setting the module name with the `DBMS_APPLICATION_INFO.SET_MODULE` procedure
> - "Granting and Revoking the Switch Privilege"

26.3.4 Switching Resource Consumer Groups

You can switch the resource consumer group of a session.

- Manually Switching Resource Consumer Groups
  You can change the resource consumer group of running sessions.

- Enabling Users or Applications to Manually Switch Consumer Groups
  You can grant a user the switch privilege so that they can switch their current consumer group using the SWITCH_CURRENT_CONSUMER_GROUP procedure in the DBMS_SESSION package.

26.3.4.1 Manually Switching Resource Consumer Groups

You can change the resource consumer group of running sessions.

- About Manually Switching Resource Consumer Groups
  The DBMS_RESOURCE_MANAGER PL/SQL package provides two procedures that enable you to change the resource consumer group of running sessions.
- Switching a Single Session
  The SWITCH_CONSUMER_GROUP_FOR_SESS procedure causes the specified session to immediately be moved into the specified resource consumer group. In effect, this procedure can raise or lower priority of the session.
- Switching All Sessions for a User
  The SWITCH_CONSUMER_GROUP_FOR_USER procedure changes the resource consumer group for all sessions pertaining to the specified user name.

26.3.4.1.1 About Manually Switching Resource Consumer Groups

The DBMS_RESOURCE_MANAGER PL/SQL package provides two procedures that enable you to change the resource consumer group of running sessions.

Both of these procedures can also change the consumer group of any parallel execution server sessions associated with the coordinator session. The changes made by these procedures pertain to current sessions only; they are not persistent. They also do not change the initial consumer groups for users.

Instead of terminating a session of a user who is using excessive CPU, you can change that user's consumer group to one that is allocated fewer resources.

26.3.4.1.2 Switching a Single Session

The SWITCH_CONSUMER_GROUP_FOR_SESS procedure causes the specified session to immediately be moved into the specified resource consumer group. In effect, this procedure can raise or lower priority of the session.

The SWITCH_CONSUMER_GROUP_FOR_SESS procedure is Oracle Real Application Clusters (Oracle RAC) instance specific. You must connect to the pluggable database in the same Oracle RAC instance where the session to be switched is running and then run this procedure.

The following PL/SQL block switches a specific session to a new consumer group. The session identifier (SID) is 17, the session serial number (SERIAL#) is 12345, and the new consumer group is the HIGH_PRIORITY consumer group.

```
BEGIN
  DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_SESS ('17', '12345',
   'HIGH_PRIORITY');
END;
/
```

The SID, session serial number, and current resource consumer group for a session are viewable using the V$SESSION view.

> **See Also:**
>
> *Oracle Database Reference* for details about the `V$SESSION` view.

26.3.4.1.3 Switching All Sessions for a User

The `SWITCH_CONSUMER_GROUP_FOR_USER` procedure changes the resource consumer group for all sessions pertaining to the specified user name.

The following PL/SQL block switches all sessions that belong to user `HR` to the `LOW_GROUP` consumer group:

```
BEGIN
  DBMS_RESOURCE_MANAGER.SWITCH_CONSUMER_GROUP_FOR_USER ('HR',
    'LOW_GROUP');
END;
/
```

26.3.4.2 Enabling Users or Applications to Manually Switch Consumer Groups

You can grant a user the switch privilege so that they can switch their current consumer group using the `SWITCH_CURRENT_CONSUMER_GROUP` procedure in the `DBMS_SESSION` package.

A user can run this procedure from an interactive session, for example from SQL*Plus, or an application can call this procedure to switch its session, effectively dynamically changing its priority.

The `SWITCH_CURRENT_CONSUMER_GROUP` procedure enables users to switch to only those consumer groups for which they have the switch privilege. If the caller is another procedure, then this procedure enables users to switch to a consumer group for which the owner of that procedure has switch privileges.

The parameters for this procedure are the following:

Parameter	Description
`NEW_CONSUMER_GROUP`	The consumer group to which the user is switching.
`OLD_CONSUMER_GROUP`	Returns the name of the consumer group from which the user switched. Can be used to switch back later.
`INITIAL_GROUP_ON_ERROR`	Controls behavior if a switching error occurs. If `TRUE`, in the event of an error, the user is switched to the initial consumer group. If `FALSE`, raises an error.

The following SQL*Plus session illustrates switching to a new consumer group. By printing the value of the output parameter `old_group`, the example illustrates how the old consumer group name is saved.

```
SET serveroutput on
DECLARE
    old_group varchar2(30);
BEGIN
  DBMS_SESSION.SWITCH_CURRENT_CONSUMER_GROUP('BATCH_GROUP', old_group, FALSE);
  DBMS_OUTPUT.PUT_LINE('OLD GROUP = ' || old_group);
```

```
END;
/
```

The following line is output:

```
OLD GROUP = OLTP_GROUP
```

Note that the Resource Manager considers a switch to have taken place even if the `SWITCH_CURRENT_CONSUMER_GROUP` procedure is called to switch the session to the consumer group that it is already in.

> **Note:**
>
> The Resource Manager also works in environments where a generic database user name is used to log on to an application. The `DBMS_SESSION` package can be called to switch the consumer group assignment of a session at session startup, or as particular modules are called.

> **See Also:**
>
> - "Granting and Revoking the Switch Privilege"
> - *Oracle Database PL/SQL Packages and Types Reference* for additional examples and more information about the `DBMS_SESSION` package

26.3.5 Specifying Automatic Consumer Group Switching

You can configure the Resource Manager to automatically switch a session to another consumer group when a certain condition is met.

Automatic switching can occur when: a session attribute changes, causing a new mapping rule to take effect, or a session exceeds the CPU, physical I/O, or logical I/O resource consumption limits set by its consumer group, or it exceeds the elapsed time limit set by its consumer group.

- Specifying Automatic Switching with Mapping Rules
  If a session attribute changes while the session is running, then the session-to-consumer group mapping rules are reevaluated. If a new rule takes effect, then the session might be moved to a different consumer group.
- Specifying Automatic Switching by Setting Resource Limits
  You can manage runaway sessions or calls that use CPU, physical I/O, or logical I/O resources beyond a specified limit. A runaway session is a SQL query, while a runaway call is a PL/SQL call.

26.3.5.1 Specifying Automatic Switching with Mapping Rules

If a session attribute changes while the session is running, then the session-to-consumer group mapping rules are reevaluated. If a new rule takes effect, then the session might be moved to a different consumer group.

See "Specifying Session-to-Consumer Group Mapping Rules" for more information.

26.3.5.2 Specifying Automatic Switching by Setting Resource Limits

You can manage runaway sessions or calls that use CPU, physical I/O, or logical I/O resources beyond a specified limit. A runaway session is a SQL query, while a runaway call is a PL/SQL call.

When you create a resource plan directive for a consumer group, you can specify limits for CPU, physical I/O, or logical I/O resource consumption for sessions in that group. You can specify limits for physical I/O and logical I/O separately. You can also specify a limit for elapsed time. If the `SWITCH_FOR_CALL` resource plan directive is set to `FALSE`, then Resource Manager enforces these limits from the start of the session. If the `SWITCH_FOR_CALL` resource plan directive is set to `TRUE`, then Resource Manager enforces these limits from the start of the SQL operation or PL/SQL block.

You can then specify the action that is to be taken if any single session or call exceeds one of these limits. The possible actions are the following:

- The session is dynamically switched to a designated consumer group.

  The target consumer group is typically one that has lower resource allocations.
- The session is terminated.
- The session's current SQL statement is terminated.
- Information about the session is logged, but no other action is taken for the session.

The following are the resource plan directive attributes that are involved in this type of automatic session switching.

- `SWITCH_GROUP`
- `SWITCH_TIME`
- `SWITCH_ESTIMATE`
- `SWITCH_IO_MEGABYTES`
- `SWITCH_IO_REQS`
- `SWITCH_FOR_CALL`
- `SWITCH_IO_LOGICAL`
- `SWITCH_ELAPSED_TIME`

See "Creating Resource Plan Directives " for descriptions of these attributes.

Switches occur for sessions that are running and consuming resources, not waiting for user input or waiting for CPU cycles. After a session is switched, it continues in the target consumer group until it becomes idle, at which point it is switched back to its original consumer group. However, if `SWITCH_FOR_CALL` is set to `TRUE`, then the Resource Manager does not wait until the session is idle to return it to its original resource consumer group. Instead, the session is returned when the current top-level call completes. A **top-level call** in PL/SQL is an entire PL/SQL block treated as one call. A top-level call in SQL is an individual SQL statement.

`SWITCH_FOR_CALL` is useful for three-tier applications where the middle tier server is using session pooling.

A switched session is allowed to continue running even if the active session pool for the new group is full. Under these conditions, a consumer group can have more sessions running than specified by its active session pool.

When SWITCH_FOR_CALL is FALSE, the Resource Manager views a session as idle if a certain amount of time passes between calls. This time interval is a few seconds and is not configurable.

The following are examples of automatic switching based on resource limits. You must create a pending area before running these examples.

Example 1

The following PL/SQL block creates a resource plan directive for the OLTP group that switches any session in that group to the LOW_GROUP consumer group if a call in the sessions exceeds 5 seconds of CPU time. This example prevents unexpectedly long queries from consuming too many resources. The switched-to consumer group is typically one with lower resource allocations.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN             => 'DAYTIME',
   GROUP_OR_SUBPLAN => 'OLTP',
   COMMENT          => 'OLTP group',
   MGMT_P1          => 75,
   SWITCH_GROUP     => 'LOW_GROUP',
   SWITCH_TIME      => 5);
END;
/
```

Example 2

The following PL/SQL block creates a resource plan directive for the OLTP group that temporarily switches any session in that group to the LOW_GROUP consumer group if the session exceeds 10,000 physical I/O requests or exceeds 2,500 Megabytes of data transferred. The session is returned to its original group after the offending top call is complete.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN                => 'DAYTIME',
   GROUP_OR_SUBPLAN    => 'OLTP',
   COMMENT             => 'OLTP group',
   MGMT_P1             => 75,
   SWITCH_GROUP        => 'LOW_GROUP',
   SWITCH_IO_REQS      => 10000,
   SWITCH_IO_MEGABYTES => 2500,
   SWITCH_FOR_CALL     => TRUE);
END;
/
```

Example 3

The following PL/SQL block creates a resource plan directive for the REPORTING group that terminates any session that exceeds 60 seconds of CPU time. This example prevents runaway queries from consuming too many resources.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN             => 'DAYTIME',
   GROUP_OR_SUBPLAN => 'REPORTING',
   COMMENT          => 'Reporting group',
   MGMT_P1          => 75,
   SWITCH_GROUP     => 'KILL_SESSION',
   SWITCH_TIME      => 60);
```

```
END;
/
```

In this example, the reserved consumer group name `KILL_SESSION` is specified for `SWITCH_GROUP`. Therefore, the session is terminated when the switch criteria is met. Other reserved consumer group names are `CANCEL_SQL` and `LOG_ONLY`. When `CANCEL_SQL` is specified, the current call is canceled when switch criteria are met, but the session is not terminated. When `LOG_ONLY` is specified, information about the session is recorded in real-time SQL monitoring, but no specific action is taken for the session.

Example 4

The following PL/SQL block creates a resource plan directive for the `OLTP` group that temporarily switches any session in that group to the `LOW_GROUP` consumer group if the session exceeds 100 logical I/O requests. The session is returned to its original group after the offending top call is complete.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN                 => 'DAYTIME',
   GROUP_OR_SUBPLAN     => 'OLTP',
   COMMENT              => 'OLTP group',
   MGMT_P1              => 75,
   SWITCH_GROUP         => 'LOW_GROUP',
   SWITCH_IO_LOGICAL    => 100,
   SWITCH_FOR_CALL      => TRUE);
END;
/
```

Example 5

The following PL/SQL block creates a resource plan directive for the `OLTP` group that temporarily switches any session in that group to the `LOW_GROUP` consumer group if a call in a session exceeds five minutes (300 seconds). The session is returned to its original group after the offending top call is complete.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN                 => 'DAYTIME',
   GROUP_OR_SUBPLAN     => 'OLTP',
   COMMENT              => 'OLTP group',
   MGMT_P1              => 75,
   SWITCH_GROUP         => 'LOW_GROUP',
   SWITCH_FOR_CALL      => TRUE,
   SWITCH_ELAPSED_TIME => 300);
END;
/
```

See Also:

- "Creating Resource Plan Directives "
- "What Solutions Does the Resource Manager Provide for Workload Management?" for information about logical I/O

26.3.6 Granting and Revoking the Switch Privilege

A user or application must have the switch privilege to switch a session to a specified resource consumer group.

- About Granting and Revoking the Switch Privilege
  Using the DBMS_RESOURCE_MANAGER_PRIVS PL/SQL package, you can grant or revoke the switch privilege to a user, role, or PUBLIC. The switch privilege enables a user or application to switch a session to a specified resource consumer group.
- Granting the Switch Privilege
  You can grant a user the privilege to switch to a specific consumer group using the GRANT_SWITCH_CONSUMER_GROUP procedure.
- Revoking Switch Privileges
  You can revoke a user's privilege to switch to a specific consumer group using the REVOKE_SWITCH_CONSUMER_GROUP procedure.

26.3.6.1 About Granting and Revoking the Switch Privilege

Using the DBMS_RESOURCE_MANAGER_PRIVS PL/SQL package, you can grant or revoke the switch privilege to a user, role, or PUBLIC. The switch privilege enables a user or application to switch a session to a specified resource consumer group.

The package also enables you to revoke the switch privilege. The relevant package procedures are listed in the following table.

Procedure	Description
GRANT_SWITCH_CONSUMER_GROUP	Grants permission to a user, role, or PUBLIC to switch to a specified resource consumer group.
REVOKE_SWITCH_CONSUMER_GROUP	Revokes permission for a user, role, or PUBLIC to switch to a specified resource consumer group.

OTHER_GROUPS has switch privileges granted to PUBLIC. Therefore, all users are automatically granted the switch privilege for this consumer group.

The following switches do not require explicit switch privilege:

- There is a consumer group mapping specified by the SET_CONSUMER_GROUP_MAPPING procedure in the DBMS_RESOURCE_MANAGER package, and a session is switching to a different consumer group due to the mapping. See "Creating Consumer Group Mapping Rules".
- There is an automatic consumer group switch when a switch condition is met based on the setting of the switch_group parameter of a resource plan directive.

Explicit switch privilege is required for a user to switch a session to a consumer group in all other cases.

> **See Also:**
>
> - "Enabling Users or Applications to Manually Switch Consumer Groups"
> - "Specifying Automatic Consumer Group Switching"

26.3.6.2 Granting the Switch Privilege

You can grant a user the privilege to switch to a specific consumer group using the `GRANT_SWITCH_CONSUMER_GROUP` procedure.

The following example grants user `SCOTT` the privilege to switch to consumer group `OLTP`.

```
BEGIN
  DBMS_RESOURCE_MANAGER_PRIVS.GRANT_SWITCH_CONSUMER_GROUP (
   GRANTEE_NAME   => 'SCOTT',
   CONSUMER_GROUP => 'OLTP',
   GRANT_OPTION   =>  TRUE);
END;
/
```

User `SCOTT` is also granted permission to grant switch privileges for `OLTP` to others.

If you grant permission to a role to switch to a particular resource consumer group, then any user who is granted that role and has enabled that role can switch their session to that consumer group.

If you grant `PUBLIC` the permission to switch to a particular consumer group, then any user can switch to that group.

If the `GRANT_OPTION` argument is `TRUE`, then users granted switch privilege for the consumer group can also grant switch privileges for that consumer group to others.

26.3.6.3 Revoking Switch Privileges

You can revoke a user's privilege to switch to a specific consumer group using the `REVOKE_SWITCH_CONSUMER_GROUP` procedure.

The following example revokes user `SCOTT`'s privilege to switch to consumer group `OLTP`.

```
BEGIN
  DBMS_RESOURCE_MANAGER_PRIVS.REVOKE_SWITCH_CONSUMER_GROUP (
   REVOKEE_NAME   => 'SCOTT',
   CONSUMER_GROUP => 'OLTP');
END;
/
```

If you revoke a user's switch privileges for a particular consumer group, any subsequent attempts by that user to switch to that consumer group manually will fail. The user's session will then be automatically assigned to `OTHER_GROUPS`.

If you revoke from a role the switch privileges to a consumer group, any users who had switch privileges for the consumer group only through that role are no longer able to switch to that consumer group.

If you revoke switch privileges to a consumer group from `PUBLIC`, any users other than those who are explicitly assigned switch privileges either directly or through a role are no longer able to switch to that consumer group.

26.4 Managing Resource Plans

Resource Manager allocates resources to pluggable databases (PDBs) in a multitenant container database (CDB).

This chapter assumes that you meet the following prerequisites:

- You understand how to configure and manage a CDB.

> **Note:**
>
> - You can complete the tasks in this chapter using SQL*Plus or Oracle SQL Developer.
> - You can also administer the Resource Manager with the graphical user interface of Oracle Enterprise Manager Cloud Control (Cloud Control).
> - For simplicity, this chapter refers to PDBs, application roots, and application PDBs as "PDBs."

- Managing CDB Resource Plans
  In a CDB, PDBs might have different levels of priority. You can create CDB resource plans to distribute resources to different PDBs based on these priorities.
- Managing PDB Resource Plans
  You can create, enable, and modify resource plans for individual PDBs.
- Creating a Simple Resource Plan
  You can quickly create a simple resource plan that is adequate for many situations using the `CREATE_SIMPLE_PLAN` procedure.
- Creating a Complex Resource Plan
  When your situation calls for a more complex resource plan, you must create the plan, with its directives and consumer groups, in a staging area called the pending area, and then validate the plan before storing it in the data dictionary.

26.4.1 Managing CDB Resource Plans

In a CDB, PDBs might have different levels of priority. You can create CDB resource plans to distribute resources to different PDBs based on these priorities.

- Creating a CDB Resource Plan for Managing PDBs
  To create a CDB resource plan for individual PDBs and define the directives for the plan, use the `DBMS_RESOURCE_MANAGER` package.
- Creating a CDB Resource Plan for Managing PDBs: Scenario
  This scenario illustrates each of the steps involved in creating a CDB resource plan for individual PDBs.

- Creating a CDB Resource Plan with PDB Performance Profiles
  Use the DBMS_RESOURCE_MANAGER package to create a CDB resource plan for PDB performance profiles and define the directives for the plan. Each PDB that uses a profile adopts the CDB resource plan directive.
- Creating a CDB Resource Plan for PDB Performance Profiles: Scenario
  This scenario illustrates the steps involved in creating a CDB resource plan for PDB performance profiles.
- Enabling a CDB Resource Plan
  You enable the Resource Manager for a CDB by setting the RESOURCE_MANAGER_PLAN initialization parameter in the root.
- Modifying a CDB Resource Plan
  Modifying a CDB resource plan includes tasks such as updating the plan, creating, updating, or deleting plan directives for PDBs, and updating default directives.
- Disabling a CDB Resource Plan
  Disable the Resource Manager for a CDB by unsetting the RESOURCE_MANAGER_PLAN initialization parameter in the CDB root.
- Viewing Information About Plans and Directives in a CDB
  You can view information about CDB resource plans, CDB resource plan directives, and predefined resource plans in a CDB.

26.4.1.1 Creating a CDB Resource Plan for Managing PDBs

To create a CDB resource plan for individual PDBs and define the directives for the plan, use the DBMS_RESOURCE_MANAGER package.

The general steps for creating a CDB resource plan for individual PDBs are the following:

1. Create the pending area using the CREATE_PENDING_AREA procedure.
2. Create the CDB resource plan using the CREATE_CDB_PLAN procedure.
3. Create directives for the PDBs using the CREATE_CDB_PLAN_DIRECTIVE procedure.
4. (Optional) Update the default PDB directive using the UPDATE_CDB_DEFAULT_DIRECTIVE procedure.
5. Validate the pending area using the VALIDATE_PENDING_AREA procedure.
6. Submit the pending area using the SUBMIT_PENDING_AREA procedure.

26.4.1.2 Creating a CDB Resource Plan for Managing PDBs: Scenario

This scenario illustrates each of the steps involved in creating a CDB resource plan for individual PDBs.

The scenario assumes that you want to create a CDB resource plan for a CDB named newcdb. The plan includes a directive for each PDB. In this scenario, you also update the default directive and the AutoTask directive.

The directives are defined using various procedures in the DBMS_RESOURCE_MANAGER package. The attributes of each directive are defined using parameters in these procedures. Table 26-12 describes the types of directives in the plan.

Table 26-12 Attributes for PDB Directives in a CDB Resource Plan

Directive Attribute	Description	See Also
`shares`	Resource allocation share for CPU and parallel execution server resources.	"Shares for Allocating Resources to PDBs"
`utilization_limit`	Resource utilization limit for CPU.	"Utilization Limits for PDBs"
`parallel_server_limit`	Maximum percentage of parallel execution servers that a PDB can use before queuing parallel statements. When the `parallel_server_limit` directive is specified for a PDB, the limit is the `PARALLEL_SERVERS_TARGET` value of the CDB root multiplied by the value of the `parallel_server_limit` parameter in the `CREATE_CDB_PLAN_DIRECTIVE` procedure. **Note:** Oracle recommends using the `PARALLEL_SERVERS_TARGET` initialization parameter instead of the `parallel_server_limit` directive in a CDB plan.	"Utilization Limits for PDBs"

Table 26-13 describes how the CDB resource plan allocates resources to its PDBs using the directive attributes described in Table 26-12.

Table 26-13 Sample Directives for PDBs in a CDB Resource Plan

PDB	shares Directive	utilization_limit Directive	parallel_server_limit Directive
`salespdb`	3	Unlimited	Unlimited
`servicespdb`	3	Unlimited	Unlimited
`hrpdb`	1	70	70
Default	1	50	50
AutoTask	1	75	75

The `salespdb` and `servicespdb` PDBs are more important than the other PDBs in the CDB. Therefore, they get a higher share (3), unlimited CPU utilization resource, and unlimited parallel execution server resource.

The default directive applies to PDBs for which specific directives have not been defined. For this scenario, assume that the CDB has several PDBs that use the default directive. This scenario updates the default directive.

In addition, this scenario updates the AutoTask directive. The AutoTask directive applies to automatic maintenance tasks that are run in the root maintenance window.

To create a CDB resource plan:

1. Create a pending area using the `CREATE_PENDING_AREA` procedure:

```
exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```

2. Create a CDB resource plan named `newcdb_plan` using the `CREATE_CDB_PLAN` procedure:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN(
    plan    => 'newcdb_plan',
    comment => 'CDB resource plan for newcdb');
END;
/
```

3. Create the CDB resource plan directives for the PDBs using the `CREATE_CDB_PLAN_DIRECTIVE` procedure. Each directive specifies how resources are allocated to a specific PDB.

   Table 26-13 describes the directives for the `salespdb`, `servicespdb`, and `hrpdb` PDBs in this scenario. Run the following procedures to create these directives:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE(
    plan                 => 'newcdb_plan',
    pluggable_database   => 'salespdb',
    shares               => 3,
    utilization_limit    => 100,
    parallel_server_limit => 100);
END;
/

BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE(
    plan                 => 'newcdb_plan',
    pluggable_database   => 'servicespdb',
    shares               => 3,
    utilization_limit    => 100,
    parallel_server_limit => 100);
END;
/

BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE(
    plan                 => 'newcdb_plan',
    pluggable_database   => 'hrpdb',
    shares               => 1,
    utilization_limit    => 70,
    parallel_server_limit => 70);
END;
/
```

   All other PDBs in this CDB use the default PDB directive.

4. If the current default CDB resource plan directive for PDBs does not meet your requirements, then update the directive using the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure.

   The default directive applies to PDBs for which specific directives have not been defined. See "The Default Directive for PDBs" for more information.

   Table 26-13 describes the default directive that PDBs use in this scenario. Run the following procedure to update the default directive:

```
BEGIN
  DBMS_RESOURCE_MANAGER.UPDATE_CDB_DEFAULT_DIRECTIVE(
    plan                      => 'newcdb_plan',
    new_shares                => 1,
    new_utilization_limit     => 50,
    new_parallel_server_limit => 50);
END;
/
```

5. Validate the pending area using the `VALIDATE_PENDING_AREA` procedure:

```
exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
```

6. Submit the pending area using the `SUBMIT_PENDING_AREA` procedure:

```
exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

26.4.1.3 Creating a CDB Resource Plan with PDB Performance Profiles

Use the `DBMS_RESOURCE_MANAGER` package to create a CDB resource plan for PDB performance profiles and define the directives for the plan. Each PDB that uses a profile adopts the CDB resource plan directive.

The general steps for creating a CDB resource plan with PDB performance profiles are the following:

1. Create the pending area using the `CREATE_PENDING_AREA` procedure.
2. Create the CDB resource plan using the `CREATE_CDB_PLAN` procedure.
3. Create directives for the PDB performance profiles using the `CREATE_CDB_PROFILE_DIRECTIVE` procedure.
4. (Optional) Update the default PDB directive using the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure.
5. Validate the pending area using the `VALIDATE_PENDING_AREA` procedure.
6. Submit the pending area using the `SUBMIT_PENDING_AREA` procedure.
7. For each PDB that will use a profile, set the `DB_PERFORMANCE_PROFILE` initialization parameter and specify the profile name.

26.4.1.4 Creating a CDB Resource Plan for PDB Performance Profiles: Scenario

This scenario illustrates the steps involved in creating a CDB resource plan for PDB performance profiles.

The scenario assumes that you want to create a CDB resource plan for a CDB named `newcdb`. The plan includes a directive for each PDB performance profile. In this scenario, you also update the default directive and the AutoTask directive.

In the CDB resource plan, you give each profile a name. In each PDB, you set the `DB_PERFORMANCE_PROFILE` initialization parameter to specify which PDB performance profile the PDB uses.

The directives are defined using various procedures in the `DBMS_RESOURCE_MANAGER` package. The attributes of each directive are defined using parameters in these procedures. The following table describes the types of directives in the plan.

Table 26-14 Attributes for PDB Performance Profile Directives in a CDB Resource Plan

Directive Attribute	Description	See Also
`shares`	Resource allocation share for CPU and parallel execution server resources.	"Shares for Allocating Resources to PDBs"
`utilization_limit`	Resource utilization limit for CPU.	"Utilization Limits for PDBs"
`parallel_server_limit`	Maximum percentage of parallel execution servers that a PDB can use. When the `parallel_server_limit` directive is specified for a PDB performance profile, the limit is the value of the `PARALLEL_SERVERS_TARGET` initialization parameter setting in the CDB root multiplied by the value of the `parallel_server_limit` parameter in the `CREATE_CDB_PROFILE_DIRECTIVE` procedure.	"Utilization Limits for PDBs"

The following table describes how the CDB resource plan allocates resources to its PDB performance profiles using the directive attributes described in Table 26-14.

Table 26-15 Sample Directives for PDB Performance Profiles in a CDB Resource Plan

PDB	shares Directive	utilization_limit Directive	parallel_server_limit Directive
`gold`	3	Unlimited	Unlimited
`silver`	2	40	40
`bronze`	1	20	20
Default	1	10	10
AutoTask	2	60	60

The default directive applies to PDBs for which specific directives have not been defined. For this scenario, assume that the CDB has several PDBs that use the default directive. This scenario updates the default directive.

In addition, this scenario updates the AutoTask directive. The AutoTask directive applies to automatic maintenance tasks that are run in the root maintenance window.

To create a CDB resource plan for PDB performance profiles:

1. For each PDB that will use a profile, set the `DB_PERFORMANCE_PROFILE` initialization parameter to the name of the profile that the PDB will use.
   a. Run an `ALTER SYSTEM` statement to set the parameter.

      For example, with the PDB as the current container, run the following SQL statement:

      ```
      ALTER SYSTEM SET DB_PERFORMANCE_PROFILE=gold SCOPE=spfile;
      ```

   b. Close the PDB:

      ```
      ALTER PLUGGABLE DATABASE CLOSE IMMEDIATE;
      ```

   c. Open the PDB:

      ```
      ALTER PLUGGABLE DATABASE OPEN;
      ```

2. Create a pending area using the `CREATE_PENDING_AREA` procedure:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Create a CDB resource plan named `newcdb_plan` using the `CREATE_CDB_PLAN` procedure:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN(
       plan    => 'newcdb_plan',
       comment => 'CDB resource plan for newcdb');
   END;
   /
   ```

4. Create the CDB resource plan directives for the PDBs using the `CREATE_CDB_PLAN_DIRECTIVE` procedure. Each directive specifies how resources are allocated to a specific PDB.

   Table 26-13 describes the directives for the `gold`, `silver`, and `bronze` profiles in this scenario. Run the following procedures to create these directives:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.CREATE_CDB_PROFILE_DIRECTIVE(
       plan                 => 'newcdb_plan',
       profile              => 'gold',
       shares               => 3,
       utilization_limit    => 100,
       parallel_server_limit => 100);
   END;
   /

   BEGIN
   ```

```
  DBMS_RESOURCE_MANAGER.CREATE_CDB_PROFILE_DIRECTIVE(
    plan                  => 'newcdb_plan',
    profile               => 'silver',
    shares                => 2,
    utilization_limit     => 40,
    parallel_server_limit => 40);
END;
/

BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CDB_PROFILE_DIRECTIVE(
    plan                  => 'newcdb_plan',
    profile               => 'bronze',
    shares                => 1,
    utilization_limit     => 20,
    parallel_server_limit => 20);
END;
/
```

All other PDBs in this CDB use the default PDB directive.

5. If the current default CDB resource plan directive for PDBs does not meet your requirements, then update the directive using the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure.

   The default directive applies to PDBs for which specific directives have not been defined.

   Table 26-13 describes the default directive that PDBs use in this scenario. Run the following procedure to update the default directive:

```
BEGIN
  DBMS_RESOURCE_MANAGER.UPDATE_CDB_DEFAULT_DIRECTIVE(
    plan                      => 'newcdb_plan',
    new_shares                => 1,
    new_utilization_limit     => 10,
    new_parallel_server_limit => 10);
END;
/
```

6. Validate the pending area using the `VALIDATE_PENDING_AREA` procedure:

```
exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
```

7. Submit the pending area using the `SUBMIT_PENDING_AREA` procedure:

```
exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

> **See Also:**
>
> "The Default Directive for PDBs"

26.4.1.5 Enabling a CDB Resource Plan

You enable the Resource Manager for a CDB by setting the `RESOURCE_MANAGER_PLAN` initialization parameter in the root.

This parameter specifies the top plan, which is the plan to be used for the current CDB instance. If no plan is specified with this parameter, then the Resource Manager is not enabled.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To enable a CDB resource plan:

1. In SQL*Plus, ensure that the current container is the root.
2. Perform one of the following actions:
   - Use an `ALTER SYSTEM` statement to set the `RESOURCE_MANAGER_PLAN` initialization parameter to the CDB resource plan.

     The following example sets the CDB resource plan to `newcdb_plan` using an `ALTER SYSTEM` statement:

     ```
     ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = 'newcdb_plan';
     ```

   - In a text initialization parameter file, set the `RESOURCE_MANAGER_PLAN` initialization parameter to the CDB resource plan, and restart the CDB.

     The following example sets the CDB resource plan to `newcdb_plan` in an initialization parameter file:

     ```
     RESOURCE_MANAGER_PLAN = 'newcdb_plan'
     ```

> **See Also:**
>
> - *Oracle Multitenant Administrator's Guide* for information about accessing a container in a CDB
> - Oracle Scheduler Concepts to learn how to schedule a CDB resource plan change with Oracle Scheduler

26.4.1.6 Modifying a CDB Resource Plan

Modifying a CDB resource plan includes tasks such as updating the plan, creating, updating, or deleting plan directives for PDBs, and updating default directives.

- Updating a CDB Resource Plan
  You can update a CDB resource plan to change its comment using the `UPDATE_CDB_PLAN` procedure.

- Managing CDB Resource Plan Directives for a PDB
  You can create, update, and delete CDB resource plan directives for a PDB.
- Managing CDB Resource Plan Directives for a PDB Performance Profile
  You can create, update, and delete CDB resource plan directives for a PDB performance profile.
- Updating the Default Directive for PDBs in a CDB Resource Plan
  You can update the default directive for PDBs in a CDB resource plan using the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure. The default directive applies to PDBs for which specific directives have not been defined.
- Updating the Default Directive for Maintenance Tasks in a CDB Resource Plan
  You can update the AutoTask directive in a CDB resource plan using the `UPDATE_CDB_AUTOTASK_DIRECTIVE` procedure. The AutoTask directive applies to automatic maintenance tasks that are run in the root maintenance window.
- Deleting a CDB Resource Plan
  You can delete a CDB resource plan using the `DELETE_CDB_PLAN` procedure.

26.4.1.6.1 Updating a CDB Resource Plan

You can update a CDB resource plan to change its comment using the `UPDATE_CDB_PLAN` procedure.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To update a CDB resource plan:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

```
exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```

3. Run the `UPDATE_CDB_PLAN` procedure, and enter a new comment in the `new_comment` parameter.

   For example, the following procedure changes the comment for the `newcdb_plan` CDB resource plan:

```
BEGIN
  DBMS_RESOURCE_MANAGER.UPDATE_CDB_PLAN(
    plan        => 'newcdb_plan',
    new_comment => 'CDB plan for PDBs in newcdb');
END;
/
```

4. Validate the pending area:

```
exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
```

5. Submit the pending area:

```
exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

See Also:

- "About CDB Resource Plans"
- *Oracle Multitenant Administrator's Guide*

26.4.1.6.2 Managing CDB Resource Plan Directives for a PDB

You can create, update, and delete CDB resource plan directives for a PDB.

- Creating New CDB Resource Plan Directives for a PDB
  When you create a PDB in a CDB, you can create a CDB resource plan directive for the PDB using the `CREATE_CDB_PLAN_DIRECTIVE` procedure. The directive specifies how resources are allocated to the new PDB.
- Updating CDB Resource Plan Directives for a PDB
  You can update the CDB resource plan directive for a PDB using the `UPDATE_CDB_PLAN_DIRECTIVE` procedure. The directive specifies how resources are allocated to the PDB.
- Deleting CDB Resource Plan Directives for a PDB
  You can delete the CDB resource plan directive for a PDB using the `DELETE_CDB_PLAN_DIRECTIVE` procedure.

26.4.1.6.2.1 Creating New CDB Resource Plan Directives for a PDB

When you create a PDB in a CDB, you can create a CDB resource plan directive for the PDB using the `CREATE_CDB_PLAN_DIRECTIVE` procedure. The directive specifies how resources are allocated to the new PDB.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To create a new CDB resource plan directive for a PDB:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

```
exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```

3. Run the `CREATE_CDB_PLAN_DIRECTIVE` procedure, and specify the appropriate values for the new PDB.

For example, the following procedure allocates resources to a PDB named `operpdb` in the `newcdb_plan` CDB resource plan:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CDB_PLAN_DIRECTIVE(
    plan                  => 'newcdb_plan',
    pluggable_database    => 'operpdb',
    shares                => 1,
    utilization_limit     => 20,
    parallel_server_limit => 30);
END;
/
```

4. Validate the pending area:

```
exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
```

5. Submit the pending area:

```
exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

See Also:

- "About CDB Resource Plans"
- *Oracle Multitenant Administrator's Guide*

26.4.1.6.2.2 Updating CDB Resource Plan Directives for a PDB

You can update the CDB resource plan directive for a PDB using the `UPDATE_CDB_PLAN_DIRECTIVE` procedure. The directive specifies how resources are allocated to the PDB.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To update a CDB resource plan directive for a PDB:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

```
exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```

3. Run the `UPDATE_CDB_PLAN_DIRECTIVE` procedure, and specify the new resource allocation values for the PDB.

For example, the following procedure updates the resource allocation to a PDB named `operpdb` in the `newcdb_plan` CDB resource plan:

```
BEGIN
  DBMS_RESOURCE_MANAGER.UPDATE_CDB_PLAN_DIRECTIVE(
    plan                     => 'newcdb_plan',
    pluggable_database       => 'operpdb',
    new_shares               => 1,
    new_utilization_limit    => 10,
    new_parallel_server_limit => 20);
END;
/
```

4. Validate the pending area:

```
exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
```

5. Submit the pending area:

```
exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

See Also:

- *Oracle Multitenant Administrator's Guide*
- "About CDB Resource Plans"

26.4.1.6.2.3 Deleting CDB Resource Plan Directives for a PDB

You can delete the CDB resource plan directive for a PDB using the `DELETE_CDB_PLAN_DIRECTIVE` procedure.

You might delete the directive for a PDB if you unplug or drop the PDB. However, you can retain the directive, and if the PDB is plugged into the CDB in the future, the existing directive applies to the PDB.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To delete a CDB resource plan directive for a PDB:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

```
exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```

3. Run the `DELETE_CDB_PLAN_DIRECTIVE` procedure, and specify the CDB resource plan and the PDB.

For example, the following procedure deletes the directive for a PDB named `operpdb` in the `newcdb_plan` CDB resource plan:

```
BEGIN
  DBMS_RESOURCE_MANAGER.DELETE_CDB_PLAN_DIRECTIVE(
    plan               => 'newcdb_plan',
    pluggable_database => 'operpdb');
END;
/
```

4. Validate the pending area:

```
exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
```

5. Submit the pending area:

```
exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

See Also:

- *Oracle Multitenant Administrator's Guide*
- "About CDB Resource Plans"

26.4.1.6.3 Managing CDB Resource Plan Directives for a PDB Performance Profile

You can create, update, and delete CDB resource plan directives for a PDB performance profile.

- Creating New CDB Resource Plan Directives for a PDB Performance Profile
  You can create a CDB resource plan directive for the a new PDB performance profile using the `CREATE_CDB_PROFILE_DIRECTIVE` procedure. The directive specifies how resources are allocated to the all PDBs that use the new profile.
- Updating CDB Resource Plan Directives for a PDB Performance Profile
  Update the CDB resource plan directive for a PDB performance profile using the `UPDATE_CDB_PROFILE_DIRECTIVE` procedure. The directive specifies how resources are allocated to the PDBs that use the PDB performance profile.
- Deleting CDB Resource Plan Directives for a PDB Performance Profile
  You can delete the CDB resource plan directive for a PDB performance profile using the `DELETE_CDB_PROFILE_DIRECTIVE` procedure.

26.4.1.6.3.1 Creating New CDB Resource Plan Directives for a PDB Performance Profile

You can create a CDB resource plan directive for the a new PDB performance profile using the `CREATE_CDB_PROFILE_DIRECTIVE` procedure. The directive specifies how resources are allocated to the all PDBs that use the new profile.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To create a new CDB resource plan directive for a PDB performance profile:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

```
exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```

3. Run the `CREATE_CDB_PROFILE_DIRECTIVE` procedure, and specify the appropriate values for the new PDB performance profile.

   For example, the following procedure allocates resources to a PDB performance profile named `copper` in the `newcdb_plan` CDB resource plan:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CDB_PROFILE_DIRECTIVE(
    plan                  => 'newcdb_plan',
    profile               => 'copper',
    shares                => 1,
    utilization_limit     => 20,
    parallel_server_limit => 30);
END;
/
```

4. Validate the pending area:

```
exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
```

5. Submit the pending area:

```
exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

> **Note:**
>
> For a PDB to use the new profile, the PDB must have the `DB_PERFORMANCE_PROFILE` initialization parameter set to the profile name.

> **See Also:**
>
> - "About CDB Resource Plans"
> - *Oracle Multitenant Administrator's Guide* for information about accessing containers in a CDB

26.4.1.6.3.2 Updating CDB Resource Plan Directives for a PDB Performance Profile

Update the CDB resource plan directive for a PDB performance profile using the `UPDATE_CDB_PROFILE_DIRECTIVE` procedure. The directive specifies how resources are allocated to the PDBs that use the PDB performance profile.

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To update a CDB resource plan directive for a PDB performance profile:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Run the `UPDATE_CDB_PROFILE_DIRECTIVE` procedure, and specify the new resource allocation values for the PDB performance profile.

   For example, the following procedure updates the resource allocation for a PDB performance profile named `copper` in the `newcdb_plan` CDB resource plan:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.UPDATE_CDB_PROFILE_DIRECTIVE(
       plan                      => 'newcdb_plan',
       profile                   => 'copper',
       new_shares                => 1,
       new_utilization_limit     => 10,
       new_parallel_server_limit => 20);
   END;
   /
   ```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

> **See Also:**
>
> - "About CDB Resource Plans"
> - *Oracle Multitenant Administrator's Guide* for information about accessing containers in a CDB

26.4.1.6.3.3 Deleting CDB Resource Plan Directives for a PDB Performance Profile

You can delete the CDB resource plan directive for a PDB performance profile using the `DELETE_CDB_PROFILE_DIRECTIVE` procedure.

If no PDBs use a performance profile, then you might delete the directive for the profile.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To delete a CDB resource plan directive for a PDB performance profile:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Run the `DELETE_CDB_PROFILE_DIRECTIVE` procedure, and specify the CDB resource plan and the PDB performance profile.

   For example, the following procedure deletes the directive for a PDB named `operpdb` in the `newcdb_plan` CDB resource plan:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.DELETE_CDB_PLAN_DIRECTIVE(
       plan    => 'newcdb_plan',
       profile => 'operpdb');
   END;
   /
   ```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

> **See Also:**
>
> - *Oracle Multitenant Administrator's Guide* for information about accessing containers in a CDB
> - "About CDB Resource Plans"

26.4.1.6.4 Updating the Default Directive for PDBs in a CDB Resource Plan

You can update the default directive for PDBs in a CDB resource plan using the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure. The default directive applies to PDBs for which specific directives have not been defined.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To update the default directive for PDBs in a CDB resource plan:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Run the `UPDATE_CDB_DEFAULT_DIRECTIVE` procedure, and specify the appropriate default resource allocation values.

   For example, the following procedure updates the default directive for PDBs in the `newcdb_plan` CDB resource plan:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.UPDATE_CDB_DEFAULT_DIRECTIVE(
       plan                 => 'newcdb_plan',
       new_shares           => 2,
       new_utilization_limit => 40);
   END;
   /
   ```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

> **See Also:**
>
> - "The Default Directive for PDBs"
> - "About CDB Resource Plans"
> - *Oracle Multitenant Administrator's Guide* for information about accessing containers in a CDB

26.4.1.6.5 Updating the Default Directive for Maintenance Tasks in a CDB Resource Plan

You can update the AutoTask directive in a CDB resource plan using the `UPDATE_CDB_AUTOTASK_DIRECTIVE` procedure. The AutoTask directive applies to automatic maintenance tasks that are run in the root maintenance window.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To update the AutoTask directive for maintenance tasks in a CDB resource plan:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Run the `UPDATE_CDB_AUTOTASK_DIRECTIVE` procedure, and specify the appropriate AutoTask resource allocation values.

   For example, the following procedure updates the AutoTask directive for maintenance tasks in the `newcdb_plan` CDB resource plan:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.UPDATE_CDB_AUTOTASK_DIRECTIVE(
       plan                  => 'newcdb_plan',
       new_shares            => 2,
       new_utilization_limit => 60);
   END;
   /
   ```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

> **See Also:**
>
> - *Oracle Multitenant Administrator's Guide* for information about accessing containers
> - "About CDB Resource Plans"

26.4.1.6.6 Deleting a CDB Resource Plan

You can delete a CDB resource plan using the `DELETE_CDB_PLAN` procedure.

The resource plan must be disabled. You might delete a CDB resource plan if the plan is no longer needed. You can enable a different CDB resource plan, or you can disable Resource Manager for the CDB.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To delete a CDB resource plan:

1. In SQL*Plus, ensure that the current container is the root.
2. Create a pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
   ```

3. Run the `DELETE_CDB_PLAN` procedure, and specify the CDB resource plan.

   For example, the following procedure deletes the `newcdb_plan` CDB resource plan:

   ```
   BEGIN
     DBMS_RESOURCE_MANAGER.DELETE_CDB_PLAN(
       plan => 'newcdb_plan');
   END;
   /
   ```

4. Validate the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
   ```

5. Submit the pending area:

   ```
   exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
   ```

> **See Also:**
>
> - "About CDB Resource Plans"
> - *Oracle Multitenant Administrator's Guide* for information about accessing containers
> - "Enabling a CDB Resource Plan"
> - "Disabling a CDB Resource Plan"

26.4.1.7 Disabling a CDB Resource Plan

Disable the Resource Manager for a CDB by unsetting the `RESOURCE_MANAGER_PLAN` initialization parameter in the CDB root.

A CDB resource plan that specifies shares or utilization limits for PDBs is required to enable CPU management, both between PDBs and within a PDB. If a resource plan with shares or utilization limits is enabled for a PDB, and if the CDB resource plan is not specified, then the CDB resource plan is set to `DEFAULT_CDB_PLAN`. This setting gives equal shares to all PDBs and specifies no utilization limits. To disable CPU resource management throughout the CDB, set `RESOURCE_MANAGER_PLAN` to `ORA$INTERNAL_CDB_PLAN`.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To disable a CDB resource plan:

1. In SQL*Plus, ensure that the current container is the root.
2. Perform one of the following actions:
   - Use an `ALTER SYSTEM` statement to unset the `RESOURCE_MANAGER_PLAN` initialization parameter for the CDB.

     The following example unsets the `RESOURCE_MANAGER_PLAN` initialization parameter using an `ALTER SYSTEM` statement:

     ```
     ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = '';
     ```

   - In an initialization parameter file, unset the `RESOURCE_MANAGER_PLAN` initialization parameter, and restart the CDB.

     The following example unsets the `RESOURCE_MANAGER_PLAN` initialization parameter in an initialization parameter file:

     ```
     RESOURCE_MANAGER_PLAN =
     ```

> **See Also:**
>
> - *Oracle Multitenant Administrator's Guide* for information about accessing a container
> - *Oracle Multitenant Administrator's Guide*
> - *Oracle Multitenant Administrator's Guide* for information about starting up a database

26.4.1.8 Viewing Information About Plans and Directives in a CDB

You can view information about CDB resource plans, CDB resource plan directives, and predefined resource plans in a CDB.

- Viewing CDB Resource Plans
  An example illustrates using the `DBA_CDB_RSRC_PLANS` view to display all CDB resource plans defined in the CDB.
- Viewing CDB Resource Plan Directives
  An example illustrates using the `DBA_CDB_RSRC_PLAN_DIRECTIVES` view to display all directives defined in all CDB resource plans in the CDB.

> **See Also:**
>
> About Resource Manager Views for information about monitoring Oracle Database Resource Manager

26.4.1.8.1 Viewing CDB Resource Plans

An example illustrates using the `DBA_CDB_RSRC_PLANS` view to display all CDB resource plans defined in the CDB.

The `DEFAULT_CDB_PLAN` is supplied with Oracle Database. You can use this default plan if it meets your requirements.

To view CDB resource plans:

1. Start SQL*Plus or SQL Developer, and log in to the CDB root.
2. Run the following query:

```
COLUMN PLAN FORMAT A30
COLUMN STATUS FORMAT A10
COLUMN COMMENTS FORMAT A35

SELECT PLAN, STATUS, COMMENTS
FROM   DBA_CDB_RSRC_PLANS
ORDER BY PLAN;
```

Your output looks similar to the following:

```
PLAN                      STATUS      COMMENTS
------------------------- ----------- ----------------------------
DEFAULT_CDB_PLAN                      Default CDB plan
DEFAULT_MAINTENANCE_PLAN              Default CDB maintenance plan
NEWCDB_PLAN                           CDB plan for PDBs in newcdb
ORA$INTERNAL_CDB_PLAN                 Internal CDB plan
```

> **Note:**
>
> Plans in the pending area have a status of `PENDING`. Plans in the pending area are being edited. Any plan that is not in the pending area has a `NULL` status.

> **See Also:**
>
> "About CDB Resource Plans"

26.4.1.8.2 Viewing CDB Resource Plan Directives

An example illustrates using the `DBA_CDB_RSRC_PLAN_DIRECTIVES` view to display all directives defined in all CDB resource plans in the CDB.

The `DEFAULT_CDB_PLAN` is a default CDB plan that is supplied with Oracle Database. With `DEFAULT_CDB_PLAN`, every PDB has 1 share and a utilization limit of 100. If the CDB resource plan has no CPU directives configured, that is, the `shares` and `utilization_limits` directives are unset, then CPU Resource Manager uses the PDB-level `CPU_MIN_COUNT` and `CPU_COUNT` parameters to manage CPU. Note that `ORA$DEFAULT_PDB_DIRECTIVE` is the default directive for PDBs.

To view CDB resource plan directives:

1. Start SQL*Plus or SQL Developer, and log in to the CDB root.
2. Run the following query:

```
COLUMN PLAN HEADING 'Plan' FORMAT A24
COLUMN PLUGGABLE_DATABASE HEADING 'Pluggable Database' FORMAT A25
COLUMN SHARES HEADING 'Shares' FORMAT 999
COLUMN UTILIZATION_LIMIT HEADING 'Utilization|Limit' FORMAT 999
COLUMN PARALLEL_SERVER_LIMIT HEADING 'Parallel|Server|Limit' FORMAT 999

SELECT PLAN,
       PLUGGABLE_DATABASE,
       SHARES,
       UTILIZATION_LIMIT,
       PARALLEL_SERVER_LIMIT
  FROM DBA_CDB_RSRC_PLAN_DIRECTIVES
  ORDER BY PLAN;
```

Your output looks similar to the following:

```
                                                                 Parallel
                                                    Utilization   Server
Plan                    Pluggable Database        Shares       Limit    Limit
----------------------- ------------------------- ------ ----------- --------
DEFAULT_CDB_PLAN        ORA$DEFAULT_PDB_DIRECTIVE      1         100      100
DEFAULT_CDB_PLAN        ORA$AUTOTASK                             90      100
DEFAULT_MAINTENANCE_PLAN ORA$AUTOTASK                            90      100
DEFAULT_MAINTENANCE_PLAN ORA$DEFAULT_PDB_DIRECTIVE      1         100      100
NEWCDB_PLAN             HRPDB                           1          70       70
NEWCDB_PLAN             SALESPDB                        3         100      100
NEWCDB_PLAN             ORA$DEFAULT_PDB_DIRECTIVE       1          50       50
NEWCDB_PLAN             ORA$AUTOTASK                    1          75       75
NEWCDB_PLAN             SERVICESPDB                     3         100      100
```

The preceding output shows the directives for the `newcdb_plan` created in "Creating a CDB Resource Plan for Managing PDBs: Scenario" and modified in "Modifying a CDB Resource Plan".

See Also:

- "About CDB Resource Plans"
- "The Default Directive for PDBs"

26.4.2 Managing PDB Resource Plans

You can create, enable, and modify resource plans for individual PDBs.

- Creating a PDB Resource Plan
  You create a PDB resource plan by using procedures in the `DBMS_RESOURCE_MANAGER` PL/SQL package.
- Enabling a PDB Resource Plan
  Enable a PDB resource plan by setting the `RESOURCE_MANAGER_PLAN` initialization parameter to the plan with an `ALTER SYSTEM` statement when the current container is the PDB.
- Modifying a PDB Resource Plan
  You can use the `DBMS_RESOURCE_MANAGER` package to modify a PDB resource plan.
- Disabling a PDB Resource Plan
  You disable a PDB resource plan by unsetting the `RESOURCE_MANAGER_PLAN` initialization parameter in the PDB.

26.4.2.1 Creating a PDB Resource Plan

You create a PDB resource plan by using procedures in the `DBMS_RESOURCE_MANAGER` PL/SQL package.

A CDB resource plan allocates a portion of the system's resources to a PDB. A PDB resource plan determines how this portion is allocated within the PDB.

The following is a summary of the steps required to create a PDB resource plan:

1. In SQL*Plus, ensure that the current container is a PDB.
2. Create a pending area using the `CREATE_PENDING_AREA` procedure.
3. Create, modify, or delete consumer groups using the `CREATE_CONSUMER_GROUP` procedure.
4. Map sessions to consumer groups using the `SET_CONSUMER_GROUP_MAPPING` procedure.
5. Create the PDB resource plan using the `CREATE_PLAN` procedure.
6. Create PDB resource plan directives using the `CREATE_PLAN_DIRECTIVE` procedure.
7. Validate the pending area using the `VALIDATE_PENDING_AREA` procedure.
8. Submit the pending area using the `SUBMIT_PENDING_AREA` procedure.

Ensure that the current container is a PDB and that the user has the required privileges when you complete these steps. See "Creating a Complex Resource Plan" for detailed information about completing these steps.

You also have the option of creating a simple resource plan that is adequate for many situations using the `CREATE_SIMPLE_PLAN` procedure. See "Creating a Simple Resource Plan " for information about creating a simple resource plan.

> **Note:**
>
> Some restrictions apply to PDB resource plans. See "About PDB Resource Plans" for information.

26.4.2.2 Enabling a PDB Resource Plan

Enable a PDB resource plan by setting the `RESOURCE_MANAGER_PLAN` initialization parameter to the plan with an `ALTER SYSTEM` statement when the current container is the PDB.

If no plan is specified with this parameter, then no PDB resource plan is enabled for the PDB.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To enable a PDB resource plan:

1. In SQL*Plus, ensure that the current container is a PDB.
2. Use an `ALTER SYSTEM` statement to set the `RESOURCE_MANAGER_PLAN` initialization parameter to the PDB resource plan.

You can also schedule a PDB resource plan change with Oracle Scheduler.

Example 26-5 Enabling a PDB Resource Plan

The following example sets the PDB resource plan to `salespdb_plan`.

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = 'salespdb_plan';
```

> **See Also:**
>
> - *Oracle Multitenant Administrator's Guide* for information about accessing a container
> - " Oracle Scheduler Concepts" to learn how to schedule a PDB resource plan change with Oracle Scheduler

26.4.2.3 Modifying a PDB Resource Plan

You can use the `DBMS_RESOURCE_MANAGER` package to modify a PDB resource plan.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To modify a PDB resource plan:

1. In SQL*Plus, ensure that the current container is a PDB.
2. Create a pending area:

```
exec DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
```

3. Modify the PDB resource plan by completing one or more of the following tasks:
   - Update a consumer group using the `UPDATE_CONSUMER_GROUP` procedure.
   - Delete a consumer group using the `DELETE_CONSUMER_GROUP` procedure.
   - Update a resource plan using the `UPDATE_PLAN` procedure.
   - Delete a resource plan using the `DELETE_PLAN` procedure.
   - Update a resource plan directive using the `UPDATE_PLAN_DIRECTIVE` procedure.
   - Delete a resource plan directive using the `DELETE_PLAN_DIRECTIVE` procedure.
4. Validate the pending area:

```
exec DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
```

5. Submit the pending area:

```
exec DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

> **See Also:**
>
> - "About PDB Resource Plans"
> - *Oracle Multitenant Administrator's Guide* for information about accessing a container
> - "Maintaining Consumer Groups, Plans, and Directives" for instructions about completing the consumer group tasks

26.4.2.4 Disabling a PDB Resource Plan

You disable a PDB resource plan by unsetting the `RESOURCE_MANAGER_PLAN` initialization parameter in the PDB.

Prerequisites

The CDB must exist and must contain PDBs. To complete a task that uses the `DBMS_RESOURCE_MANAGER` package, you must have `ADMINISTER_RESOURCE_MANAGER` system privilege.

To disable a PDB resource plan:

1. In SQL*Plus, ensure that the current container is a PDB.
2. Use an `ALTER SYSTEM` statement to unset the `RESOURCE_MANAGER_PLAN` initialization parameter for the PDB.

Example 26-6 Disabling a PDB Resource Plan

The following example disables the PDB resource plan.

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = '';
```

> **See Also:**
>
> - "*Oracle Multitenant Administrator's Guide*" for information about modifying a PDB at the system level
> - *Oracle Multitenant Administrator's Guide* for information about accessing a container

26.4.3 Creating a Simple Resource Plan

You can quickly create a simple resource plan that is adequate for many situations using the `CREATE_SIMPLE_PLAN` procedure.

This procedure enables you to both create consumer groups and allocate resources to them by executing a single procedure call. Using this procedure, you are not required to invoke the procedures that are described in succeeding sections for creating a pending area, creating each consumer group individually, specifying resource plan directives, and so on.

You specify the following arguments for the CREATE_SIMPLE_PLAN procedure:

Parameter	Description
SIMPLE_PLAN	Name of the plan
CONSUMER_GROUP1	Consumer group name for first group
GROUP1_PERCENT	CPU resource allocated to this group
CONSUMER_GROUP2	Consumer group name for second group
GROUP2_PERCENT	CPU resource allocated to this group
CONSUMER_GROUP3	Consumer group name for third group
GROUP3_PERCENT	CPU resource allocated to this group
CONSUMER_GROUP4	Consumer group name for fourth group
GROUP4_PERCENT	CPU resource allocated to this group
CONSUMER_GROUP5	Consumer group name for fifth group
GROUP5_PERCENT	CPU resource allocated to this group
CONSUMER_GROUP6	Consumer group name for sixth group
GROUP6_PERCENT	CPU resource allocated to this group
CONSUMER_GROUP7	Consumer group name for seventh group
GROUP7_PERCENT	CPU resource allocated to this group
CONSUMER_GROUP8	Consumer group name for eighth group
GROUP8_PERCENT	CPU resource allocated to this group

You can specify up to eight consumer groups with this procedure. The only resource allocation method supported is CPU. The plan uses the EMPHASIS CPU allocation policy (the default) and each consumer group uses the ROUND_ROBIN scheduling policy (also the default).

Example: Creating a Simple Plan with the CREATE_SIMPLE_PLAN Procedure

The following PL/SQL block creates a simple resource plan with two user-specified consumer groups:

```
BEGIN
   DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN(SIMPLE_PLAN => 'SIMPLE_PLAN1',
   CONSUMER_GROUP1 => 'MYGROUP1', GROUP1_PERCENT => 80,
   CONSUMER_GROUP2 => 'MYGROUP2', GROUP2_PERCENT => 20);
END;
/
```

After executing the preceding statements, you can display the plan created using the following query:

```
SELECT plan, group_or_subplan, mgmt_p1
FROM   dba_rsrc_plan_directives
WHERE  plan = 'SIMPLE_PLAN1';
```

The plan created in a non-multitenant environment is:

```
PLAN                 GROUP_OR_SUBPLAN        MGMT_P1
-------------------- -------------------- ----------
SIMPLE_PLAN1         MYGROUP1                     80
SIMPLE_PLAN1         MYGROUP2                     20
SIMPLE_PLAN1         SYS_GROUP                    50
SIMPLE_PLAN1         OTHER_GROUPS                  5
```

The plan created in a multitenant environment is:

```
PLAN                 GROUP_OR_SUBPLAN        MGMT_P1
-------------------- -------------------- ----------
SIMPLE_PLAN1         MYGROUP1                     80
SIMPLE_PLAN1         MYGROUP2                     20
SIMPLE_PLAN1         OTHER_GROUPS                  5
```

See Also:

- "Creating a Resource Plan" for more information on the `EMPHASIS` CPU allocation policy
- "Creating Resource Consumer Groups " for more information on the `ROUND_ROBIN` scheduling policy
- "Elements of the Resource Manager"

26.4.4 Creating a Complex Resource Plan

When your situation calls for a more complex resource plan, you must create the plan, with its directives and consumer groups, in a staging area called the pending area, and then validate the plan before storing it in the data dictionary.

The following is a summary of the steps required to create a complex resource plan.

Note:

A complex resource plan is any resource plan that is not created with the `DBMS_RESOURCE_MANAGER.CREATE_SIMPLE_PLAN` procedure.

Step 1: Create a pending area.

Step 2: Create, modify, or delete consumer groups.

Step 3: Map sessions to consumer groups.

Step 4: Create the resource plan.

Step 5: Create resource plan directives.

Step 6: Validate the pending area.

Step 7: Submit the pending area.

You use procedures in the `DBMS_RESOURCE_MANAGER` PL/SQL package to complete these steps.

- About the Pending Area
  The **pending area** is a staging area where you can create a new resource plan, update an existing plan, or delete a plan without affecting currently running applications.
- Creating a Pending Area
  You create a pending area with the `CREATE_PENDING_AREA` procedure.
- Creating Resource Consumer Groups
  You create a resource consumer group using the `CREATE_CONSUMER_GROUP` procedure.
- Mapping Sessions to Consumer Groups
  You can map sessions to consumer groups using the `SET_CONSUMER_GROUP_MAPPING` procedure.
- Creating a Resource Plan
  You create a resource plan with the `CREATE_PLAN` procedure.
- Creating Resource Plan Directives
  You use the `CREATE_PLAN_DIRECTIVE` procedure to create resource plan directives. Each directive belongs to a plan or subplan and allocates resources to either a consumer group or subplan.
- Validating the Pending Area
  At any time when you are making changes in the pending area, you can call `VALIDATE_PENDING_AREA` to ensure that the pending area is valid so far.
- Submitting the Pending Area
  After you have validated your changes, call the `SUBMIT_PENDING_AREA` procedure to make your changes active.
- Clearing the Pending Area
  You can clear the pending area at any time using the `CLEAR_PENDING_AREA` procedure.

See Also:

- Predefined Consumer Group Mapping Rules
- *Oracle Database PL/SQL Packages and Types Reference* for details on the `DBMS_RESOURCE_MANAGER` PL/SQL package.
- "Elements of the Resource Manager"

26.4.4.1 About the Pending Area

The **pending area** is a staging area where you can create a new resource plan, update an existing plan, or delete a plan without affecting currently running applications.

When you create a pending area, the database initializes it and then copies existing plans into the pending area so that they can be updated.

> **Tip:**
>
> After you create the pending area, if you list all plans by querying the `DBA_RSRC_PLANS` data dictionary view, you see two copies of each plan: one with the `PENDING` status, and one without. The plans with the `PENDING` status reflect any changes you made to the plans since creating the pending area. Pending changes can also be viewed for consumer groups using `DBA_RSRC_CONSUMER_GROUPS` and for resource plan directives using `DBA_RSRC_PLAN_DIRECTIVES`. See Resource Manager Data Dictionary Views for more information.

After you make changes in the pending area, you validate the pending area and then submit it. Upon submission, all pending changes are applied to the data dictionary, and the pending area is cleared and deactivated.

If you attempt to create, update, or delete a plan (or create, update, or delete consumer groups or resource plan directives) without first creating the pending area, you receive an error message.

Submitting the pending area does not activate any new plan that you create; it just stores new or updated plan information in the data dictionary. However, if you modify a plan that is currently active, the plan is reactivated with the new plan definition. See "Enabling Oracle Database Resource Manager and Switching Plans" for information about activating a resource plan.

When you create a pending area, no other users can create one until you submit or clear the pending area or log out.

26.4.4.2 Creating a Pending Area

You create a pending area with the `CREATE_PENDING_AREA` procedure.

Example: Creating a pending area:

The following PL/SQL block creates and initializes a pending area:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
END;
/
```

26.4.4.3 Creating Resource Consumer Groups

You create a resource consumer group using the `CREATE_CONSUMER_GROUP` procedure.

You can specify the following parameters:

Parameter	Description
`CONSUMER_GROUP`	Name to assign to the consumer group.
`COMMENT`	Any comment.
`CPU_MTH`	Deprecated. Use `MGMT_MTH`.

Parameter	Description
MGMT_MTH	The resource allocation method for distributing CPU among sessions in the consumer group. The default is 'ROUND-ROBIN', which uses a round-robin scheduler to ensure that sessions are fairly executed. 'RUN-TO-COMPLETION' specifies that long-running sessions are scheduled ahead of other sessions. This setting helps long-running sessions (such as batch processes) complete sooner.

Example: Creating a Resource Consumer Group

The following PL/SQL block creates a consumer group called OLTP with the default (ROUND-ROBIN) method of allocating resources to sessions in the group:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
   CONSUMER_GROUP => 'OLTP',
   COMMENT        => 'OLTP applications');
END;
/
```

See Also:

- "Updating a Consumer Group"
- "Deleting a Consumer Group"

26.4.4.4 Mapping Sessions to Consumer Groups

You can map sessions to consumer groups using the SET_CONSUMER_GROUP_MAPPING procedure.

You can specify the following parameters:

Parameter	Description
ATTRIBUTE	Session attribute type, specified as a package constant.
VALUE	Value of the attribute.
CONSUMER_GROUP	Name of the consumer group.

Example: Mapping a Session to a Consumer Group

The following PL/SQL block maps the oe user to the OLTP consumer group:

```
BEGIN
  DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING(
    ATTRIBUTE      => DBMS_RESOURCE_MANAGER.ORACLE_USER,
    VALUE          => 'OE',
    CONSUMER_GROUP => 'OLTP');
END;
/
```

> **See Also:**
>
> "Creating Consumer Group Mapping Rules"

26.4.4.5 Creating a Resource Plan

You create a resource plan with the `CREATE_PLAN` procedure.

You can specify the parameters shown in the following table. The first two parameters are required. The remainder are optional.

Parameter	Description
`PLAN`	Name to assign to the plan.
`COMMENT`	Any descriptive comment.
`CPU_MTH`	Deprecated. Use `MGMT_MTH`.
`ACTIVE_SESS_POOL_MTH`	Active session pool resource allocation method. `ACTIVE_SESS_POOL_ABSOLUTE` is the default and only method available.
`PARALLEL_DEGREE_LIMIT_MTH`	Resource allocation method for specifying a limit on the `PARALLEL_DEGREE_LIMIT_ABSOLUTE` is the default and only method available.
`QUEUEING_MTH`	Queuing resource allocation method. Controls the order in which queued inactive sessions are removed from the queue and added to the active session pool. `FIFO_TIMEOUT` is the default and only method available.
`MGMT_MTH`	Resource allocation method for specifying how much CPU each consumer group or subplan gets. `'EMPHASIS'`, the default method, is for single-level or multilevel plans that use percentages to specify how CPU is distributed among consumer groups. `'RATIO'` is for single-level plans that use ratios to specify how CPU is distributed.
`SUB_PLAN`	If `TRUE`, the plan cannot be used as the top plan; it can be used as a subplan only. Default is `FALSE`.

Example: Creating a Resource Plan

The following PL/SQL block creates a resource plan named `DAYTIME`:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN(
   PLAN    => 'DAYTIME',
   COMMENT => 'More resources for OLTP applications');
END;
/
```

- About the RATIO CPU Allocation Method
  The `RATIO` method is an alternate CPU allocation method intended for simple plans that have only a single level of CPU allocation.

26.4.4.5.1 About the RATIO CPU Allocation Method

The `RATIO` method is an alternate CPU allocation method intended for simple plans that have only a single level of CPU allocation.

Instead of percentages, you specify numbers corresponding to the ratio of CPU that you want to give to each consumer group. To use the `RATIO` method, you set the `MGMT_MTH` argument for the `CREATE_PLAN` procedure to '`RATIO`'. See "Creating Resource Plan Directives " for an example of a plan that uses this method.

> **See Also:**
>
> - "Updating a Plan"
> - "Deleting a Plan"

26.4.4.6 Creating Resource Plan Directives

You use the `CREATE_PLAN_DIRECTIVE` procedure to create resource plan directives. Each directive belongs to a plan or subplan and allocates resources to either a consumer group or subplan.

> **Note:**
>
> The set of directives for a resource plan and its subplans can name a particular subplan only once.
>
> You can specify directives for a particular consumer group in a top plan and its subplans. However, Oracle recommends that the set of directives for a resource plan and its subplans name a particular consumer group only once.

You can specify the following parameters:

Parameter	Description
`PLAN`	Name of the resource plan to which the directive belongs.
`GROUP_OR_SUBPLAN`	Name of the consumer group or subplan to which to allocate resources.
`COMMENT`	Any comment.
`CPU_P1`	Deprecated. Use `MGMT_P1`.
`CPU_P2`	Deprecated. Use `MGMT_P2`.
`CPU_P3`	Deprecated. Use `MGMT_P3`.
`CPU_P4`	Deprecated. Use `MGMT_P4`.
`CPU_P5`	Deprecated. Use `MGMT_P5`.
`CPU_P6`	Deprecated. Use `MGMT_P6`.
`CPU_P7`	Deprecated. Use `MGMT_P7`.
`CPU_P8`	Deprecated. Use `MGMT_P8`.
`ACTIVE_SESS_POOL_P1`	Specifies the maximum number of concurrently active sessions for a consumer group. Other sessions await execution in an inactive session queue. Default is `UNLIMITED`.

Parameter	Description
QUEUEING_P1	Specifies time (in seconds) after which a session in an inactive session queue (waiting for execution) times out and the call is terminated. Default is UNLIMITED.
PARALLEL_DEGREE_LIMIT_P1	Specifies a limit on the degree of parallelism for any operation. Default is UNLIMITED.
SWITCH_GROUP	Specifies the consumer group to which a session is switched if switch criteria are met. If the group name is CANCEL_SQL, then the current call is canceled when switch criteria are met. If the group name is CANCEL_SQL, then the SWITCH_FOR_CALL parameter is always set to TRUE, overriding the user-specified setting. If the group name is KILL_SESSION, then the session is terminated when switch criteria are met. If the group name is LOG_ONLY, then information about the session is recorded in real-time SQL monitoring, but no specific action is taken for the session. If NULL, then the session is not switched and no additional logging is performed. The default is NULL. An error is returned if this parameter is set to NULL and any other switch parameter is set to non-NULL. **Note:** The following consumer group names are reserved: CANCEL_SQL, KILL_SESSION, and LOG_ONLY. An error results if you attempt to create a consumer group with one of these names.
SWITCH_TIME	Specifies the time (in CPU seconds) that a call can execute before an action is taken. Default is UNLIMITED. The action is specified by SWITCH_GROUP.
SWITCH_ESTIMATE	If TRUE, the database estimates the execution time of each call, and if estimated execution time exceeds SWITCH_TIME, the session is switched to the SWITCH_GROUP before beginning the call. Default is FALSE. The execution time estimate is obtained from the optimizer. The accuracy of the estimate is dependent on many factors, especially the quality of the optimizer statistics. In general, you should expect statistics to be no more accurate than ± 10 minutes.
MAX_EST_EXEC_TIME	Specifies the maximum execution time (in CPU seconds) allowed for a call. If the optimizer estimates that a call will take longer than MAX_EST_EXEC_TIME, the call is not allowed to proceed and ORA-07455 is issued. If the optimizer does not provide an estimate, this directive has no effect. Default is UNLIMITED. The accuracy of the estimate is dependent on many factors, especially the quality of the optimizer statistics.
UNDO_POOL	Sets a maximum in kilobytes (K) on the total amount of undo for uncommitted transactions that can be generated by a consumer group. Default is UNLIMITED.
MAX_IDLE_TIME	Indicates the maximum session idle time, in seconds. Default is NULL, which implies unlimited.
MAX_IDLE_BLOCKER_TIME	Indicates the maximum session idle time of a blocking session, in seconds. Default is NULL, which implies unlimited.
SWITCH_TIME_IN_CALL	Deprecated. Use SWITCH_FOR_CALL.

Parameter	Description
MGMT_P1	For a plan with the MGMT_MTH parameter set to EMPHASIS, specifies the CPU percentage to allocate at the first level. For MGMT_MTH set to RATIO, specifies the weight of CPU usage. Default is NULL for all MGMT_P*n* parameters.
MGMT_P2	For EMPHASIS, specifies CPU percentage to allocate at the second level. Not applicable for RATIO.
MGMT_P3	For EMPHASIS, specifies CPU percentage to allocate at the third level. Not applicable for RATIO.
MGMT_P4	For EMPHASIS, specifies CPU percentage to allocate at the fourth level. Not applicable for RATIO.
MGMT_P5	For EMPHASIS, specifies CPU percentage to allocate at the fifth level. Not applicable for RATIO.
MGMT_P6	For EMPHASIS, specifies CPU percentage to allocate at the sixth level. Not applicable for RATIO.
MGMT_P7	For EMPHASIS, specifies CPU percentage to allocate at the seventh level. Not applicable for RATIO.
MGMT_P8	For EMPHASIS, specifies CPU percentage to allocate at the eighth level. Not applicable for RATIO.
SWITCH_IO_MEGABYTES	Specifies the number of megabytes of physical I/O that a session can transfer (read and write) before an action is taken. Default is UNLIMITED. The action is specified by SWITCH_GROUP.
SWITCH_IO_REQS	Specifies the number of physical I/O requests that a session can execute before an action is taken. Default is UNLIMITED. The action is specified by SWITCH_GROUP.
SWITCH_FOR_CALL	If TRUE, a session that was automatically switched to another consumer group (according to SWITCH_TIME, SWITCH_IO_MEGABYTES, or SWITCH_IO_REQS) is returned to its original consumer group when the top level call completes. Default is NULL.
PARALLEL_QUEUE_TIMEOUT	Specifies the maximum time, in seconds, that a parallel statement can wait in the parallel statement queue before it is timed out.
PARALLEL_SERVER_LIMIT	Specifies the maximum percentage of the parallel execution server pool that a particular consumer group can use. The number of parallel execution servers used by a particular consumer group is counted as the sum of the parallel execution servers used by all sessions in that consumer group.
UTILIZATION_LIMIT	Specifies the maximum CPU utilization percentage permitted for the consumer group. This value overrides any level allocations for CPU (MGMT_P1 through MGMT_P8), and also imposes a limit on total CPU utilization when unused allocations are redistributed. You can specify this attribute and leave MGMT_P1 through MGMT_P8 NULL.
SWITCH_IO_LOGICAL	Number of logical I/O requests that will trigger the action specified by SWITCH_GROUP. As with other switch directives, if SWITCH_FOR_CALL is TRUE, then the number of logical I/O requests is accumulated from the start of a call. Otherwise, the number of logical I/O requests is accumulated for the length of the session.

Parameter	Description
SWITCH_ELAPSED_TIME	Elapsed time, in seconds, that will trigger the action specified by SWITCH_GROUP. As with other switch directives, if SWITCH_FOR_CALL is TRUE, then the elapsed time is accumulated from the start of a call. Otherwise, the elapsed time is accumulated for the length of the session.
SHARES	Allocates resources among pluggable databases (PDBs) in a multitenant container database (CDB). Also allocates resources among consumer groups in a PDB.
PARALLEL_STMT_CRITICAL	Specifies whether parallel statements from the consumer group are critical. When BYPASS_QUEUE is specified, parallel statements from the consumer group are critical. These statements bypass the parallel queue and are executed immediately. When FALSE or NULL (the default) is specified, parallel statements from the consumer group are not critical. These statements are added to the parallel queue when necessary.
SESSION_PGA_LIMIT	Specifies the maximum amount of PGA memory, in megabytes, that can be allocated to each session in a particular consumer group. If a session exceeds the limit, then its process is terminated with an ORA-10260 error.

Example 1

The following PL/SQL block creates a resource plan directive for plan DAYTIME. (It assumes that the DAYTIME plan and OLTP consumer group are already created in the pending area.)

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN             => 'DAYTIME',
   GROUP_OR_SUBPLAN => 'OLTP',
   COMMENT          => 'OLTP group',
   MGMT_P1          => 75);
END;
/
```

This directive assigns 75% of CPU resources to the OLTP consumer group at level 1.

You can also create the REPORTING consumer group, and then execute the following PL/SQL block:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN                     => 'DAYTIME',
   GROUP_OR_SUBPLAN         => 'REPORTING',
   COMMENT                  => 'Reporting group',
   MGMT_P1                  => 15,
   PARALLEL_DEGREE_LIMIT_P1 => 8,
   ACTIVE_SESS_POOL_P1      => 4,
   SESSION_PGA_LIMIT        => 20);

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
   PLAN                     => 'DAYTIME',
   GROUP_OR_SUBPLAN         => 'OTHER_GROUPS',
   COMMENT                  => 'This one is required',
   MGMT_P1                  => 10);
```

```
END;
/
```

In this plan, consumer group REPORTING has a maximum degree of parallelism of 8 for any operation, while none of the other consumer groups are limited in their degree of parallelism. In addition, the REPORTING group has a maximum of 4 concurrently active sessions. Each session can use a maximum of 20 MB of PGA memory.

Example 2

This example uses the RATIO method to allocate CPU, which uses ratios instead of percentages. Suppose your application suite offers three service levels to clients: Gold, Silver, and Bronze. You create three consumer groups named GOLD_CG, SILVER_CG, and BRONZE_CG, and you create the following resource plan:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PLAN
   (PLAN             => 'SERVICE_LEVEL_PLAN',
    MGMT_MTH         => 'RATIO',
    COMMENT          => 'Plan that supports three service levels');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
   (PLAN             => 'SERVICE_LEVEL_PLAN',
    GROUP_OR_SUBPLAN => 'GOLD_CG',
    COMMENT          => 'Gold service level customers',
    MGMT_P1          => 10);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
   (PLAN             => 'SERVICE_LEVEL_PLAN',
    GROUP_OR_SUBPLAN => 'SILVER_CG',
    COMMENT          => 'Silver service level customers',
    MGMT_P1          => 5);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
   (PLAN             => 'SERVICE_LEVEL_PLAN',
    GROUP_OR_SUBPLAN => 'BRONZE_CG',
    COMMENT          => 'Bronze service level customers',
    MGMT_P1          => 2);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE
    (PLAN            => 'SERVICE_LEVEL_PLAN',
    GROUP_OR_SUBPLAN => 'OTHER_GROUPS',
    COMMENT          => 'Lowest priority sessions',
    MGMT_P1          => 1);
END;
/
```

The ratio of CPU allocation is 10:5:2:1 for the GOLD_CG, SILVER_CG, BRONZE_CG, and OTHER_GROUPS consumer groups, respectively.

If sessions exist only in the GOLD_CG and SILVER_CG consumer groups, then the ratio of CPU allocation is 10:5 between the two groups.

- Conflicting Resource Plan Directives
  Although this is allowed, Oracle strongly recommends that you avoid referencing the same consumer group from a top plan and any of its subplans.

26.4.4.6.1 Conflicting Resource Plan Directives

Although this is allowed, Oracle strongly recommends that you avoid referencing the same consumer group from a top plan and any of its subplans.

You may have occasion to reference the same consumer group from the top plan and any number of subplans. This results in multiple resource plan directives referring to the same consumer group.

Similarly, when multiple resource plan directives refer to the same consumer group, they have conflicting directives. Although this is allowed, Oracle strongly recommends that you avoid multiple resource plan directives that refer to the same consumer group.

See Also:

- "Updating a Resource Plan Directive"
- "Deleting a Resource Plan Directive"

26.4.4.7 Validating the Pending Area

At any time when you are making changes in the pending area, you can call `VALIDATE_PENDING_AREA` to ensure that the pending area is valid so far.

The following rules must be adhered to, and are checked by the validate procedure:

- No plan can contain any loops. A loop occurs when a subplan contains a directive that references a plan that is above the subplan in the plan hierarchy. For example, a subplan cannot reference the top plan.
- All plans and resource consumer groups referred to by plan directives must exist.
- All plans must have plan directives that point to either plans or resource consumer groups.
- All percentages in any given level must not add up to greater than 100.
- A plan that is currently being used as a top plan by an active instance cannot be deleted.
- The following parameters can appear only in plan directives that refer to resource consumer groups, not other resource plans:
  - `ACTIVE_SESS_POOL_P1`
  - `MAX_EST_EXEC_TIME`
  - `MAX_IDLE_BLOCKER_TIME`
  - `MAX_IDLE_TIME`
  - `PARALLEL_DEGREE_LIMIT_P1`
  - `QUEUEING_P1`
  - `SESSION_PGA_LIMIT`
  - `SWITCH_ESTIMATE`
  - `SWITCH_FOR_CALL`
  - `SWITCH_GROUP`
  - `SWITCH_IO_MEGABYTES`
  - `SWITCH_IO_REQS`
  - `SWITCH_TIME`

  - UNDO_POOL
  - UTILIZATION_LIMIT

- There can be no more than 28 resource consumer groups in any active plan. Also, at most, a plan can have 28 children.
- Plans and resource consumer groups cannot have the same name.
- There must be a plan directive for OTHER_GROUPS somewhere in any active plan. This ensures that a session that is not part of any of the consumer groups included in the currently active plan is allocated resources (as specified by the directive for OTHER_GROUPS).

VALIDATE_PENDING_AREA raises an error if any of the preceding rules are violated. You can then make changes to fix any problems and call the procedure again.

It is possible to create "orphan" consumer groups that have no plan directives referring to them. This allows the creation of consumer groups that will not currently be used, but might be part of some plan to be implemented in the future.

Example: Validating the Pending Area:

The following PL/SQL block validates the pending area.

```
BEGIN
  DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
END;
/
```

> **See Also:**
>
> "About the Pending Area"

26.4.4.8 Submitting the Pending Area

After you have validated your changes, call the SUBMIT_PENDING_AREA procedure to make your changes active.

The submit procedure also performs validation, so you do not necessarily need to make separate calls to the validate procedure. However, if you are making major changes to plans, debugging problems is often easier if you incrementally validate your changes. No changes are submitted (made active) until validation is successful on all of the changes in the pending area.

The SUBMIT_PENDING_AREA procedure clears (deactivates) the pending area after successfully validating and committing the changes.

> **Note:**
>
> A call to SUBMIT_PENDING_AREA might fail even if VALIDATE_PENDING_AREA succeeds. This can happen if, for example, a plan being deleted is loaded by an instance after a call to VALIDATE_PENDING_AREA, but before a call to SUBMIT_PENDING_AREA.

Example: Submitting the Pending Area:

The following PL/SQL block submits the pending area:

```
BEGIN
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

> **See Also:**
>
> "About the Pending Area"

26.4.4.9 Clearing the Pending Area

You can clear the pending area at any time using the `CLEAR_PENDING_AREA` procedure.

This PL/SQL block causes all of your changes to be cleared from the pending area and deactivates the pending area:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CLEAR_PENDING_AREA();
END;
/
```

After calling `CLEAR_PENDING_AREA`, you must call the `CREATE_PENDING_AREA` procedure before you can again attempt to make changes.

> **See Also:**
>
> "About the Pending Area"

26.5 Putting It All Together: Oracle Database Resource Manager Examples

Examples illustrate how to allocate resources with Resource Manager.

- Multilevel Plan Example
  An example illustrates a multilevel plan.
- Examples of Using the Utilization Limit Attribute
  You can use the `UTILIZATION_LIMIT` directive attribute to limit the CPU utilization for applications. One of the most common scenarios in which this attribute can be used is for database consolidation.
- Example of Using Several Resource Allocation Methods
  An example illustrates using several resource allocation methods.
- Example of Managing Parallel Statements Using Directive Attributes
  An example illustrates managing parallel statements using directive attributes.

- An Oracle-Supplied Mixed Workload Plan
  Oracle Database includes a predefined resource plan, `MIXED_WORKLOAD_PLAN`, that prioritizes interactive operations over batch operations, and includes the required subplans and consumer groups recommended by Oracle.

26.5.1 Multilevel Plan Example

An example illustrates a multilevel plan.

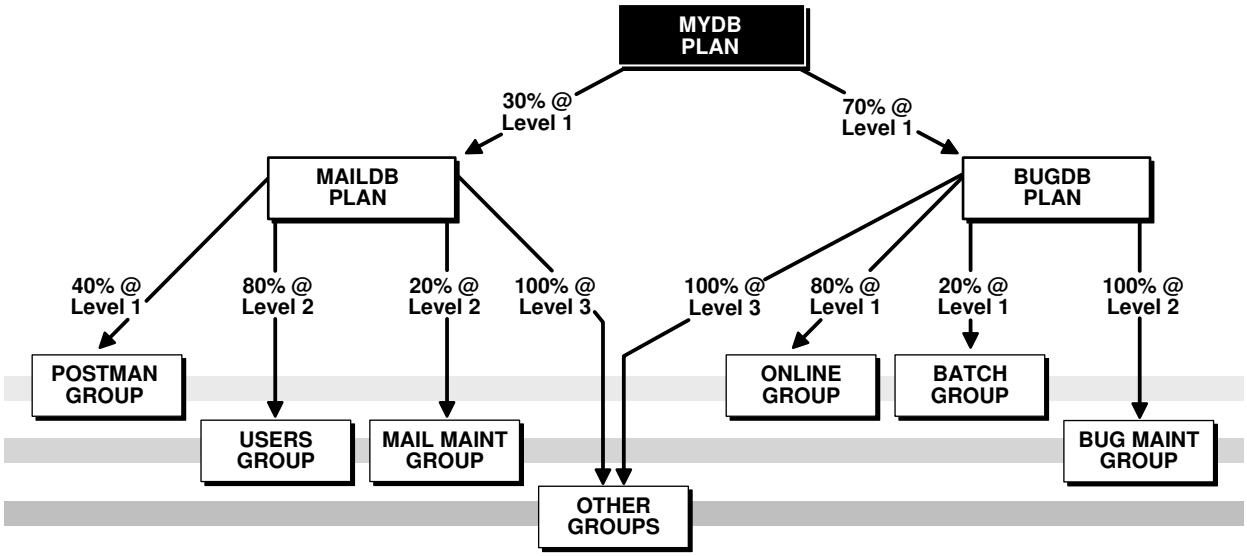
The following PL/SQL block creates a multilevel plan as illustrated in Figure 26-7. Default resource allocation method settings are used for all plans and resource consumer groups.

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'bugdb_plan',
   COMMENT => 'Resource plan/method for bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'maildb_plan',
   COMMENT => 'Resource plan/method for mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'mydb_plan',
   COMMENT => 'Resource plan/method for bug and mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Online_group',
   COMMENT => 'Resource consumer group/method for online bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Batch_group',
   COMMENT => 'Resource consumer group/method for batch job bug users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Bug_Maint_group',
   COMMENT => 'Resource consumer group/method for users sessions for bug db maint');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Users_group',
   COMMENT => 'Resource consumer group/method for mail users sessions');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Postal_Service_group',
   COMMENT => 'Resource consumer group/method for mail service');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'Mail_Maint_group',
   COMMENT => 'Resource consumer group/method for users sessions for mail db maint');
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
   GROUP_OR_SUBPLAN => 'Online_group',
   COMMENT => 'online bug users sessions at level 1', MGMT_P1 => 80, MGMT_P2=> 0);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
   GROUP_OR_SUBPLAN => 'Batch_group',
   COMMENT => 'batch bug users sessions at level 1', MGMT_P1 => 20, MGMT_P2 => 0,
   PARALLEL_DEGREE_LIMIT_P1 => 8);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
   GROUP_OR_SUBPLAN => 'Bug_Maint_group',
   COMMENT => 'bug maintenance users sessions at level 2', MGMT_P1 => 0, MGMT_P2 => 100);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'bugdb_plan',
   GROUP_OR_SUBPLAN => 'OTHER_GROUPS',
   COMMENT => 'all other users sessions at level 3', MGMT_P1 => 0, MGMT_P2 => 0,
   MGMT_P3 => 100);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
   GROUP_OR_SUBPLAN => 'Postal_Service_group',
   COMMENT => 'mail service at level 1', MGMT_P1 => 40, MGMT_P2 => 0);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
   GROUP_OR_SUBPLAN => 'Users_group',
   COMMENT => 'mail users sessions at level 2', MGMT_P1 => 0, MGMT_P2 => 80);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
   GROUP_OR_SUBPLAN => 'Mail_Maint_group',
   COMMENT => 'mail maintenance users sessions at level 2', MGMT_P1 => 0, MGMT_P2 => 20);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'maildb_plan',
   GROUP_OR_SUBPLAN => 'OTHER_GROUPS',
   COMMENT => 'all other users sessions at level 3', MGMT_P1 => 0, MGMT_P2 => 0,
   MGMT_P3 => 100);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'mydb_plan',
   GROUP_OR_SUBPLAN => 'maildb_plan',
   COMMENT=> 'all mail users sessions at level 1', MGMT_P1 => 30);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'mydb_plan',
   GROUP_OR_SUBPLAN => 'bugdb_plan',
   COMMENT => 'all bug users sessions at level 1', MGMT_P1 => 70);
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
```

```
END;
/
```

The preceding call to `VALIDATE_PENDING_AREA` is optional because the validation is implicitly performed in `SUBMIT_PENDING_AREA`.

Figure 26-7 Multilevel Plan Schema



In this plan schema, CPU resources are allocated as follows:

- Under `mydb_plan`, 30% of CPU is allocated to the `maildb_plan` subplan, and 70% is allocated to the `bugdb_plan` subplan. Both subplans are at level 1. Because `mydb_plan` itself has no levels below level 1, any resource allocations that are unused by either subplan at level 1 can be used by its sibling subplan. Thus, if `maildb_plan` uses only 20% of CPU, then 80% of CPU is available to `bugdb_plan`.
- `maildb_plan` and `bugdb_plan` define allocations at levels 1, 2, and 3. The levels in these subplans are independent of levels in their parent plan, `mydb_plan`. That is, all plans and subplans in a plan schema have their own level 1, level 2, level 3, and so on.
- Of the 30% of CPU allocated to `maildb_plan`, 40% of that amount (effectively 12% of total CPU) is allocated to `Postal_Service_group` at level 1. Because `Postal_Service_group` has no siblings at level 1, there is an implied 60% remaining at level 1. This 60% is then shared by `Users_group` and `Mail_Maint_group` at level 2, at 80% and 20%, respectively. In addition to this 60%, `Users_group` and `Mail_Maint_group` can also use any of the 40% not used by `Postal_Service_group` at level 1.
- CPU resources not used by either `Users_group` or `Mail_Maint_group` at level 2 are allocated to `OTHER_GROUPS`, because in multilevel plans, unused resources are reallocated to consumer groups or subplans at the next lower level, not to siblings at the same level. Thus, if `Users_group` uses only 70% instead of 80%, the remaining 10% cannot be used by `Mail_Maint_group`. That 10% is available only to `OTHER_GROUPS` at level 3.
- The 70% of CPU allocated to the `bugdb_plan` subplan is allocated to its consumer groups in a similar fashion. If either `Online_group` or `Batch_group` does not use its full allocation, the remainder may be used by `Bug_Maint_group`. If `Bug_Maint_group` does not use all of that allocation, the remainder goes to `OTHER_GROUPS`.

26.5.2 Examples of Using the Utilization Limit Attribute

You can use the `UTILIZATION_LIMIT` directive attribute to limit the CPU utilization for applications. One of the most common scenarios in which this attribute can be used is for database consolidation.

During database consolidation, you may need to be able to do the following:

- Manage the performance impact that one application can have on another.

  One method of managing this performance impact is to create a consumer group for each application and allocate resources to each consumer group.

- Limit the utilization of each application.

  Typically, in addition to allocating a specific percentage of the CPU resources to each consumer group, you may need to limit the maximum CPU utilization for each group. This limit prevents a consumer group from using all of the CPU resources when all the other consumer groups are idle.

  In some cases, you may want all application users to experience consistent performance regardless of the workload from other applications. This can be achieved by specifying a utilization limit for each consumer group in a resource plan.

The following examples demonstrate how to use the `UTILIZATION_LIMIT` resource plan directive attribute to:

- Restrict total database CPU utilization
- Quarantine runaway queries
- Limit CPU usage for applications
- Limit CPU utilization during maintenance windows

Example 1 - Restricting Overall Database CPU Utilization

In this example, regardless of database load, system workload from Oracle Database never exceeds 90% of CPU, leaving 10% of CPU for other applications sharing the server.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();

  DBMS_RESOURCE_MANAGER.CREATE_PLAN(
    PLAN    => 'MAXCAP_PLAN',
    COMMENT => 'Limit overall database CPU');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
    PLAN              => 'MAXCAP_PLAN',
    GROUP_OR_SUBPLAN  => 'OTHER_GROUPS',
    COMMENT           => 'This group is mandatory',
    UTILIZATION_LIMIT => 90);

  DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

Because there is no plan directive other than the one for `OTHER_GROUPS`, all sessions are mapped to `OTHER_GROUPS`.

Example 2 - Quarantining Runaway Queries

In this example, runaway queries are switched to a consumer group with a utilization limit of 20%, limiting the amount of resources that they can consume until you can intervene. A runaway query is characterized here as one that takes more than 10 minutes of CPU time. Assume that session mapping rules start all sessions in `START_GROUP`.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();

  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'START_GROUP',
     COMMENT        => 'Sessions start here');

  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'QUARANTINE_GROUP',
     COMMENT        => 'Sessions switched here to quarantine them');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN(
    PLAN    => 'Quarantine_plan',
    COMMENT => 'Quarantine runaway queries');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
    PLAN                  => 'Quarantine_plan',
    GROUP_OR_SUBPLAN      => 'START_GROUP',
    COMMENT               => 'Max CPU 10 minutes before switch',
    MGMT_P1               => 75,
    switch_group          => 'QUARANTINE_GROUP',
    switch_time           => 600);

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
    PLAN                  => 'Quarantine_plan',
    GROUP_OR_SUBPLAN      => 'OTHER_GROUPS',
    COMMENT               => 'Mandatory',
    MGMT_P1               => 25);

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(
    PLAN                  => 'Quarantine_plan',
    GROUP_OR_SUBPLAN      => 'QUARANTINE_GROUP',
    COMMENT               => 'Limited CPU',
    MGMT_P2               => 100,
    UTILIZATION_LIMIT     => 20);

  DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

> **Note:**
>
> Although you could set the utilization limit to zero for `QUARANTINE_GROUP`, thus completely quarantining runaway queries, it is recommended that you avoid doing this. If the runaway query is holding any resources—PGA memory, locks, and so on—required by any other session, then a zero allocation setting could lead to a deadlock.

Example 3 - Limiting CPU for Applications

In this example, assume that mapping rules map application sessions into one of four application groups. Each application group is allocated a utilization limit of 30%. This limits CPU utilization of any one application to 30%. The sum of the `UTILIZATION_LIMIT` values exceeds 100%, which is permissible and acceptable in a situation where all applications are not active simultaneously.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();

  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(
     CONSUMER_GROUP => 'APP1_GROUP',
     COMMENT        => 'Apps group 1');
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP2_GROUP',
     COMMENT        => 'Apps group 2');
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP3_GROUP',
     COMMENT        => 'Apps group 3');
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP4_GROUP',
     COMMENT        => 'Apps group 4');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN(
    PLAN    => 'apps_plan',
    COMMENT => 'Application consolidation');

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                 => 'apps_plan',
    GROUP_OR_SUBPLAN     => 'APP1_GROUP',
    COMMENT              => 'Apps group 1',
    UTILIZATION_LIMIT    => 30);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                 => 'apps_plan',
    GROUP_OR_SUBPLAN     => 'APP2_GROUP',
    COMMENT              => 'Apps group 2',
    UTILIZATION_LIMIT    => 30);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                 => 'apps_plan',
    GROUP_OR_SUBPLAN     => 'APP3_GROUP',
    COMMENT              => 'Apps group 3',
    UTILIZATION_LIMIT    => 30);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                 => 'apps_plan',
    GROUP_OR_SUBPLAN     => 'APP4_GROUP',
    COMMENT              => 'Apps group 4',
    UTILIZATION_LIMIT    => 30);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                 => 'apps_plan',
    GROUP_OR_SUBPLAN     => 'OTHER_GROUPS',
    COMMENT              => 'Mandatory',
    UTILIZATION_LIMIT    => 20);

  DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

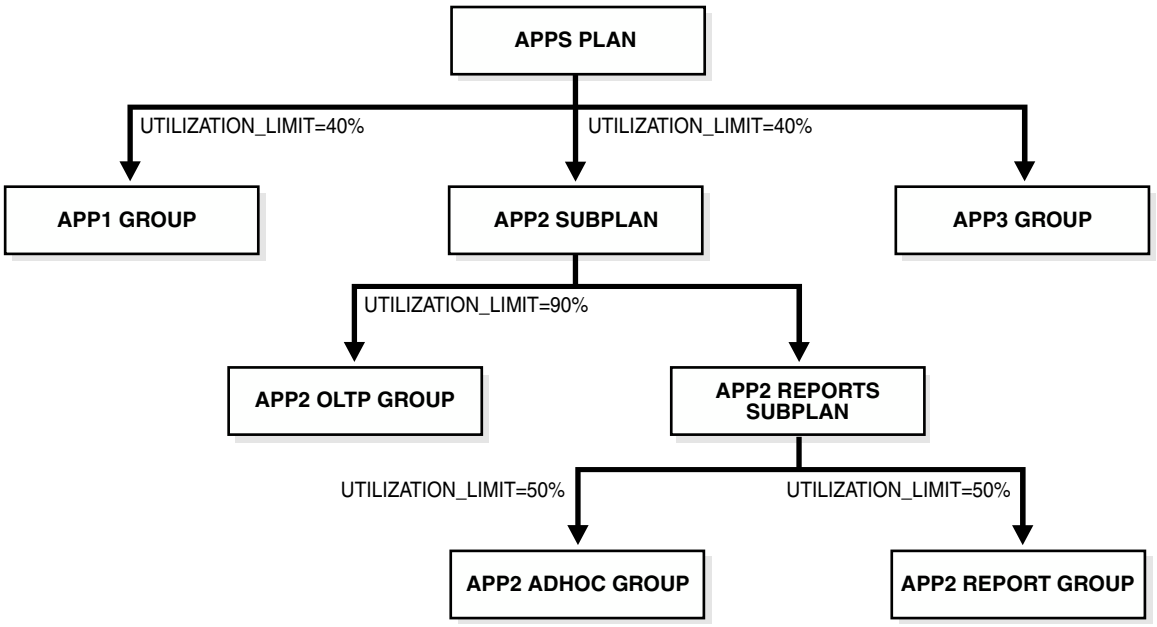
If all four application groups can fully use the CPU allocated to them (30% in this case), then the minimum CPU that is allocated to each application group is computed as a ratio of the

application group's limit to the total of the limits of all application groups. In this example, all four application groups are allocated a utilization limit of 30%. Therefore, when all four groups fully use their limits, the CPU allocation to each group is 30/(30+30+30+30) = 25%.

Example 4 - Specifying a Utilization Limit for Consumer Groups and Subplans

The following example describes how the utilization limit is computed for scenarios, such as the one in Figure 26-8, where you set `UTILIZATION_LIMIT` for a subplan and for consumer groups within the subplan. For simplicity, the requirement to include the `OTHER_GROUPS` consumer group is ignored, and resource plan directives are not shown, even though they are part of the plan.

Figure 26-8 Resource Plan with Maximum Utilization for Subplan and Consumer Groups



The following PL/SQL block creates the plan described in Figure 26-8.

```
BEGIN
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
    DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP1_GROUP',
     COMMENT        => 'Group for application #1');
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP2_OLTP_GROUP',
     COMMENT        => 'Group for OLTP activity in application #2');
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP2_ADHOC_GROUP',
     COMMENT        => 'Group for ad-hoc queries in application #2');
  DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP (
     CONSUMER_GROUP => 'APP2_REPORT_GROUP',
     COMMENT        => 'Group for reports in application #2');
   DBMS_RESOURCE_MANAGER.CREATE_PLAN(
    PLAN    => 'APPS_PLAN',
    COMMENT => 'Plan for managing 3 applications');
  DBMS_RESOURCE_MANAGER.CREATE_PLAN(
    PLAN    => 'APP2_SUBPLAN',
    COMMENT => 'Subplan for managing application #2',
```

```
    SUB_PLAN => TRUE);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN(
    PLAN    => 'APP2_REPORTS_SUBPLAN',
    COMMENT => 'Subplan for managing reports in application #2',
    SUB_PLAN => TRUE);

  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                 => 'APPS_PLAN',
    GROUP_OR_SUBPLAN     => 'APP1_GROUP',
    COMMENT              => 'Limit CPU for application #1 to 40%',
    UTILIZATION_LIMIT    => 40);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                 => 'APPS_PLAN',
    GROUP_OR_SUBPLAN     => 'APP2_SUBPLAN',
    COMMENT              => 'Limit CPU for application #2 to 40%',
    UTILIZATION_LIMIT    => 40);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                 => 'APP2_SUBPLAN',
    GROUP_OR_SUBPLAN     => 'APP2_OLTP_GROUP',
    COMMENT              => 'Limit CPU for OLTP to 90% of application #2',
    UTILIZATION_LIMIT    => 90);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                 => 'APP2_SUBPLAN',
    GROUP_OR_SUBPLAN     => 'APP2_REPORTS_SUBPLAN',
    COMMENT              => 'Subplan for ad-hoc and normal reports for application #2');
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                 => 'APP2_REPORTS_SUBPLAN',
    GROUP_OR_SUBPLAN     => 'APP2_ADHOC_GROUP',
    COMMENT              => 'Limit CPU for ad-hoc queries to 50% of application #2
reports',
    UTILIZATION_LIMIT    => 50);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                 => 'APP2_REPORTS_SUBPLAN',
    GROUP_OR_SUBPLAN     => 'APP2_REPORT_GROUP',
    COMMENT              => 'Limit CPU for reports to 50% of application #2 reports',
    UTILIZATION_LIMIT    => 50);
  DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE (
    PLAN                 => 'APPS_PLAN',
    GROUP_OR_SUBPLAN     => 'OTHER_GROUPS',
    COMMENT              => 'No directives for default users');
   DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

In this example, the maximum CPU utilization for the consumer group `APP1_GROUP` and subplan `APP2_SUBPLAN` is set to 40%. The limit for the consumer groups `APP2_ADHOC_GROUP` and `APP2_REPORT_GROUP` is set to 50%.

Because there is no limit specified for the subplan `APP2_REPORTS_SUBPLAN`, it inherits the limit of its parent subplan `APP2_SUBPLAN`, which is 40%. The absolute limit for the consumer group `APP2_REPORT_GROUP` is computed as 50% of its parent subplan, which is 50% of 40%, or 20%.

Similarly, because the consumer group `APP2_ADHOC_GROUP` is contained in the subplan `APP2_REPORTS_SUBPLAN`, its limit is computed as a percentage of its parent subplan. The utilization limit for the consumer group `APP2_ADHOC_GROUP` is 50% of 40%, or 20%.

The maximum CPU utilization for the consumer group `APP2_OLTP_GROUP` is set to 90%. The parent subplan of `APP2_OLTP_GROUP`, `APP2_SUBPLAN`, has a limit of 40%. Therefore, the absolute limit for the group `APP2_OLTP_GROUP` is 90% of 40%, or 36%.

26.5.3 Example of Using Several Resource Allocation Methods

An example illustrates using several resource allocation methods.

The example presented here could represent a plan for a database supporting a packaged ERP (Enterprise Resource Planning) or CRM (Customer Relationship Management) application. The work in such an environment can be highly varied. There may be a mix of short transactions and quick queries, in combination with longer running batch jobs that include large parallel queries. The goal is to give good response time to OLTP (Online Transaction Processing), while allowing batch jobs to run in parallel.

The plan is summarized in the following table.

Group	CPU Resource Allocation %	Parallel Statement Queuing	Automatic Consumer Group Switching	Maximum Estimated Execution Time	Undo Pool	PGA Limit for Each Session
`oltp`	60%	--	Switch to group: `batch` Switch time: 3 secs	--	200K	20M
`batch`	30%	Parallel server limit: 8 Parallel queue timeout: 600 secs	--	3600 secs	--	--
`OTHER_GROUPS`	10%	--	--	--	--	--

The following statements create the preceding plan, which is named `erp_plan`:

```
BEGIN
DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.CREATE_PLAN(PLAN => 'erp_plan',
  COMMENT => 'Resource plan/method for ERP Database');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'oltp',
  COMMENT => 'Resource consumer group/method for OLTP jobs');
DBMS_RESOURCE_MANAGER.CREATE_CONSUMER_GROUP(CONSUMER_GROUP => 'batch',
  COMMENT => 'Resource consumer group/method for BATCH jobs');
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
  GROUP_OR_SUBPLAN => 'oltp', COMMENT => 'OLTP sessions', MGMT_P1 => 60,
  SWITCH_GROUP => 'batch', SWITCH_TIME => 3, UNDO_POOL => 200,
  SWITCH_FOR_CALL => TRUE, SESSION_PGA_LIMIT => 20);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
  GROUP_OR_SUBPLAN => 'batch', COMMENT => 'BATCH sessions', MGMT_P1 => 30,
  PARALLEL_SERVER_LIMIT => 8, PARALLEL_QUEUE_TIMEOUT => 600,
  MAX_EST_EXEC_TIME => 3600);
DBMS_RESOURCE_MANAGER.CREATE_PLAN_DIRECTIVE(PLAN => 'erp_plan',
  GROUP_OR_SUBPLAN => 'OTHER_GROUPS', COMMENT => 'mandatory', MGMT_P1 => 10);
DBMS_RESOURCE_MANAGER.VALIDATE_PENDING_AREA();
DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

26.5.4 Example of Managing Parallel Statements Using Directive Attributes

An example illustrates managing parallel statements using directive attributes.

A typical data warehousing environment consists of different types of users with varying resource requirements. Users with common processing needs are grouped into a consumer group. The consumer group URGENT_GROUP consists of users who run reports that provide important information to top management. This group generates a large number of parallel queries. Users from the consumer group ETL_GROUP import data from source systems and perform extract, transform, and load (ETL) operations. The group OTHER_GROUPS contains users who execute ad-hoc queries. You must manage the requirements of these diverse groups of users while optimizing performance.

You can use the following directive attributes to manage and optimize the execution of parallel statements:

- MGMT_P*n*
- PARALLEL_SERVER_LIMIT
- PARALLEL_STMT_CRITICAL
- PARALLEL_QUEUE_TIMEOUT
- PQ_TIMEOUT_ACTION
- PARALLEL_DEGREE_LIMIT_P1
- SHARES

Note:

- The MGMT_P*n* management attributes and SHARES attribute control how a parallel statement is selected from the parallel statement queue for execution. You can prioritize the parallel statements of one consumer group over another by setting a higher value for the management attributes of that group.
- In a multitenant environment, if you want more per-workload management, then you can use the SHARES attribute to specify the share of resource allocation for pluggable databases (PDBs), which includes the parallel statement queuing resource. Alternatively, you can use the other directive attributes mentioned above.

Table 26-16 describes the resource allocations of the plan DW_PLAN, which can be used to manage the needs of the data warehouse users. This plan contains the consumer groups URGENT_GROUP, ETL_GROUP, and OTHER_GROUPS. This example demonstrates the use of directive attributes in ensuring that one application or consumer group does not use all the available parallel execution servers.

Table 26-16 Resource Plan with Parallel Statement Directives

Consumer Group	Level 1 CPU Allocation	Level 2 CPU Allocation	Level 3 CPU Allocation	PARALLEL_DEGREE_LIMIT_P1	PARALLEL_SERVER_LIMIT	PARALLEL_QUEUE_TIMEOUT
URGENT_GROUP	100%			12		
ETL_GROUP		100%		8	50%	
OTHER_GROUPS			100%	2	50%	360

In this example, the parameter PARALLEL_SERVERS_TARGET initialization parameter is set to 64, which means that the number of parallel execution servers available is 64. The total number of parallel execution servers that can be used for parallel statement execution before URGENT_GROUP sessions with PARALLEL_DEGREE_POLICY set to AUTO are added to the parallel statement queue is equal to 64. Because the PARALLEL_SERVER_LIMIT attribute of ETL_GROUP and OTHER_GROUPS is 50%, the maximum number of parallel execution servers that can be used by these groups is 50% of 64, or 32 parallel execution servers each.

Note that parallel statements from a consumer group will only be queued if the PARALLEL_DEGREE_POLICY parameter is set to AUTO and the total number of active servers for the consumer group is higher than PARALLEL_SERVERS_TARGET. If PARALLEL_DEGREE_POLICY is set to MANUAL or LIMITED, then the statements are run provided there are enough parallel execution servers available. The parallel execution servers used by such a statement will count toward the total number of parallel execution servers used by the consumer group. However, the parallel statement will not be added to the parallel statement queue.

> **Tip:**
>
> For low-priority applications, it is a common practice to set low values for PARALLEL_DEGREE_LIMIT_P1 and PARALLEL_SERVER_LIMIT.

Because URGENT_GROUP has 100% of the allocation at level 1, its parallel statements will always be dequeued ahead of the other consumer groups from the parallel statement queue. Although URGENT_GROUP has no PARALLEL_SERVER_LIMIT directive attribute, a statement issued by a session in this group might still be queued if there are not enough available parallel execution servers to run it.

When you create the resource plan directive for the URGENT_GROUP, you can set the PARALLEL_STMT_CRITICAL parameter to BYPASS_QUEUE. With this setting, parallel statements from the consumer group bypass the parallel statements queue and are executed immediately. However, the number of parallel execution servers might exceed the setting of the PARALLEL_SERVERS_TARGET initialization parameter, and the degree of parallelism might be lower if the limit set by the PARALLEL_MAX_SERVERS initialization parameter is reached.

The degree of parallelism, represented by PARALLEL_DEGREE_LIMIT_P1, is set to 12 for URGENT_GROUP. Therefore, each parallel statement from URGENT_GROUP can use a maximum of 12 parallel execution servers. Similarly, each parallel statement from the ETL_GROUP can use a maximum of 8 parallel execution servers and each parallel statement from the OTHER_GROUPS can use 2 parallel execution servers.

Suppose, at a given time, the only parallel statements are from the ETL_GROUP, and they are using 26 out of the 32 parallel execution servers available to this group. Sessions from this consumer group have PARALLEL_DEGREE_POLICY set to AUTO. If another parallel statement with the PARALLEL_DEGREE_LIMIT_P1 attribute set to 8 is launched from ETL_GROUP, then this query cannot be run immediately because the available parallel execution servers in the ETL_GROUP is 32-26=6 parallel execution servers. The new parallel statement is queued until the number of parallel execution servers it requires is available in ETL_GROUP.

While the parallel statements in ETL_GROUP are being executed, suppose a parallel statement is launched from OTHER_GROUPS. This group still has 32 parallel execution servers available and so the parallel statement is executed.

The PARALLEL_QUEUE_TIMEOUT attribute for OTHER_GROUPS is set to 360. Therefore, any parallel statement from this group can remain in the parallel execution server queue for 360 seconds

only. After this time, the parallel statement is removed from the queue and the error ORA-07454 is returned.

See Also:

- "Parallel Execution Servers"
- "Creating Resource Plan Directives "

26.5.5 An Oracle-Supplied Mixed Workload Plan

Oracle Database includes a predefined resource plan, MIXED_WORKLOAD_PLAN, that prioritizes interactive operations over batch operations, and includes the required subplans and consumer groups recommended by Oracle.

MIXED_WORKLOAD_PLAN is defined as follows:

Group or Subplan	CPU Resource Allocation				
	Level 1	Level 2	Level 3	Automatic Consumer Group Switching	Max Degree of Parallelism
BATCH_GROUP			100%		
INTERACTIVE_GROUP		85%		Switch to group: BATCH_GROUP Switch time: 60 seconds Switch for call: TRUE	1
ORA$AUTOTASK		5%			
OTHER_GROUPS		5%			
SYS_GROUP	100%				

In this plan, because INTERACTIVE_GROUP is intended for short transactions, any call that consumes more than 60 seconds of CPU time is automatically switched to BATCH_GROUP, which is intended for longer batch operations.

You can use this predefined plan if it is appropriate for your environment. (You can modify the plan, or delete it if you do not intend to use it.) Note that there is nothing special about the names BATCH_GROUP and INTERACTIVE_GROUP. The names reflect only the intended purposes of the groups, and it is up to you to map application sessions to these groups and adjust CPU resource allocation percentages accordingly so that you achieve proper resource management for your interactive and batch applications. For example, to ensure that your interactive applications run under the INTERACTIVE_GROUP consumer group, you must map your interactive applications' user sessions to this consumer group based on user name, service name, program name, module name, or action, as described in "Specifying Session-to-Consumer Group Mapping Rules". You must map your batch applications to the BATCH_GROUP in the same way. Finally, you must enable this plan as described in "Enabling Oracle Database Resource Manager and Switching Plans".

See Table 26-17 and Table 26-18 for explanations of the other resource consumer groups and subplans in this plan.

26.6 Managing Multiple Database Instances on a Single Server

Oracle Database provides a method for managing CPU allocations on a multi-CPU server running multiple database instances. This method is called instance caging. Instance caging and Oracle Database Resource Manager (the Resource Manager) work together to support desired levels of service across multiple instances.

- About Instance Caging
  A simple way to limit CPU consumption for each database instance is to use instance caging. **Instance caging** is a method that uses an initialization parameter to limit the number of CPUs that an instance can use simultaneously.
- Enabling Instance Caging
  You can enable instance caging using by creating a resource plan with CPU directives and setting the `CPU_COUNT` initialization parameter.

26.6.1 About Instance Caging

A simple way to limit CPU consumption for each database instance is to use instance caging. **Instance caging** is a method that uses an initialization parameter to limit the number of CPUs that an instance can use simultaneously.

You might decide to run multiple Oracle database instances on a single multi-CPU server. A typical reason to do so would be server consolidation—using available hardware resources more efficiently. When running multiple instances on a single server, the instances compete for CPU. One resource-intensive database instance could significantly degrade the performance of the other instances. For example, on a 16-CPU system with four database instances, the operating system might be running one database instance on the majority of the CPUs during a period of heavy load for that instance. This could degrade performance in the other three instances. CPU allocation decisions such as this are made solely by the operating system; the user generally has no control over them.

In the previous example, if you use instance caging to limit the number of CPUs to four for each of the four instances, there is less likelihood that one instance can interfere with the others. When constrained to four CPUs, an instance might become CPU-bound. This is when the Resource Manager begins to do its work to allocate CPU among the various database sessions according to the resource plan that you set for the instance. Thus, instance caging and the Resource Manager together provide a simple, effective way to manage multiple instances on a single server.

There are two typical approaches to instance caging for a server:

- Over-subscribing—You would use this approach for non-critical databases such as development and test systems, or low-load non-critical production systems. In this approach, the sum of the CPU limits for each instance exceeds the actual number of CPUs on the system. For example, on a 4-CPU system with four database instances, you might limit each instance to three CPUs. When a server is over-subscribed in this way, the instances can impact each other's performance. However, instance caging limits the impact and helps provide somewhat predictable performance. However, if one of the instances has a period of high load, the CPUs are available to handle it. This is a reasonable approach for non-critical systems, because one or more of the instances may frequently be idle or at a very low load.

- Partitioning—This approach is for critical production systems, where you want to prevent instances from interfering with each other. You allocate CPUs such that the sum of all allocations is equal to the number of CPUs on the server. For example, on a 16-server system, you might allocate 8 CPUs to the first instance, 4 CPUs to the second, and 2 each to the remaining two instances. By dedicating CPU resources to each database instance, the load on one instance cannot affect another's, and each instance performs predictably.

Using Instance Caging with Utilization Limit

If you enable instance caging and set a utilization limit in your resource plan, then the absolute limit is computed as a percentage of the allocated CPU resources.

For example, if you enable instance caging and set the `CPU_COUNT` to 4, and a consumer group has a utilization limit of 50%, then the consumer group can use a maximum of 50% of 4 CPUs, which is 2 CPUs.

26.6.2 Enabling Instance Caging

You can enable instance caging using by creating a resource plan with CPU directives and setting the `CPU_COUNT` initialization parameter.

To enable instance caging, do the following for each instance on the server:

1. Enable the Resource Manager by assigning a resource plan, and ensure that the resource plan has CPU directives, using the `MGMT_P1` through `MGMT_P8` parameters.

   See "Enabling Oracle Database Resource Manager and Switching Plans" for instructions.

2. Set the `cpu_count` initialization parameter.

   This is a dynamic parameter, and can be set with the following statement:

   ```
   ALTER SYSTEM SET CPU_COUNT = 4;
   ```

26.7 Maintaining Consumer Groups, Plans, and Directives

You can maintain consumer groups, resource plans, and resource plan directives for Oracle Database Resource Manager (the Resource Manager). You perform maintenance tasks using the `DBMS_RESOURCE_MANAGER` PL/SQL package.

- Updating a Consumer Group
  You use the `UPDATE_CONSUMER_GROUP` procedure to update consumer group information.
- Deleting a Consumer Group
  The `DELETE_CONSUMER_GROUP` procedure deletes the specified consumer group.
- Updating a Plan
  You use the `UPDATE_PLAN` procedure to update plan information.
- Deleting a Plan
  The `DELETE_PLAN` procedure deletes the specified plan as well as all the plan directives associated with it.
- Updating a Resource Plan Directive
  Use the `UPDATE_PLAN_DIRECTIVE` procedure to update plan directives.
- Deleting a Resource Plan Directive
  To delete a resource plan directive, use the `DELETE_PLAN_DIRECTIVE` procedure.

> **See Also:**
>
> - Predefined Consumer Group Mapping Rules
> - *Oracle Database PL/SQL Packages and Types Reference* for details on the `DBMS_RESOURCE_MANAGER` PL/SQL package.

26.7.1 Updating a Consumer Group

You use the `UPDATE_CONSUMER_GROUP` procedure to update consumer group information.

To update a consumer group:

1. Create a pending area.
2. Run the `UPDATE_CONSUMER_GROUP` procedure .

   If you do not specify the arguments for the `UPDATE_CONSUMER_GROUP` procedure, then they remain unchanged in the data dictionary.
3. Submit the pending area.

Related Topics

- Creating a Pending Area
  You create a pending area with the `CREATE_PENDING_AREA` procedure.
- Submitting the Pending Area
  After you have validated your changes, call the `SUBMIT_PENDING_AREA` procedure to make your changes active.

26.7.2 Deleting a Consumer Group

The `DELETE_CONSUMER_GROUP` procedure deletes the specified consumer group.

To delete a consumer group:

1. Create a pending area.
2. Run the `DELETE_CONSUMER_GROUP` procedure .
3. Submit the pending area.

Upon deletion of a consumer group, all users having the deleted group as their initial consumer group are assigned the `OTHER_GROUPS` as their initial consumer group. All currently running sessions belonging to a deleted consumer group are assigned to a new consumer group, based on the consumer group mapping rules. If no consumer group is found for a session through mapping, the session is switched to the `OTHER_GROUPS`.

You cannot delete a consumer group if it is referenced by a resource plan directive.

Related Topics

- Creating a Pending Area
  You create a pending area with the `CREATE_PENDING_AREA` procedure.
- Submitting the Pending Area
  After you have validated your changes, call the `SUBMIT_PENDING_AREA` procedure to make your changes active.

26.7.3 Updating a Plan

You use the UPDATE_PLAN procedure to update plan information.

To update a plan:

1. Create a pending area.
2. Run the UPDATE_PLAN procedure. For example, the following PL/SQL block updates the COMMENT parameter:

```
BEGIN
  DBMS_RESOURCE_MANAGER.UPDATE_PLAN(
   PLAN => 'DAYTIME',
   NEW_COMMENT => '50% more resources for OLTP applications');
END;
/
```

   If you do not specify the arguments for the UPDATE_PLAN procedure, they remain unchanged in the data dictionary.
3. Submit the pending area.

Related Topics

- Creating a Pending Area
  You create a pending area with the CREATE_PENDING_AREA procedure.
- Submitting the Pending Area
  After you have validated your changes, call the SUBMIT_PENDING_AREA procedure to make your changes active.

26.7.4 Deleting a Plan

The DELETE_PLAN procedure deletes the specified plan as well as all the plan directives associated with it.

To delete a plan:

1. Create a pending area.
2. Run the DELETE_PLAN_CASCADE procedure. For example, the following PL/SQL block deletes the great_bread plan and its directives.

```
BEGIN
  DBMS_RESOURCE_MANAGER.DELETE_PLAN(PLAN => 'great_bread');
END;
/
```

   If you do not specify the arguments for the UPDATE_PLAN procedure, they remain unchanged in the data dictionary.

   The resource consumer groups referenced by the deleted directives are not deleted, but they are no longer associated with the great_bread plan.

   The DELETE_PLAN_CASCADE procedure deletes the specified plan as well as all its descendants: plan directives and those subplans and resource consumer groups that are not marked by the database as mandatory. If DELETE_PLAN_CASCADE encounters an error, then it rolls back, leaving the plan unchanged.

   You cannot delete the currently active plan.

3. Submit the pending area.

Related Topics

- Creating a Pending Area
  You create a pending area with the `CREATE_PENDING_AREA` procedure.
- Submitting the Pending Area
  After you have validated your changes, call the `SUBMIT_PENDING_AREA` procedure to make your changes active.

26.7.5 Updating a Resource Plan Directive

Use the `UPDATE_PLAN_DIRECTIVE` procedure to update plan directives.

To update a resource plan directive:

1. Create a pending area.
2. Run the `UPDATE_PLAN_DIRECTIVE` procedure.

   The following example adds a comment to a directive:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CLEAR_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.UPDATE_PLAN_DIRECTIVE(
         PLAN             => 'SIMPLE_PLAN1',
         GROUP_OR_SUBPLAN => 'MYGROUP1',
         NEW_COMMENT      => 'Higher priority'
        );
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

   To clear (nullify) a comment, pass a null string (`''`). To clear (zero or nullify) any numeric directive parameter, set its new value to -1:

```
BEGIN
  DBMS_RESOURCE_MANAGER.CLEAR_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.CREATE_PENDING_AREA();
  DBMS_RESOURCE_MANAGER.UPDATE_PLAN_DIRECTIVE(
         PLAN                  => 'SIMPLE_PLAN1',
         GROUP_OR_SUBPLAN      => 'MYGROUP1',
         NEW_MAX_EST_EXEC_TIME => -1
        );
  DBMS_RESOURCE_MANAGER.SUBMIT_PENDING_AREA();
END;
/
```

   If you do not specify an argument for the `UPDATE_PLAN_DIRECTIVE` procedure, then its corresponding parameter in the directive remains unchanged.

3. Submit the pending area.

Related Topics

- Creating a Pending Area
  You create a pending area with the `CREATE_PENDING_AREA` procedure.
- Submitting the Pending Area
  After you have validated your changes, call the `SUBMIT_PENDING_AREA` procedure to make your changes active.

26.7.6 Deleting a Resource Plan Directive

To delete a resource plan directive, use the `DELETE_PLAN_DIRECTIVE` procedure.

To delete a resource plan directive:

1. Create a pending area.
2. Run the `DELETE_PLAN_DIRECTIVE` procedure.
3. Submit the pending area.

Related Topics

- Creating a Pending Area
  You create a pending area with the `CREATE_PENDING_AREA` procedure.
- Submitting the Pending Area
  After you have validated your changes, call the `SUBMIT_PENDING_AREA` procedure to make your changes active.

26.8 Viewing Database Resource Manager Configuration and Status

You can use several static data dictionary views and dynamic performance views to view the current configuration and status of Oracle Database Resource Manager (the Resource Manager).

- About Resource Manager Views
  A set of dynamic performance views enable you to monitor the results of your Oracle Database Resource Manager settings.
- Viewing Consumer Groups Granted to Users or Roles
  The `DBA_RSRC_CONSUMER_GROUP_PRIVS` view displays the consumer groups granted to users or roles.
- Viewing Plan Information
  An example illustrates using the `DBA_RSRC_PLANS` view to display all of the resource plans defined in the database.
- Viewing Current Consumer Groups for Sessions
  You can use the `V$SESSION` view to display the consumer groups that are currently assigned to sessions.
- Viewing the Currently Active Plans
  The `V$RSRC_PLAN` view displays currently active plans.
- Monitoring PDBs Managed by Oracle Database Resource Manager
  A set of dynamic performance views enables you to monitor the results of your Oracle Database Resource Manager settings for PDBs.

> **See Also:**
>
> *Oracle Database Reference* for details on all static data dictionary views and dynamic performance views

26.8.1 About Resource Manager Views

A set of dynamic performance views enable you to monitor the results of your Oracle Database Resource Manager settings.

Use the following dynamic performance views to help you monitor the results of your Oracle Database Resource Manager settings:

- V$RSRC_PLAN
- V$RSRC_CONSUMER_GROUP
- V$RSRC_SESSION_INFO
- V$RSRC_PLAN_HISTORY
- V$RSRC_CONS_GROUP_HISTORY
- V$RSRCMGRMETRIC
- V$RSRCMGRMETRIC_HISTORY

These views provide:

- Current status information
- History of resource plan activations
- Current and historical statistics on resource consumption and CPU waits by both resource consumer group and session

In addition, historical statistics are available through the `DBA_HIST_RSRC_PLAN` and `DBA_HIST_RSRC_CONSUMER_GROUP` views, which contain Automatic Workload Repository (AWR) snapshots of the `V$RSRC_PLAN_HISTORY` and `V$RSRC_CONS_GROUP_HISTORY`, respectively.

For assistance with tuning, the views `V$RSRCMGRMETRIC` and `V$RSRCMGRMETRIC_HISTORY` show how much time was spent waiting for CPU and how much CPU was consumed per minute for every consumer group for the past hour. These metrics can also be viewed graphically with Cloud Control, on the Resource Manager Statistics page.

When Resource Manager is enabled, Resource Manager automatically records statistics about resource usage, and you can examine these statistics using real-time SQL monitoring and Resource Manager dynamic performance views.

You can use real-time SQL monitoring by accessing the SQL Monitor page in Cloud Control or by querying the `V$SQL_MONITOR` view and other related views. The `V$SQL_MONITOR` view also includes information about the last action performed by Resource Manager for a consumer group in the following columns: `RM_CONSUMER_GROUP`, `RM_LAST_ACTION`, `RM_LAST_ACTION_REASON`, and `RM_LAST_ACTION_TIME`.

In addition, the following dynamic performance views contain statistics about resource usage:

- `V$RSRCMGRMETRIC`
- `V$RSRCMGRMETRIC_HISTORY`
- `V$RSRC_CONSUMER_GROUP`
- `V$RSRC_CONS_GROUP_HISTORY`

> **See Also:**
>
> *Oracle Database SQL Tuning Guide* for more information about real-time SQL monitoring

V$RSRC_PLAN

This view displays the currently active resource plan and its subplans.

```
SELECT name, is_top_plan FROM v$rsrc_plan;

NAME                             IS_TOP_PLAN
-------------------------------- -----------
DEFAULT_PLAN                     TRUE
ORA$AUTOTASK                     FALSE
ORA$AUTOTASK_HIGH_SUB_PLAN       FALSE
```

The plan for which `IS_TOP_PLAN` is `TRUE` is the currently active (top) plan, and the other plans are subplans of either the top plan or of other subplans in the list.

This view also contains other information, including the following:

- The `INSTANCE_CAGING` column shows whether instance caging is enabled.
- The `CPU_MANAGED` column shows whether CPU is being managed.
- The `PARALLEL_EXECUTION_MANAGED` column shows whether parallel statement queuing is enabled.

> **See Also:**
>
> *Oracle Database Reference*

V$RSRC_CONSUMER_GROUP

Use the `V$RSRC_CONSUMER_GROUP` view to monitor resources consumed, including CPU, I/O, and parallel execution servers. It can also be used to monitor statistics related to CPU resource management, runaway query management, parallel statement queuing, and so on. All of the statistics are cumulative from the time when the plan was activated.

```
SELECT name, active_sessions, queue_length,
  consumed_cpu_time, cpu_waits, cpu_wait_time
  FROM v$rsrc_consumer_group;

NAME               ACTIVE_SESSIONS QUEUE_LENGTH CONSUMED_CPU_TIME  CPU_WAITS CPU_WAIT_TIME
------------------ --------------- ------------ ----------------- ---------- -------------
OLTP_ORDER_ENTRY                 1            0             29690        467          6709
OTHER_GROUPS                     0            0           5982366       4089         60425
SYS_GROUP                        1            0           2420704        914         19540
DSS_QUERIES                      4            2           4594660       3004         55700
```

In the preceding query results, the `DSS_QUERIES` consumer group has four sessions in its active session pool and two more sessions queued for activation.

A key measure in this view is CPU_WAIT_TIME. This indicates the total time that sessions in the consumer group waited for CPU because of resource management. Not included in this measure are waits due to latch or enqueue contention, I/O waits, and so on.

> **Note:**
>
> The V$RSRC_CONSUMER_GROUP view records statistics for resources that are not currently being managed by Resource Manager. when the STATISTICS_LEVEL initialization parameter is set to ALL or TYPICAL.

> **See Also:**
>
> *Oracle Database Reference*

V$RSRC_SESSION_INFO

Use this view to monitor the status of one or more sessions. The view shows how the session has been affected by the Resource Manager. It provides information such as:

- The consumer group that the session currently belongs to.
- The consumer group that the session originally belonged to.
- The session attribute that was used to map the session to the consumer group.
- Session state (RUNNING, WAIT_FOR_CPU, QUEUED, and so on).
- Current and cumulative statistics for metrics, such as CPU consumed, wait times, queued time, and number of active parallel servers used. Current statistics reflect statistics for the session since it joined its current consumer group. Cumulative statistics reflect statistics for the session in all consumer groups to which it has belonged since it was created.

```
SELECT se.sid sess_id, co.name consumer_group,
 se.state, se.consumed_cpu_time cpu_time, se.cpu_wait_time, se.queued_time
 FROM v$rsrc_session_info se, v$rsrc_consumer_group co
 WHERE se.current_consumer_group_id = co.id;

SESS_ID CONSUMER_GROUP     STATE     CPU_TIME CPU_WAIT_TIME QUEUED_TIME
------- ------------------ -------- --------- ------------- -----------
    113 OLTP_ORDER_ENTRY   WAITING     137947         28846           0
    135 OTHER_GROUPS       IDLE        785669         11126           0
    124 OTHER_GROUPS       WAITING      50401         14326           0
    114 SYS_GROUP          RUNNING        495             0           0
    102 SYS_GROUP          IDLE         88054            80           0
    147 DSS_QUERIES        WAITING     460910        512154           0
```

CPU_WAIT_TIME in this view has the same meaning as in the V$RSRC_CONSUMER_GROUP view, but applied to an individual session.

You can join this view with the V$SESSION view for more information about a session.

> **See Also:**
>
> - *Oracle Database Reference*
> - *Oracle Database VLDB and Partitioning Guide*

V$RSRC_PLAN_HISTORY

This view shows when resource plans were enabled or disabled on the instance. Each resource plan activation or deactivation is assigned a sequence number. For each entry in the view, the `V$RSRC_CONS_GROUP_HISTORY` view has a corresponding entry for each consumer group in the plan that shows the cumulative statistics for the consumer group. The two views are joined by the `SEQUENCE#` column in each.

```
SELECT sequence# seq, name plan_name,
to_char(start_time, 'DD-MON-YY HH24:MM') start_time,
to_char(end_time, 'DD-MON-YY HH24:MM') end_time, window_name
FROM v$rsrc_plan_history;

 SEQ PLAN_NAME                  START_TIME      END_TIME        WINDOW_NAME
---- -------------------------- --------------- --------------- ----------------
   1                            29-MAY-07 23:05 29-MAY-07 23:05
   2 DEFAULT_MAINTENANCE_PLAN   29-MAY-07 23:05 30-MAY-07 02:05 TUESDAY_WINDOW
   3                            30-MAY-07 02:05 30-MAY-07 22:05
   4 DEFAULT_MAINTENANCE_PLAN   30-MAY-07 22:05 31-MAY-07 02:05 WEDNESDAY_WINDOW
   5                            31-MAY-07 02:05 31-MAY-07 22:05
   6 DEFAULT_MAINTENANCE_PLAN   31-MAY-07 22:05                 THURSDAY_WINDOW
```

A null value under `PLAN_NAME` indicates that no plan was active.

AWR snapshots of this view are stored in the `DBA_HIST_RSRC_PLAN` view.

> **See Also:**
>
> *Oracle Database Reference*

V$RSRC_CONS_GROUP_HISTORY

This view helps you understand how resources were shared among the consumer groups over time. The `sequence#` column corresponds to the column of the same name in the `V$RSRC_PLAN_HISTORY` view. Therefore, you can determine the plan that was active for each row of consumer group statistics.

```
SELECT sequence# seq, name, cpu_wait_time, cpu_waits,
consumed_cpu_time FROM v$rsrc_cons_group_history;

 SEQ NAME                     CPU_WAIT_TIME  CPU_WAITS CONSUMED_CPU_TIME
---- ------------------------ ------------- ---------- ----------------
   2 SYS_GROUP                        18133        691         33364431
   2 OTHER_GROUPS                     51252        825        181058333
   2 ORA$AUTOTASK_MEDIUM_GROUP           21          5          4019709
   2 ORA$AUTOTASK_URGENT_GROUP           35          1           198760
   2 ORA$AUTOTASK_STATS_GROUP             0          0                0
   2 ORA$AUTOTASK_SPACE_GROUP             0          0                0
   2 ORA$AUTOTASK_SQL_GROUP               0          0                0
```

```
2 ORA$AUTOTASK_HEALTH_GROUP             0          0                0
4 SYS_GROUP                         40344         85         42519265
4 OTHER_GROUPS                     123295       1040        371481422
4 ORA$AUTOTASK_MEDIUM_GROUP             1          4          7433002
4 ORA$AUTOTASK_URGENT_GROUP         22959        158         19964703
4 ORA$AUTOTASK_STATS_GROUP              0          0                0
   .
   .
```

AWR snapshots of this view are stored in the `DBA_HIST_RSRC_CONSUMER_GROUP` view. Use `DBA_HIST_RSRC_CONSUMER_GROUP` with `DBA_HIST_RSRC_PLAN` to determine the plan that was active for each historical set of consumer group statistics.

Note:

The `V$RSRC_CONS_GROUP_HISTORY` view records statistics for resources that are not currently being managed by Resource Manager. when the `STATISTICS_LEVEL` initialization parameter is set to `ALL` or `TYPICAL`.

See Also:

- *Oracle Database Reference*
- *Oracle Database Performance Tuning Guide* for information about the AWR.

V$RSRCMGRMETRIC

This view enables you to track CPU metrics in milliseconds, in terms of number of sessions, or in terms of utilization for the past one minute. It provides real-time metrics for each consumer group and is very useful in scenarios where you are running workloads and want to continuously monitor CPU resource utilization.

Use this view to compare the maximum possible CPU utilization and average CPU utilization percentage for consumer groups with other consumer group settings such as CPU time used, time waiting for CPU, average number of sessions that are consuming CPU, and number of sessions that are waiting for CPU allocation. For example, you can view the amount of CPU resources a consumer group used and how long it waited for resource allocation. Or, you can view how many sessions from each consumer group are executed against the total number of active sessions.

To track CPU consumption in terms of CPU utilization, use the `CPU_UTILIZATION_LIMIT` and `AVG_CPU_UTILIZATION` columns. `AVG_CPU_UTILIZATION` lists the average percentage of the server's CPU that is consumed by a consumer group. `CPU_UTILIZATION_LIMIT` represents the maximum percentage of the server's CPU that a consumer group can use. This limit is set using the `UTILIZATION_LIMIT` directive attribute.

```
SELECT consumer_group_name, cpu_utilization_limit,
avg_cpu_utilization FROM v$rsrcmgrmetric;
```

Use the `CPU_CONSUMED_TIME` and `CPU_TIME_WAIT` columns to track CPU consumption and throttling in milliseconds. The column `NUM_CPUS` represents the number of CPUs that Resource Manager is managing.

```
SELECT consumer_group_name, cpu_consumed_time,
cpu_wait_time, num_cpus FROM v$rsrcmgrmetric;
```

To track the CPU consumption and throttling in terms of number of sessions, use the `RUNNING_SESSIONS_LIMIT`, `AVG_RUNNING_SESSIONS`, and `AVG_WAITING_SESSIONS` columns. `RUNNING_SESSIONS_LIMIT` lists the maximum number of sessions, from a particular consumer group, that can be running at any time. This limit is defined by the `UTILIZATION_LIMIT` directive attribute that you set either for the consumer group or for a subplan that contains the consumer group. For each consumer group, `AVG_RUNNING_SESSIONS` lists the average number of sessions that are consuming CPU and `AVG_WAITING_SESSIONS` lists the average number of sessions that are waiting for CPU.

```
SELECT sequence#, consumer_group_name, running_sessions_limit,
avg_running_sessions, avg_waiting_sessions FROM v$rsrcmgrmetric;
```

To track parallel statements and parallel server use for a consumer group, use the `AVG_ACTIVE_PARALLEL_STMTS`, `AVG_QUEUED_PARALLEL_STMTS`, `AVG_ACTIVE_PARALLEL_SERVERS`, `AVG_QUEUED_PARALLEL_SERVERS`, and `PARALLEL_SERVERS_LIMIT` columns. `AVG_ACTIVE_PARALLEL_STMTS` and `AVG_ACTIVE_PARALLEL_SERVERS` list the average number of parallel statements running and the average number of parallel servers used by the parallel statements. `AVG_QUEUED_PARALLEL_STMTS` and `AVG_QUEUED_PARALLEL_SERVERS` list the average number of parallel statements queued and average number of parallel servers that were requested by queued parallel statements. `PARALLEL_SERVERS_LIMIT` lists the number of parallel servers allowed to be used by the consumer group.

```
SELECT avg_active_parallel_stmts, avg_queued_parallel_stmts,
avg_active_parallel_servers, avg_queued_parallel_servers, parallel_servers_limit
FROM v$rsrcmgrmetric;
```

> **Note:**
>
> The `V$RSRCMGRMETRIC` view records statistics for resources that are not currently being managed by Resource Manager. when the `STATISTICS_LEVEL` initialization parameter is set to `ALL` or `TYPICAL`.

> **See Also:**
>
> *Oracle Database Reference*

V$RSRCMGRMETRIC_HISTORY

The columns in the `V$RSRCMGRMETRIC_HISTORY` are the same view as V$RSRCMGRMETRIC. The only difference between these views is that V$RSRCMGRMETRIC contains metrics for the past one minute only, whereas V$`RSRCMGRMETRIC_HISTORY` contains metrics for the last 60 minutes.

> **Note:**
>
> The V$RSRCMGRMETRIC_HISTORY view records statistics for resources that are not currently being managed by Resource Manager. when the STATISTICS_LEVEL initialization parameter is set to ALL or TYPICAL.

> **See Also:**
>
> *Oracle Database Reference*

26.8.2 Viewing Consumer Groups Granted to Users or Roles

The DBA_RSRC_CONSUMER_GROUP_PRIVS view displays the consumer groups granted to users or roles.

Specifically, it displays the groups to which a user or role is allowed to belong or be switched. For example, in the view shown below, user SCOTT always starts in the SALES consumer group, can switch to the MARKETING group through a specific grant, and can switch to the DEFAULT_CONSUMER_GROUP (OTHER_GROUPS) and LOW_GROUP groups because they are granted to PUBLIC. SCOTT also can grant the SALES group but not the MARKETING group to other users.

```
SELECT * FROM dba_rsrc_consumer_group_privs;

GRANTEE            GRANTED_GROUP                  GRANT_OPTION INITIAL_GROUP
------------------ ------------------------------ ------------ -------------
PUBLIC             DEFAULT_CONSUMER_GROUP         YES          YES
PUBLIC             LOW_GROUP                      NO           NO
SCOTT              MARKETING                      NO           NO
SCOTT              SALES                          YES          YES
SYSTEM             SYS_GROUP                      NO           YES
```

SCOTT was granted the ability to switch to these groups using the DBMS_RESOURCE_MANAGER_PRIVS package.

26.8.3 Viewing Plan Information

An example illustrates using the DBA_RSRC_PLANS view to display all of the resource plans defined in the database.

All plans have a NULL status, meaning that they are not in the pending area.

> **Note:**
>
> Plans in the pending area have a status of PENDING. Plans in the pending area are being edited.

```
SELECT plan,status,comments FROM dba_rsrc_plans;
```

```
PLAN                         STATUS   COMMENTS
---------------------------- -------- ---------------------------------------
DSS_PLAN                              Example plan for DSS workloads that prio...
ETL_CRITICAL_PLAN                     Example plan for DSS workloads that prio...
MIXED_WORKLOAD_PLAN                   Example plan for a mixed workload that p...
DEFAULT_MAINTENANCE_PLAN              Default plan for maintenance windows tha...
DEFAULT_PLAN                          Default, basic, pre-defined plan that pr...
INTERNAL_QUIESCE                      Plan for quiescing the database.  This p...
INTERNAL_PLAN                         Internally-used plan for disabling the r...
.
.
.
```

26.8.4 Viewing Current Consumer Groups for Sessions

You can use the `V$SESSION` view to display the consumer groups that are currently assigned to sessions.

The following example queries the `V$SESSION` view:

```
SELECT sid,serial#,username,resource_consumer_group FROM v$session;

SID    SERIAL#  USERNAME                  RESOURCE_CONSUMER_GROUP
-----  -------  ------------------------  --------------------------------
   11       136 SYS                       SYS_GROUP
   13     16570 SCOTT                     SALES
   ...
```

26.8.5 Viewing the Currently Active Plans

The `V$RSRC_PLAN` view displays currently active plans.

This example sets `mydb_plan`, as created by the example shown in "Multilevel Plan Example", as the top level plan. It then queries the `V$RSRC_PLAN` view to display the currently active plans. The view displays the current top level plan and all of its descendent subplans.

```
ALTER SYSTEM SET RESOURCE_MANAGER_PLAN = mydb_plan;

System altered.

SELECT name, is_top_plan FROM v$rsrc_plan;

NAME             IS_TOP_PLAN
---------------------------
MYDB_PLAN        TRUE
MAILDB_PLAN      FALSE
BUGDB_PLAN       FALSE
```

26.8.6 Monitoring PDBs Managed by Oracle Database Resource Manager

A set of dynamic performance views enables you to monitor the results of your Oracle Database Resource Manager settings for PDBs.

- About Resource Manager Views for PDBs
  You can monitor the results of your Oracle Database Resource Manager settings for PDBs using views.

- Monitoring CPU Usage for PDBs
  The V$RSRCPDBMETRIC view enables you to track CPU metrics in milliseconds, in terms of number of sessions, or in terms of utilization for the past one minute.
- Monitoring Parallel Execution for PDBs
  The V$RSRCPDBMETRIC view enables you to track parallel statements and parallel server use for PDBs.
- Monitoring the I/O Generated by PDBs
  The V$RSRCPDBMETRIC view enables you to track the amount of I/O generated by PDBs.
- Monitoring Memory Usage for PDBs
  The V$RSRCPDBMETRIC view enables you to track the amount memory used by PDBs.

26.8.6.1 About Resource Manager Views for PDBs

You can monitor the results of your Oracle Database Resource Manager settings for PDBs using views.

The following views are available:

- V$RSRCPDBMETRIC

  The V$RSRCPDBMETRIC view provides current statistics on resource consumption for PDBs, including CPU usage, parallel execution, I/O generated, and memory usage.

- V$RSRCPDBMETRIC_HISTORY

  The columns in the V$RSRCPDBMETRIC_HISTORY view are the same as the columns in the V$RSRCPDBMETRIC view. The only difference between these views is that the V$RSRCPDBMETRIC view contains metrics for the past one minute only, whereas the V$RSRCPDBMETRIC_HISTORY view contains metrics for the last 60 minutes.

- V$RSRC_PDB

  The V$RSRC_PDB view provides cumulative statistics. The statistics are accumulated since the time that the CDB resource plan was set.

- DBA_HIST_RSRC_PDB_METRIC

  This view contains the historical statistics of V$RSRCPDBMETRIC_HISTORY, taken using Automatic Workload Repository (AWR) snapshots.

> **Note:**
>
> The V$RSRCPDBMETRIC and V$RSRCPDBMETRIC_HISTORY views record statistics for resources that are not currently being managed by Resource Manager when the STATISTICS_LEVEL initialization parameter is set to ALL or TYPICAL.

> **See Also:**
>
> - *Oracle Database SQL Tuning Guide* for more information about real-time SQL monitoring
> - *Oracle Database Reference* to learn about `V$RSRCPDBMETRIC`, `V$RSRCPDBMETRIC_HISTORY`, `V$RSRC_PDB`, and `DBA_HIST_RSRC_PDB_METRIC`

26.8.6.2 Monitoring CPU Usage for PDBs

The `V$RSRCPDBMETRIC` view enables you to track CPU metrics in milliseconds, in terms of number of sessions, or in terms of utilization for the past one minute.

The view provides real-time metrics for each PDB and is very useful in scenarios where you are running workloads and want to continuously monitor CPU resource utilization.

The active CDB resource plan manages CPU usage for a PDB. Use this view to compare the maximum and average CPU utilization for PDBs with other PDB settings such as the following:

- CPU time used
- Time waiting for CPU
- Average number of sessions that are consuming CPU
- Number of sessions that are waiting for CPU allocation

For example, you can view the amount of CPU resources a PDB used and how long it waited for resource allocation. Alternatively, you can view how many sessions from each PDB are executed against the total number of active sessions.

Tracking CPU Consumption in Terms of CPU Utilization for PDBs

To track CPU consumption in terms of CPU utilization, query the `CPU_UTILIZATION_LIMIT` and `AVG_CPU_UTILIZATION` columns. `AVG_CPU_UTILIZATION` lists the average percentage of the server's CPU that is consumed by a PDB. `CPU_UTILIZATION_LIMIT` represents the maximum percentage of the server's CPU that a PDB can use. This limit is set using the `UTILIZATION_LIMIT` directive attribute.

The following query displays this information by showing the container ID (`CON_ID`) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10

SELECT r.CON_ID,
       p.PDB_NAME,
       r.CPU_UTILIZATION_LIMIT,
       r.AVG_CPU_UTILIZATION
FROM   V$RSRCPDBMETRIC r,
       CDB_PDBS p
WHERE r.CON_ID = p.CON_ID;
```

Tracking CPU Consumption and Throttling for PDBs

Use the CPU_CONSUMED_TIME and CPU_TIME_WAIT columns to track CPU consumption and throttling in milliseconds for each PDB. The column NUM_CPUS represents the number of CPUs that Resource Manager is managing.

The following query displays this information by showing the container ID (CON_ID) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10

SELECT r.CON_ID,
       p.PDB_NAME,
       r.CPU_CONSUMED_TIME,
       r.CPU_WAIT_TIME,
       r.NUM_CPUS
FROM   V$RSRCPDBMETRIC r,
       CDB_PDBS p
WHERE r.CON_ID = p.CON_ID;
```

Tracking CPU Consumption and Throttling in Terms of Number of Sessions for PDBs

To track the CPU consumption and throttling in terms of number of sessions, use the RUNNING_SESSIONS_LIMIT, AVG_RUNNING_SESSIONS, and AVG_WAITING_SESSIONS columns. RUNNING_SESSIONS_LIMIT lists the maximum number of sessions from a particular PDB that can be running at any time. This limit is defined by the UTILIZATION_LIMIT directive attribute that you set for the PDB. AVG_RUNNING_SESSIONS lists the average number of sessions that are consuming CPU, and AVG_WAITING_SESSIONS lists the average number of sessions that are waiting for CPU.

The following query displays this information by showing the container ID (CON_ID) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10

SELECT r.CON_ID,
       p.PDB_NAME,
       r.RUNNING_SESSIONS_LIMIT,
       r.AVG_RUNNING_SESSIONS,
       r.AVG_WAITING_SESSIONS
FROM   V$RSRCPDBMETRIC r,
       CDB_PDBS p
WHERE r.CON_ID = p.CON_ID;
```

26.8.6.3 Monitoring Parallel Execution for PDBs

The V$RSRCPDBMETRIC view enables you to track parallel statements and parallel server use for PDBs.

Parallel execution servers for a PDB are managed with the active CDB resource plan of the PDB's CDB. To track parallel statements and parallel server use for PDBs, use the AVG_ACTIVE_PARALLEL_STMTS, AVG_QUEUED_PARALLEL_STMTS, AVG_ACTIVE_PARALLEL_SERVERS, AVG_QUEUED_PARALLEL_SERVERS, and PARALLEL_SERVERS_LIMIT columns.

AVG_ACTIVE_PARALLEL_STMTS and AVG_ACTIVE_PARALLEL_SERVERS list the average number of parallel statements running and the average number of parallel servers used by the parallel statements. AVG_QUEUED_PARALLEL_STMTS and AVG_QUEUED_PARALLEL_SERVERS list the average number of parallel statements queued and average number of parallel servers that were requested by queued parallel statements. PARALLEL_SERVERS_LIMIT lists the number of parallel servers allowed to be used by the PDB.

The following query displays this information by showing the container ID (CON_ID) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10

SELECT r.CON_ID, p.PDB_NAME, r.AVG_ACTIVE_PARALLEL_STMTS,
r.AVG_QUEUED_PARALLEL_STMTS,
   r.AVG_ACTIVE_PARALLEL_SERVERS, r.AVG_QUEUED_PARALLEL_SERVERS,
r.PARALLEL_SERVERS_LIMIT
   FROM V$RSRCPDBMETRIC r, CDB_PDBS p
   WHERE r.CON_ID = p.CON_ID;
```

26.8.6.4 Monitoring the I/O Generated by PDBs

The V$RSRCPDBMETRIC view enables you to track the amount of I/O generated by PDBs.

I/O is limited for a PDB by setting the MAX_IOPS initialization parameter or the MAX_MBPS initialization parameter in the PDB. Use this view to compare the I/O generated by PDBs in terms of the number of operations each second and the number of megabytes each second.

Tracking the Number of I/O Operations Generated Each Second by PDBs

To track the I/O operations generated each second by PDBs during the previous minute, use the IOPS column.

The following query displays this information by showing the container ID (CON_ID) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10

SELECT r.CON_ID, p.PDB_NAME, r.IOPS
   FROM V$RSRCPDBMETRIC r, CDB_PDBS p
   WHERE r.CON_ID = p.CON_ID;
```

Tracking the Number Megabytes Generated for I/O Operations Each Second by PDBs

To track number of megabytes generated for I/O operations each second by PDBs during the previous minute, use the IOMBPS column.

The following query displays this information by showing the container ID (CON_ID) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10

SELECT r.CON_ID, p.PDB_NAME, r.IOMBPS
   FROM V$RSRCPDBMETRIC r, CDB_PDBS p
   WHERE r.CON_ID = p.CON_ID;
```

26.8.6.5 Monitoring Memory Usage for PDBs

The `V$RSRCPDBMETRIC` view enables you to track the amount memory used by PDBs.

Use this view to track the amount of SGA, PGA, buffer cache, and shared pool memory currently used by PDBs.

To track the current memory usage, in bytes, for specific PDBs, use the `SGA_BYTES`, `PGA_BYTES`, `BUFFER_CACHE_BYTES`, and `SHARED_POOL_BYTES` columns.

The following query displays this information by showing the container ID (`CON_ID`) and name of each PDB:

```
COLUMN PDB_NAME FORMAT A10

SELECT r.CON_ID, p.PDB_NAME, r.SGA_BYTES, r.PGA_BYTES, r.BUFFER_CACHE_BYTES,
r.SHARED_POOL_BYTES
   FROM V$RSRCPDBMETRIC r, CDB_PDBS p
   WHERE r.CON_ID = p.CON_ID;
```

26.9 Interacting with Operating-System Resource Control

Many operating systems provide tools for resource management. These tools often contain "workload manager" or "resource manager" in their names, and are intended to allow multiple applications to share the resources of a single server, using an administrator-defined policy. Examples are Hewlett Packard's Process Resource Manager or Solaris Containers, Zones, and Resource Pools.

- Guidelines for Using Operating-System Resource Control
  Follow guidelines if you use operating-system resource control.

26.9.1 Guidelines for Using Operating-System Resource Control

Follow guidelines if you use operating-system resource control.

If you choose to use operating-system resource control with Oracle Database, then you must use it judiciously, according to the following guidelines:

- If you have multiple instances on a node, and you want to distribute resources among them, then each instance should be assigned to a dedicated operating-system resource manager group or managed entity. To run multiple instances in the managed entity, use instance caging to manage how the CPU resources within the managed entity should be distributed among the instances. When Oracle Database Resource Manager is managing CPU resources, it expects a fixed amount of CPU resources for the instance. Without instance caging, it expects the available CPU resources to be equal to the number of CPUs in the managed entity. With instance caging, it expects the available CPU resources to be equal to the value of the `CPU_COUNT` initialization parameter. If there are less CPU resources than expected, then the Oracle Database Resource Manager is not as effective at enforcing the resource allocations in the resource plan. The PDB-level parameter `CPU_MIN_COUNT` is used to set the PDB share in the resource plan and the PDB-level `CPU_COUNT` to set the PDB utilization limit in the resource plan. See "Managing Multiple Database Instances on a Single Server" for information about instance caging.
- The dedicated entity running all the instance's processes must run at one priority (or resource consumption) level.

- The CPU resources assigned to the dedicated entity cannot be changed more frequently than once every few minutes.

  If the operating-system resource manager is rapidly changing the CPU resources allocated to an Oracle instance, then the Oracle Database Resource Manager might not manage CPU resources effectively. In particular, if the CPU resources allocated to the Oracle instance changes more frequently than every couple of minutes, then these changes might not be observed by Oracle because it only checks for such changes every couple of minutes. In these cases, Oracle Database Resource Manager can over-schedule processes if it concludes that more CPU resources are available than there actually are, and it can under-schedule processes if it concludes that less CPU resources are available than there actually are. If it over-schedules processes, then the `UTILIZATION_LIMIT` directives might be exceeded, and the CPU directives might not be accurately enforced. If it under-schedules processes, then the Oracle instance might not fully use the server's resources.
- Process priority management must not be enabled.
- Management of individual database processes at different priority levels (for example, using the `nice` command on UNIX platforms) is not supported. Severe consequences, including instance crashes, can result. Similar undesirable results are possible if operating-system resource control is permitted to manage the memory to which an Oracle Database instance is pinned.

Related Topics

- CPU-Related Initialization Parameters for PDBs

26.10 Oracle Database Resource Manager Reference

Resource Manager includes predefined resource plans, consumer groups, and consumer groups mapping rules. You can query data dictionary views for information about your Resource Manager configuration.

- Predefined Resource Plans and Consumer Groups
  Oracle Database includes predefined resource plans.
- Predefined Consumer Group Mapping Rules
  Oracle Database includes predefined consumer group mapping rules.
- Resource Manager Data Dictionary Views
  You can query a set of data dictionary views for information relating to database resource management.

26.10.1 Predefined Resource Plans and Consumer Groups

Oracle Database includes predefined resource plans.

Table 26-17 lists the resource plans and Table 26-18 lists the resource consumer groups that are predefined in each Oracle database. You can verify these by querying the views `DBA_RSRC_PLANS` and `DBA_RSRC_CONSUMER_GROUPS`.

The following query displays the CPU allocations in the example plan `DSS_PLAN`:

```
SELECT group_or_subplan, mgmt_p1, mgmt_p2, mgmt_p3, mgmt_p4
   FROM dba_rsrc_plan_directives WHERE plan = 'DSS_PLAN';

GROUP_OR_SUBPLAN                 MGMT_P1    MGMT_P2    MGMT_P3    MGMT_P4
------------------------------ ---------- ---------- ---------- ----------
SYS_GROUP                              75          0          0          0
```

```
DSS_CRITICAL_GROUP                18          0          0          0
DSS_GROUP                          3          0          0          0
ETL_GROUP                          1          0          0          0
BATCH_GROUP                        1          0          0          0
ORA$AUTOTASK                       1          0          0          0
OTHER_GROUPS                       1          0          0          0
```

Table 26-17 Predefined Resource Plans

Resource Plan	Description
DEFAULT_MAINTENANCE_PLAN	Default plan for maintenance windows. See "About Resource Allocations for Automated Maintenance Tasks" for details of this plan. Because maintenance windows are regular Oracle Scheduler windows, you can change the resource plan associated with them, if desired. If you do change a maintenance window resource plan, ensure that you include the subplan ORA$AUTOTASK in the new plan.
DEFAULT_PLAN	Basic default plan that prioritizes SYS_GROUP operations and allocates minimal resources for automated maintenance and diagnostics operations.
DSS_PLAN	Example plan for a data warehouse that prioritizes critical DSS queries over non-critical DSS queries and ETL operations.
ETL_CRITICAL_PLAN	Example plan for a data warehouse that prioritizes ETL operations over DSS queries.
INTERNAL_PLAN	For disabling the resource manager. For internal use only.
INTERNAL_QUIESCE	For quiescing the database. This plan cannot be activated directly. To activate, use the QUIESCE command.
MIXED_WORKLOAD_PLAN	Example plan for a mixed workload that prioritizes interactive operations over batch operations. See "An Oracle-Supplied Mixed Workload Plan" for details.

Table 26-18 Predefined Resource Consumer Groups

Resource Consumer Group	Description
BATCH_GROUP	Consumer group for batch operations. Referenced by the example plan MIXED_WORKLOAD_PLAN.
DSS_CRITICAL_GROUP	Consumer group for critical DSS queries. Referenced by the example plans DSS_PLAN and ETL_CRITICAL_PLAN.
DSS_GROUP	Consumer group for non-critical DSS queries. Referenced by the example plans DSS_PLAN and ETL_CRITICAL_PLAN.
ETL_GROUP	Consumer group for ETL jobs. Referenced by the example plans DSS_PLAN and ETL_CRITICAL_PLAN.
INTERACTIVE_GROUP	Consumer group for interactive, OLTP operations. Referenced by the example plan MIXED_WORKLOAD_PLAN.
LOW_GROUP	Consumer group for low-priority sessions.
ORA$AUTOTASK	Consumer group for maintenance tasks.

Table 26-18 (Cont.) Predefined Resource Consumer Groups

Resource Consumer Group	Description
OTHER_GROUPS	Default consumer group for all sessions that do not have an explicit initial consumer group, are not mapped to a consumer group with session-to-consumer group mapping rules, or are mapped to a consumer group that is not in the currently active resource plan. OTHER_GROUPS must have a resource plan directive specified in every plan. It cannot be assigned explicitly to sessions through mapping rules.
SYS_GROUP	Consumer group for system administrators. It is the initial consumer group for all sessions created by user accounts SYS or SYSTEM. This initial consumer group can be overridden by session-to-consumer group mapping rules.

26.10.2 Predefined Consumer Group Mapping Rules

Oracle Database includes predefined consumer group mapping rules.

Table 26-19 summarizes the consumer group mapping rules that are predefined in Oracle Database. You can verify these rules by querying the view DBA_RSRC_GROUP_MAPPINGS. You can use the DBMS_RESOURCE_MANAGER.SET_CONSUMER_GROUP_MAPPING procedure to modify or delete any of these mapping rules.

Table 26-19 Predefined Consumer Group Mapping Rules

Attribute	Value	Mapped Consumer Group	Notes
ORACLE_USER	SYS	SYS_GROUP	
ORACLE_USER	SYSTEM	SYS_GROUP	
ORACLE_FUNCTION	BACKUP	BATCH_GROUP	The session is running a backup operation with RMAN. The session is automatically switched to BATCH_GROUP when the operation begins.
ORACLE_FUNCTION	COPY	BATCH_GROUP	The session is running a copy operation with RMAN. The session is automatically switched to BATCH_GROUP when the operation begins.
ORACLE_FUNCTION	DATALOAD	ETL_GROUP	The session is performing a data load operation with Data Pump. The session is automatically switched to ETL_GROUP when the operation begins.

See Also:

"Specifying Session-to-Consumer Group Mapping Rules"

26.10.3 Resource Manager Data Dictionary Views

You can query a set of data dictionary views for information relating to database resource management.

Table 26-20 lists views that are associated with the Resource Manager.

Table 26-20 Resource Manager Data Dictionary Views

View	Description
DBA_RSRC_CONSUMER_GROUP_PRIVS USER_RSRC_CONSUMER_GROUP_PRIVS	DBA view lists all resource consumer groups and the users and roles to which they have been granted. USER view lists all resource consumer groups granted to the user.
DBA_RSRC_CONSUMER_GROUPS	Lists all resource consumer groups that exist in the database.
DBA_RSRC_MANAGER_SYSTEM_PRIVS USER_RSRC_MANAGER_SYSTEM_PRIVS	DBA view lists all users and roles that have been granted Resource Manager system privileges. USER view lists all the users that are granted system privileges for the DBMS_RESOURCE_MANAGER package.
DBA_RSRC_PLAN_DIRECTIVES	Lists all resource plan directives that exist in the database.
DBA_RSRC_PLANS	Lists all resource plans that exist in the database.
DBA_RSRC_GROUP_MAPPINGS	Lists all of the various mapping pairs for all of the session attributes.
DBA_RSRC_MAPPING_PRIORITY	Lists the current mapping priority of each attribute.
DBA_HIST_RSRC_PLAN	Displays historical information about resource plan activation. This view contains AWR snapshots of V$RSRC_PLAN_HISTORY.
DBA_HIST_RSRC_CONSUMER_GROUP	Displays historical statistical information about consumer groups. This view contains AWR snapshots of V$RSRC_CONS_GROUP_HISTORY.
DBA_USERS USER_USERS	DBA view contains information about all users of the database. It contains the initial resource consumer group for each user. USER view contains information about the current user. It contains the current user's initial resource consumer group.
V$RSRC_CONS_GROUP_HISTORY	For each entry in the view V$RSRC_PLAN_HISTORY, contains an entry for each consumer group in the plan showing the cumulative statistics for the consumer group.
V$RSRC_CONSUMER_GROUP	Displays information about active resource consumer groups. This view can be used for tuning.
V$RSRCMGRMETRIC	Displays a history of resources consumed and cumulative CPU wait time (due to resource management) per consumer group for the past minute.
V$RSRCMGRMETRIC_HISTORY	Displays a history of resources consumed and cumulative CPU wait time (due to resource management) per consumer group for the past hour on a minute-by-minute basis. If a new resource plan is enabled, the history is cleared.
V$RSRC_PLAN	Displays the names of all currently active resource plans.
V$RSRC_PLAN_HISTORY	Shows when Resource Manager plans were enabled or disabled on the instance. It helps you understand how resources were shared among the consumer groups over time.

Table 26-20 (Cont.) Resource Manager Data Dictionary Views

View	Description
V$RSRC_SESSION_INFO	Displays Resource Manager statistics for each session. Shows how the session has been affected by the Resource Manager. Can be used for tuning.
V$SESSION	Lists session information for each current session. Specifically, lists the name of the resource consumer group of each current session.

26.11 Operating System CPU Resource Management

Starting with Oracle Database 23ai, Oracle offers an alternative method for CPU resource management, which operates across all database instances on the server.

In previous releases, Oracle offered one method of CPU resource management, which operated within a database instance. Starting with Oracle Database 23ai, you can set RESOURCE_MANAGER_CPU_SCOPE to control the scope of CPU resource management. This parameter allows you to choose between the two control methods.

Setting RESOURCE_MANAGER_CPU_SCOPE to SERVER_WIDE enables server-level inter-instance CPU resource management. Introduced in Oracle Database 23ai, this method can be used when multiple database instances share a Linux server. It enables CPU resource management across all database instances on the server. With this method, Oracle automatically configures Linux control groups (cgroups) with the desired CPU sharing and CPU utilization limits, and then places database sessions into the appropriate cgroups.

In order to use this method, the following system and database requirements must be met:

- This method requires the Linux operating system. It is supported on Oracle Autonomous Database, Oracle Database Exadata Cloud@Customer, Oracle Database Exadata Cloud Service, and on-premises Oracle Exadata systems. This method is not supported on non-Exadata Linux systems.
- The server operating system must be Unbreakable Enterprise Kernel (UEK) version 5 or later.

You can configure this method as follows:

- Oracle Autonomous Database: All system and database requirements are installed and configured by default. RESOURCE_MANAGER_CPU_SCOPE is set to SERVER_WIDE by default.
- Oracle Database Exadata Cloud@Customer: All system and database requirements are installed and configured by default. RESOURCE_MANAGER_CPU_SCOPE is set to INSTANCE_ONLY by default. You must set it to SERVER_WIDE if you want to use this method.
- Oracle Database Exadata Cloud Service and on-premises Oracle Exadata systems: The Exadata image meets all system and database requirements. RESOURCE_MANAGER_CPU_SCOPE is set to INSTANCE_ONLY by default. You must set it to SERVER_WIDE if you want to use this method.

When you specify the SERVER_WIDE setting, the value of the PROCESSOR_GROUP_NAME initialization parameter is ignored.

Setting RESOURCE_MANAGER_CPU_SCOPE to INSTANCE_ONLY enables CPU resource management only within the database instance. This is the method of CPU resource management used in releases prior to Oracle Database 23ai. It is supported on all Oracle Databases.

Related Topics

- RESOURCE_MANAGER_CPU_SCOPE
- V_RSRC_PLAN
- V_RSRC_PLAN_HISTORY